ModelArts

Best Practices

Issue	01
Date	2024-01-09

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.

Contents

1 Official Samples

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.

ModelArts Permissions (Basic)

Sample	Function	Scenario	Description
Scenarios	IAM permissions and global configurations	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.

Samples for Custom Algorithms in Model Development (Advanced)

Table 1-1 Custom algorithm samples

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

2 Permissions Management

2.1 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.

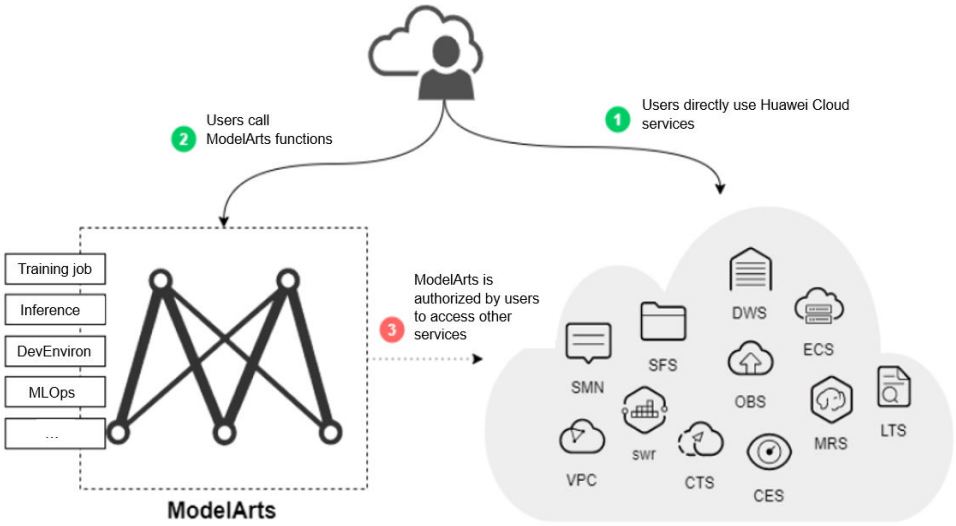
- You are an enterprise user, and
  - There are multiple departments in your enterprise, and you need to control users' permissions so that users in different departments can access only their dedicated resources and functions.
  - There are multiple roles (such as administrators, algorithm developers, and application O&M personnel) in your enterprise. You need them to use only specific functions.
  - There are logically multiple environments (such as the development environment, pre-production environment, and production environment) and are isolated from each other. You need to control users' permissions on different environments.
  - 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 2-1 Permissions management



Exposed ModelArts functions are controlled through 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.

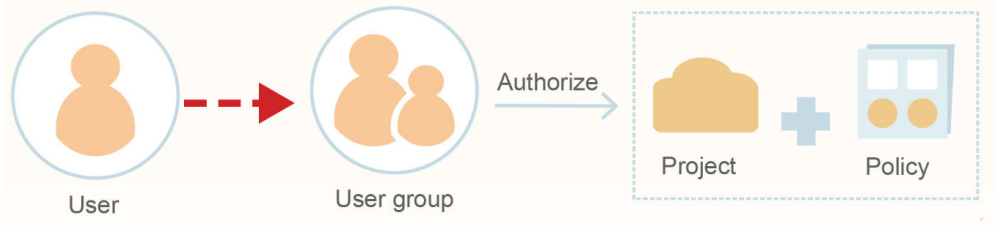
ModelArts Permissions Management

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.

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 2-1 lists all the preset system-defined policies supported by ModelArts.

Table 2-1 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.

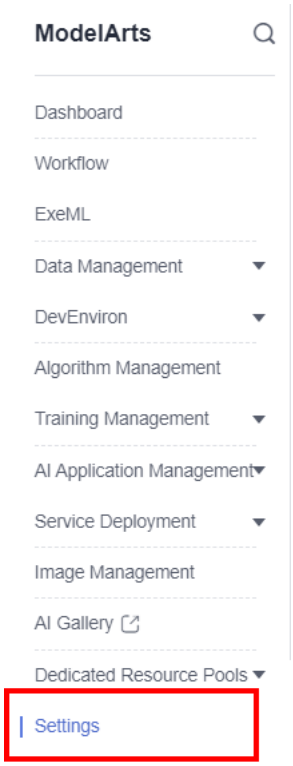
For details about the basic concepts and operations of agencies, see **Cloud Service Delegation**.

To simplify agency authorization, ModelArts supports automatic agency authorization configuration. You only need to configure an agency for yourself or specified users on the **Global Configuration** 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.

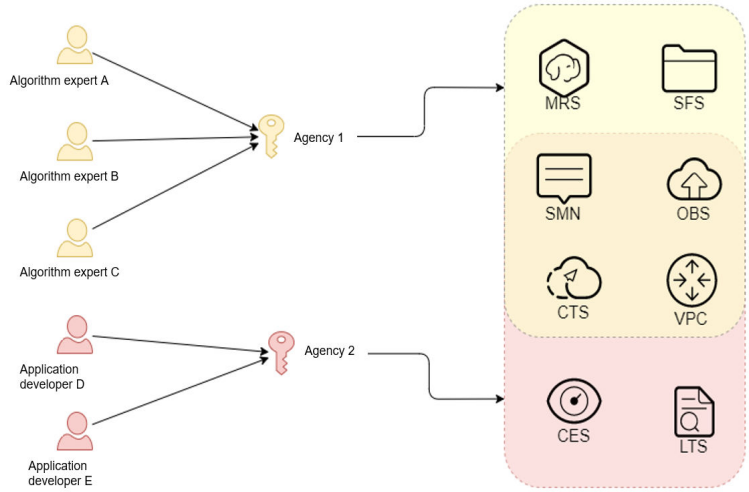
Figure 2-2 Settings



On the **Global Configuration** 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 2-3 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 **Global Configuration** 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.

2.2 Permission Management Mechanisms

2.2.1 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 ModelArts resources, Identity and Access Management (IAM) is a good choice for fine-grained permissions management. IAM provides identity authentication, permissions management, and access control, helping you securely access Huawei Cloudcloud resources. If your Huawei accountaccount 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 control access to specific Huawei Cloudcloud 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?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 APIG resources in the selected projects. If you set **Scope** to **All resources**, the users have permissions for APIG 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 2-2 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 2-2 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 does 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 2-3 Roles or policies that are required for performing operations on the ModelArts console

Console Function	Dependency	Role/Policy Required
Data management	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	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 Administrator
	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
Training management	OBS	OBS Administrator
	Simple Message Notification (SMN)	SMN Administrator
	CTS	CTS Administrator
	SFS Turbo	SFS Turbo ReadOnlyAccess
	SWR	SWR Administrator
	AOM	AOM FullAccess
	KMS	KMS CMKFullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Workflow	OBS	OBS Administrator
	CTS	CTS Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
ExeML	OBS	OBS Administrator
	CTS	CTS Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI application management	OBS	OBS Administrator
	Enterprise Project Management Service (EPS)	EPS FullAccess
	CTS	CTS Administrator
	SWR	SWR Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access

Console Function	Dependency	Role/Policy Required
Service deployment	OBS	OBS Administrator
	Cloud Eye Service (CES)	CES ReadOnlyAccess
	SMN	SMN Administrator
	EPS	EPS FullAccess
	CTS	CTS Administrator
	Log Tank Service (LTS)	LTS FullAccess
	Virtual Private Cloud (VPC)	VPC FullAccess
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
AI Gallery	OBS	OBS Administrator
	CTS	CTS Administrator
	SWR	SWR Administrator
	ModelArts	ModelArts CommonOperations ModelArts Dependency Access
Dedicated resource pool	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
	ModelArts	ModelArts FullAccess
	Billing Center	BSS Administrator

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**.

You can create custom policies in either of the following ways:

- 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**. The following lists examples of common ModelArts custom 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:

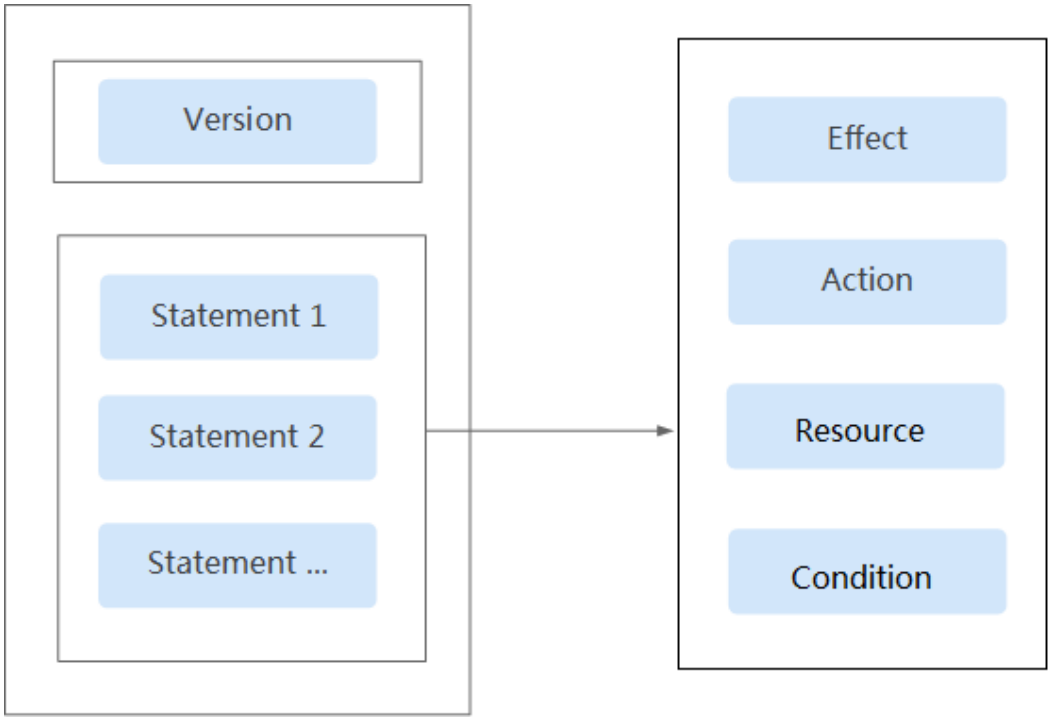
```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "modelarts:service:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lts:logs:list"
            ]
        }
    ]
}
```

Policy Fields in JSON Format

Policy Structure

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

Figure 2-4 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 2-4 Policy parameters

Parameter		Description	Value
Version		Policy version	**1.1**: indicates policy-based access control.
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 2-5 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
trainJobVersion	Version of a training job (supported by old-version training jobs that will be discontinued soon)
trainConfig	Configuration of a training job (supported by old-version training jobs that will be discontinued soon)
tensorboard	Visualization job of training results (supported by old-version training jobs that will be discontinued soon)
model	Models
service	Real-time service
nodeservice	Edge service
workspace	Workspace
dataset	Dataset
dataAnnotation	Dataset labels

Resource Type	Description
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**

2.2.2 Agencies and Dependencies

Function Dependency

Function Dependency Policies

When using ModelArts to develop algorithms or manage training jobs, you are required to use other Cloud services. For example, before submitting a training job, select an OBS path 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.

Table 2-6 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.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
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 2-7 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 2-8 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 aom:metric:get aom:metric:list aom:alarm:list	Start, stop, create, delete, and update an instance.
Dynamically mounting storage	ModelArts	modelarts:notebook:listMountedStorages modelarts:notebook:mountStorage modelarts:notebook:getMountedStorage modelarts:notebook:umountStorage	Dynamically mount storage.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	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.
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	

Application Scenario	Dependent Service	Dependent Policy	Supported Function
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 2-9 Managing training jobs

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Training management	ModelArts	modelarts:trainJob:* modelarts:trainJobLog:* modelarts:aiAlgorithm:* modelarts:image:list	Create a training job and view training logs.
		modelarts:workspace:getQuotas	Obtain a workspace quota. This policy is required if the **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:repository:listTags swr:repository:getRepository swr:repository:listRepositories	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 2-10 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
Managing AI applications	ModelArts	modelarts:model:list modelarts:model:get modelarts:model:create modelarts:model:delete modelarts:model:update	Manage ModelArts AI applications 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 the 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 2-11 Managing AI applications

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing AI applications	SWR	swr:repository:deleteRepository swr:repository:deleteTag swr:repository:getRepository swr:repository:listTags	Import a model from a custom image. Use a custom engine when importing a model from OBS.

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 a template. Specify an OBS path for model conversion.

Table 2-12 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.

Table 2-13 Managing datasets

Application Scenario	Dependent Service	Dependent Policy	Supported Function
Managing datasets and labels	OBS	obs:bucket:ListBucket (Listing objects in a bucket) 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:object:DeleteObject (Deleting an object or batch deleting objects) obs:bucket:HeadBucket (Obtaining bucket metadata) obs:bucket:GetBucketAcl (Obtaining a bucket ACL) obs:bucket:PutBucketAcl (Configuring a bucket ACL) obs:bucket:GetBucketPolicy (Obtaining a bucket policy) obs:bucket:PutBucketPolicy (Configuring a bucket policy) obs:bucket:DeleteBucketPolicy (Deleting a bucket policy) obs:bucket:PutBucketCORS (Configuring or deleting CORS rules of a bucket) obs:bucket:GetBucketCORS (Obtaining the CORS rules of a bucket) obs:object:PutObjectAcl (Configuring an object ACL)	Manage datasets in OBS. Label OBS data. Create a data management job.
Managing table datasets	DLI	dli:database:displayAllDatabases dli:database:displayAllTables dli:table:describe_table	Manage DLI data in a dataset.
Managing table datasets	DWS	dws:openAPICluster:list dws:openAPICluster:getDetail	Manage DWS data in a dataset.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
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.
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 2-14 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. Dependent permissions must be configured in the IAM project view.

Application Scenario	Dependent Service	Dependent Policy	Supported Function
	ECS	ecs:availabilityZones:list	Show AZs. Dependent permissions must be configured in the IAM project view.
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	Create and delete ModelArts networks, and interconnect VPCs. Dependent permissions must be configured in the IAM project view.

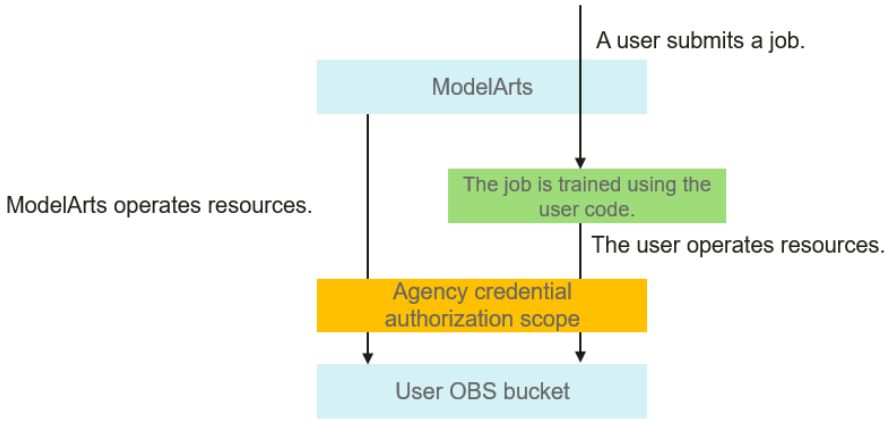
Application Scenario	Dependent Service	Dependent Policy	Supported Function
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	Interconnect your network with SFS Turbo. Dependent permissions must be configured in the IAM project view.
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 to run jobs, 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 2-5 Agency authorization



As shown in **Figure 2-5**, 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 **Global Configuration** 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 in the ModelArts global configuration 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 2-15 Basic agency authorization for a development environment

Application Scenario	Dependent Service	Agency Authorization	Description	Configuration Suggestion
JupyterLab	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	Use OBS to upload and download data in JupyterLab through ModelArts notebook.	Recommended

Application Scenario	Dependent Service	Agency Authorization	Description	Configuration Suggestion
Development environment monitoring	AOM	aom:alarm:put	Call the AOM API to obtain monitoring data and events of notebook instances and display them in ModelArts notebook.	Recommended

Table 2-16 Basic agency authorization for training jobs

Application Scenario	Dependent Service	Agency Authorization	Description
Training jobs	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	Download data, models, and code before starting a training job. Upload logs and models when a training job is running.

Table 2-17 Basic agency authorization for deploying services

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.

Application Scenario	Dependent Service	Agency Authorization	Description
Batch services	OBS	obs:bucket:ListBucket obs:object:GetObject obs:object:PutObject	Run a batch service.
Edge services	IEF	ief:deployment:list ief:deployment:create ief:deployment:update ief:deployment:delete ief:node:createNodeCert ief:iefInstance:list ief:node:list	Deploy an edge service using IEF.

Table 2-18 Basic agency authorization for managing data

Application Scenario	Dependent Service	Agency Authorization	Description
Dataset and data labeling	OBS	obs:object:GetObject obs:object:PutObject obs:object:DeleteObject obs:object:PutObjectAcl obs:bucket:ListBucket obs:bucket:HeadBucket obs:bucket:GetBucketAcl obs:bucket:PutBucketAcl obs:bucket:GetBucketPolicy obs:bucket:PutBucketPolicy obs:bucket:DeleteBucketPolicy obs:bucket:PutBucketCORS obs:bucket:GetBucketCORS	Manage datasets in an OBS bucket.
Labeling data	ModelArts inference	modelarts:service:get modelarts:service:create modelarts:service:update	Perform auto labeling based on ModelArts inference.

Table 2-19 Basic agency authorization for managing dedicated resource pools

Application Scenario	Dependent Service	Agency Authorization	Description
Network management (New version)	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	Create and delete ModelArts networks, and interconnect VPCs. Dependent permissions must be configured in the IAM project view.
	SFS Turbo	sfsturbo:shares:addShareNic sfsturbo:shares:deleteShareNic sfsturbo:shares:showShareNic sfsturbo:shares:listShareNics	Interconnect your network with SFS Turbo. Dependent permissions must be configured in the IAM project view.

Application Scenario	Dependent Service	Agency Authorization	Description
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. Dependent permissions must be configured in the IAM project view.
Managing resource pools	ECS	ecs:availabilityZones:list	Show AZs. Dependent permissions must be configured in the IAM project view.

2.2.3 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.

2.3 Configuration Practices in Typical Scenarios

2.3.1 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.

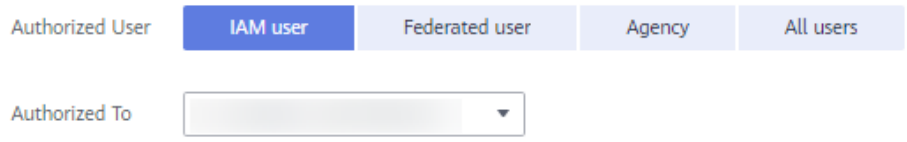
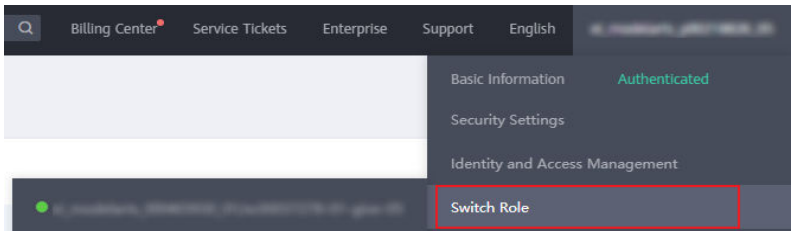
Constraints

- 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 management console. In the navigation pane on the left, choose **Settings**. The **Global Configuration** page is displayed.
2. Click **Add Authorization**. On the **Add Authorization** page that is displayed, configure the parameters.

Table 2-20 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**.
Authorized To	This parameter is not displayed when **Authorized User** is set to **All users**. • **IAM user**: Select an IAM user and configure an agency for the IAM user. **Figure 2-6** Selecting an IAM user  • **Federated user**: Enter the username or user ID of the target federated user. **Figure 2-7** Selecting a federated user  • **Agency**: Select an agency name. You can use account A to create an agency and configure the agency for account B. When using account B, you can switch the role in the upper right corner of the console to account A and use the agency permissions of account A. **Figure 2-8** Switch Role 

Parameter	Description
Agency	• **Use existing**: If there are agencies in the list, select an available one to authorize the selected user. Click the drop-down arrow next to an agency name to view its permission details. • **Add agency**: If there is no available agency, create one. If you use ModelArts for the first time, select **Add agency**.
Add agency > Agency Name	The system automatically creates a changeable agency name.
Add agency > Authorization Method	• **Role-based**: A coarse-grained IAM authorization strategy to assign permissions based on user responsibilities. Only a limited number of service-level roles are available. When using roles to grant permissions, assign other roles on which the permissions depend to take effect. Roles are not ideal for fine-grained authorization and secure access control. • **Policy-based**: A fine-grained authorization tool that defines permissions for operations on specific cloud resources under certain conditions. This type of authorization is more flexible and ideal for secure access control. For details about roles and policies, see **Basic Concepts**.
Add agency > Permissions > Common User	**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 **Global Configuration** page. Click **View Permissions** in the **Authorization Content** column to view the permission details.

Figure 2-9 View Permissions

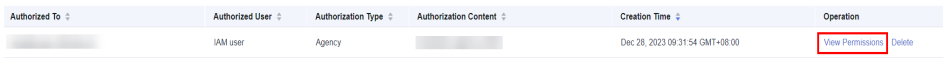
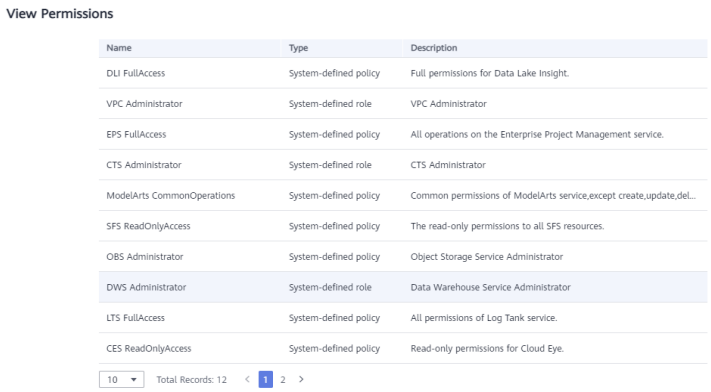


Figure 2-10 Common user permissions



2.3.2 Assigning Basic Permissions for Using ModelArts

2.3.2.1 Scenarios

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 2-21 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

Target Service	Description	IAM Permission	Mandatory
	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
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

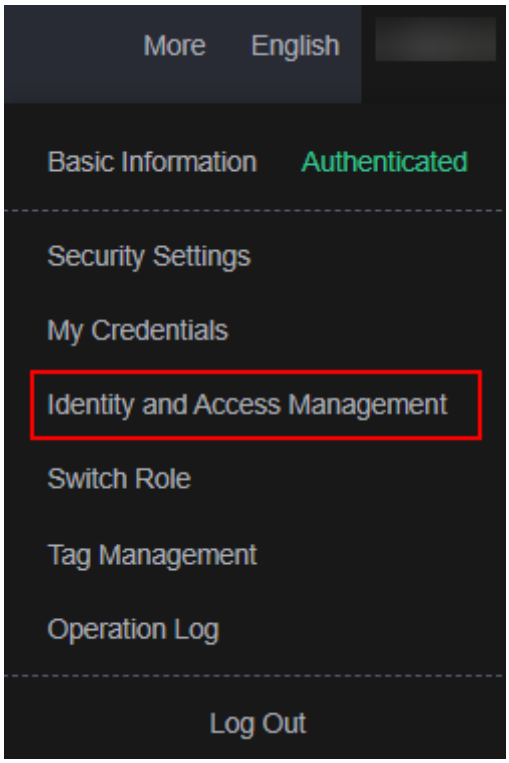
Target Service	Description	IAM Permission	Mandatory
Simple Message Notification (SMN)	Assign permissions to IAM users for using SMN. SMN is used with Cloud Eye.	SMN FullAccess	No
Virtual Private Cloud (VPC)	During the creation of a dedicated resource pool for ModelArts, IAM users require VPC permissions so that they can customize networks.	VPC FullAccess	No
Scalable File Service (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

2.3.2.2 Step 1 Create a User Group and Add 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.

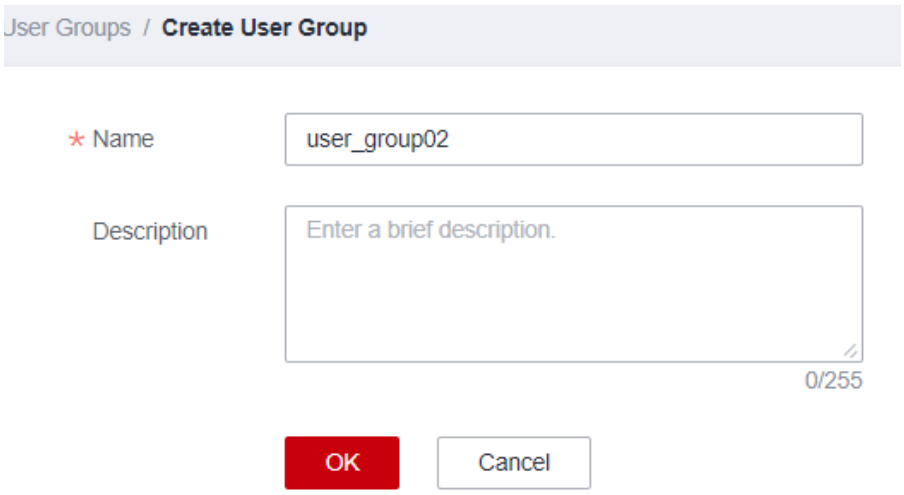
1. Log in to the management console as a tenant user, hover over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.

Figure 2-11 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**.

Figure 2-12 Creating a user group

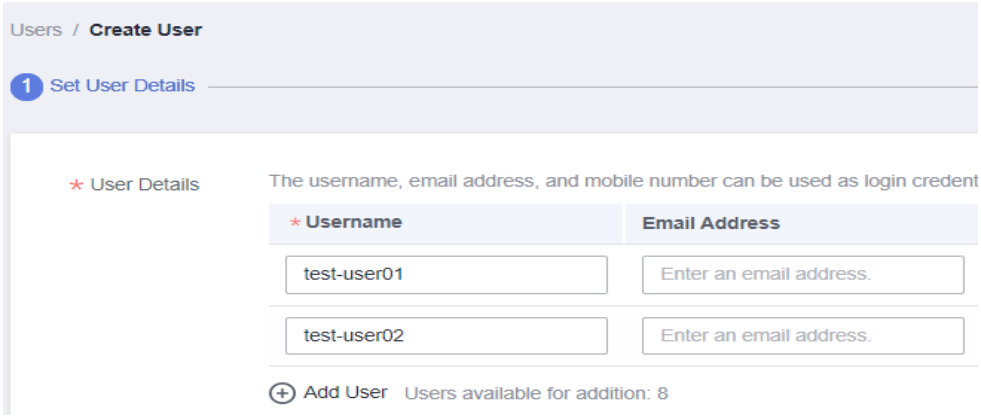


After the user group is created, the system automatically switches to the user group list. Then, you can add existing IAM users to the user group through user group management. If there is no existing IAM user, create users and add them to the user group.

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.

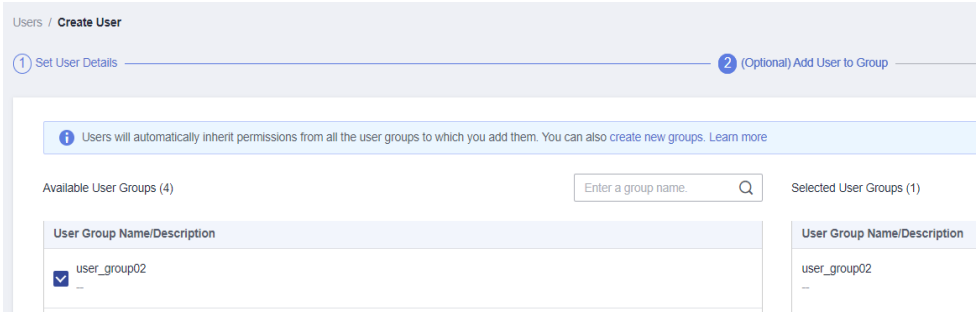
   Set parameters as prompted and click **Next**.

   Figure 2-13 Creating multiple users



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

   Figure 2-14 Adding users to the target user group



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

2.3.2.3 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 2-15 Authorize

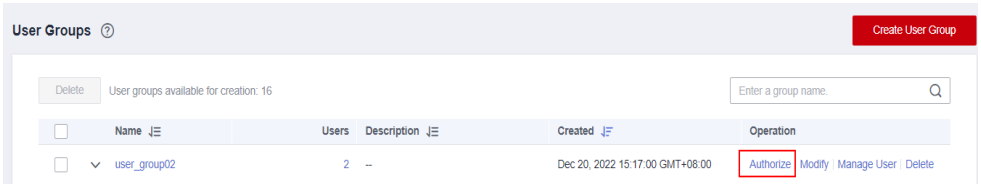
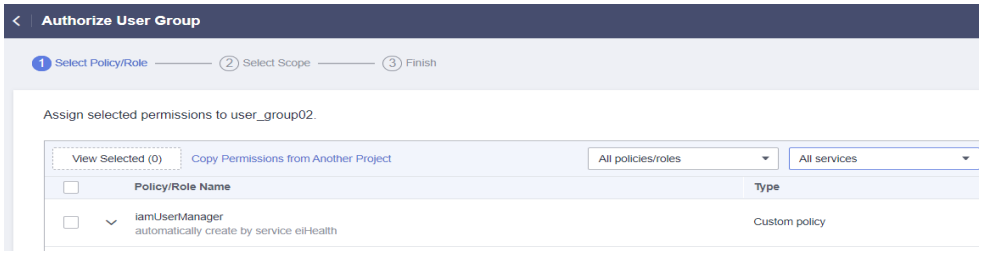


Figure 2-16 Authorize User Group

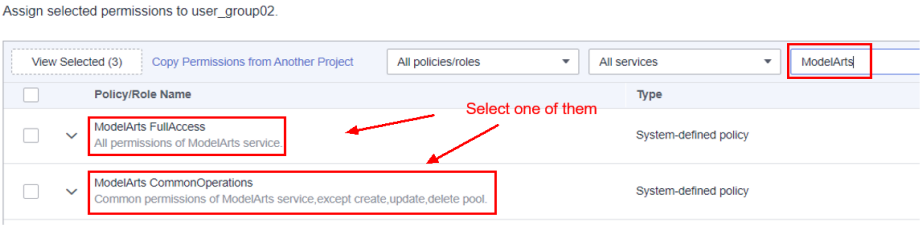


2. Before assigning permissions, learn about minimum permissions requirements of each ModelArts module, as shown in **Table 2-21**.
3. Assign permissions for using ModelArts. Search for **ModelArts**. Select either **ModelArts FullAccess** or **ModelArts CommonOperations**.

   The differences between the options are as follows:

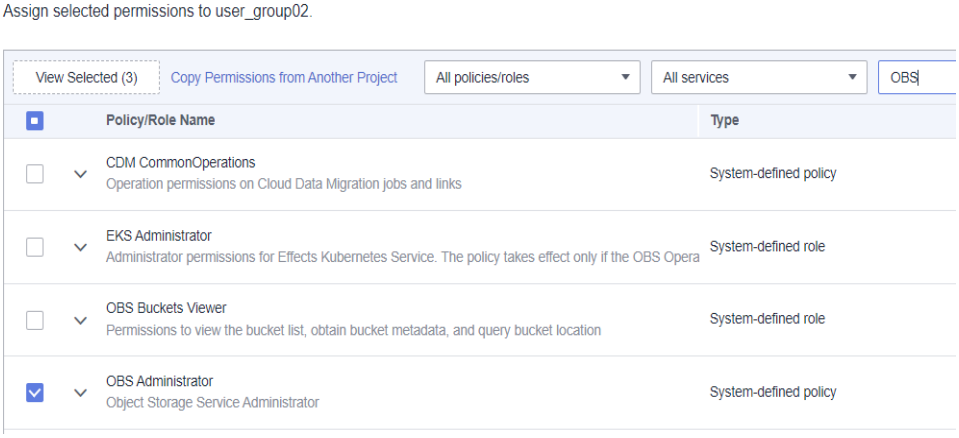
   - The users with the ModelArts CommonOperations permission can only use resources, but cannot create, update, or delete any dedicated resource pool. You are advised to assign this permission to IAM users.
   - The users with the ModelArts FullAccess permission have all access permissions, including creating, updating, and deleting dedicated resource pools. Exercise caution when selecting this option.

Figure 2-17 Assigning permissions for using ModelArts



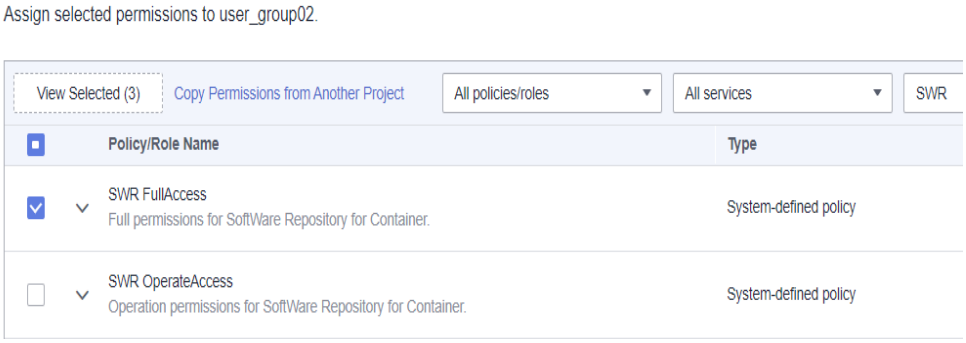
4. Assign permissions for using OBS. Search for **OBS** and select **OBS Administrator**. ModelArts training jobs use OBS for forwarding data. Therefore, the permissions for using OBS are required.

Figure 2-18 Assigning permissions for using OBS



5. Assign permissions for using SWR. Search for **SWR** and select **SWR FullAccess**. ModelArts custom images require the SWR FullAccess permission.

Figure 2-19 Assigning permissions for using SWR



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.

Figure 2-20 DEW key management permission

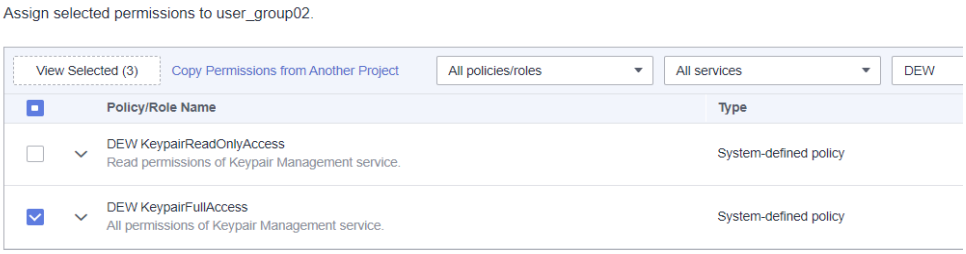
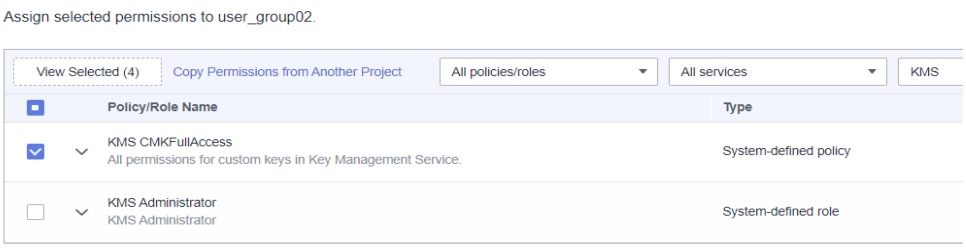


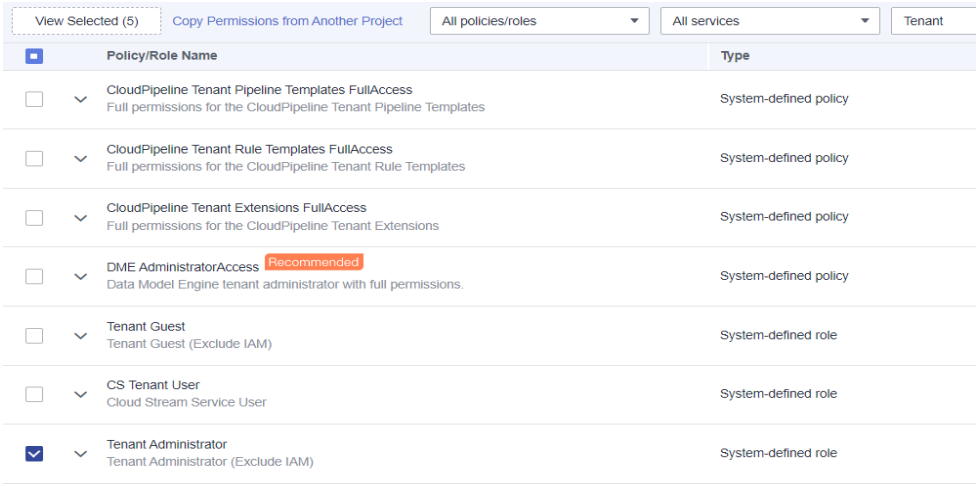
Figure 2-21 KMS key management permission



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 the ModelArts service. Exercise caution when assigning the Tenant Administrator permission.

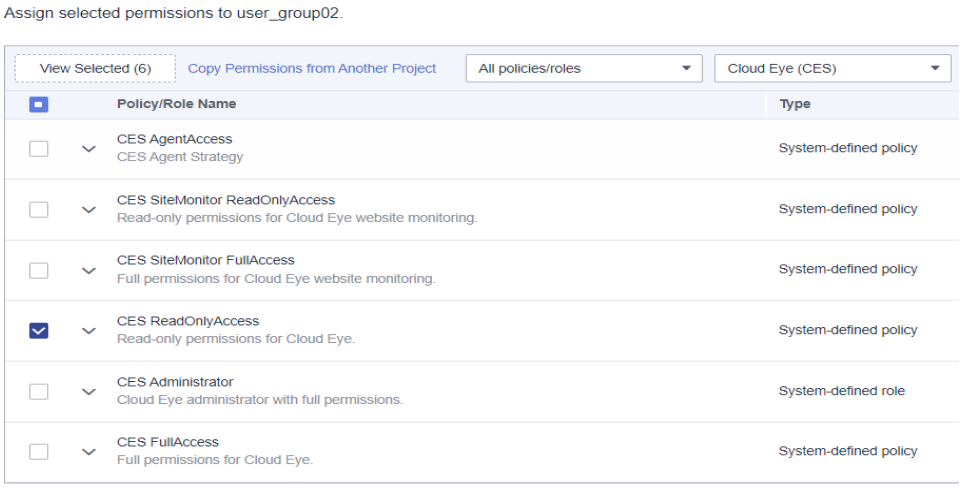
Figure 2-22 Assigning permissions for using IEF



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**.

   Figure 2-23 CES ReadOnlyAccess

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

Figure 2-24 Assigning alarm monitoring permissions

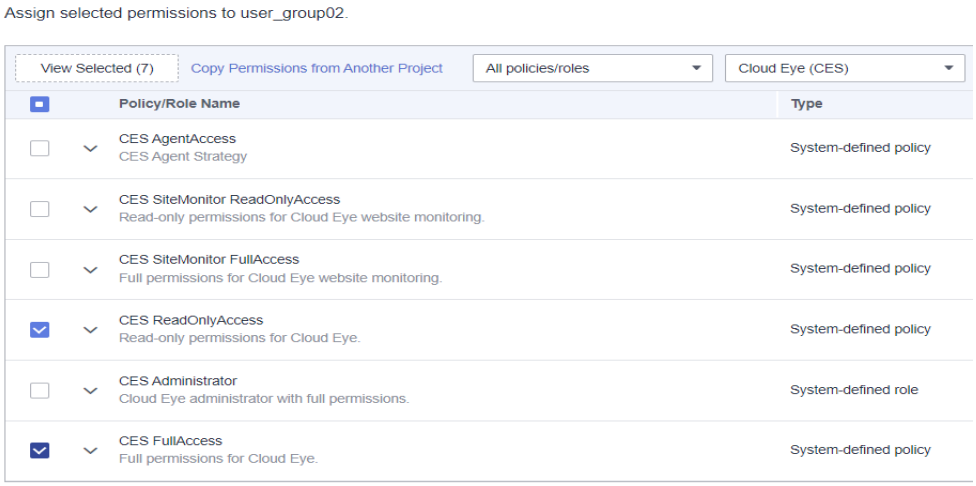
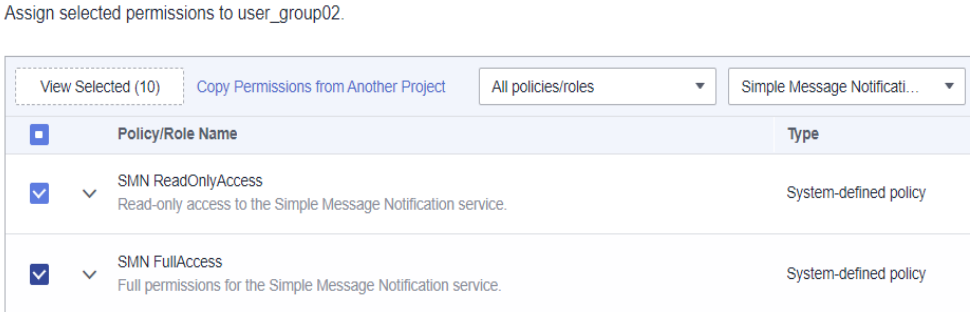
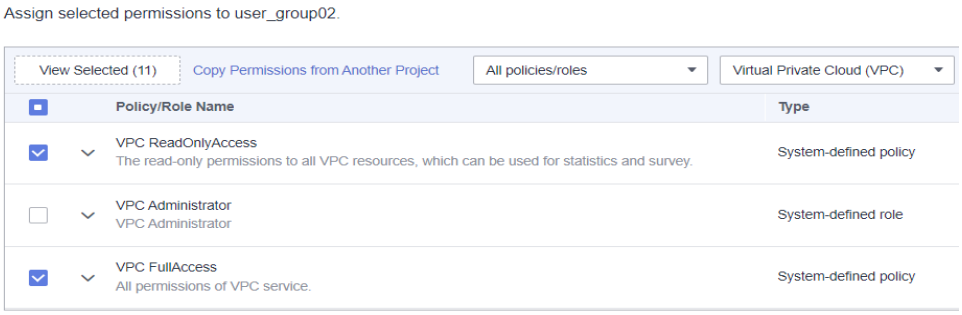


Figure 2-25 Assigning permissions for using SMN



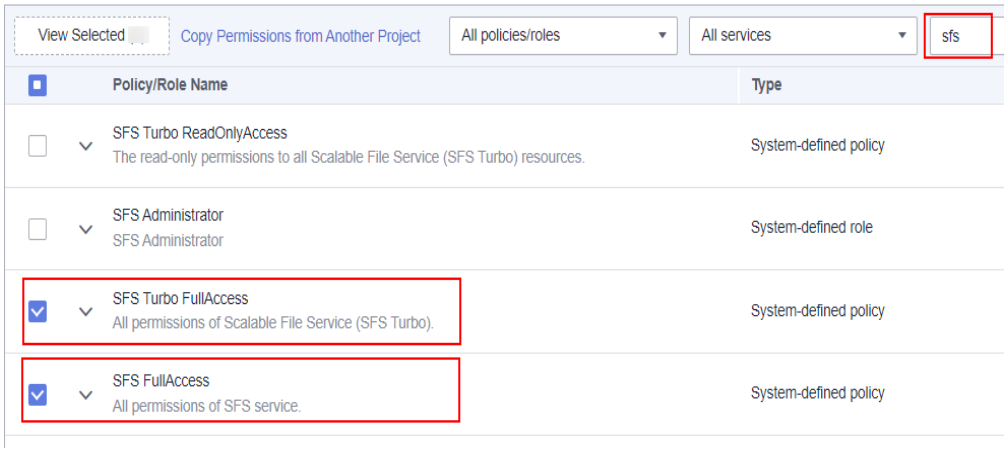
9. (Optional) Assign permissions for using VPC. To enable custom network configuration when creating a dedicated resource pool, assign permissions for using VPC.

Figure 2-26 Assigning permissions for using VPC



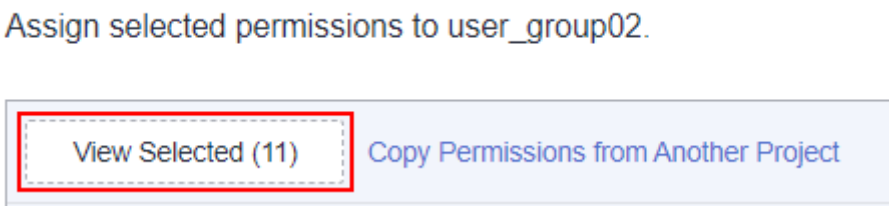
10. (Optional) Assign permissions for using SFS and SFS Turbo. To mount an SFS system to a dedicated resource pool as the storage for the development environment or training, assign the permission to use the SFS system.

Figure 2-27 Assigning permissions for using SFS and SFS Turbo



11. Click **View Selected** in the upper left corner and confirm the selected permissions.

Figure 2-28 Viewing 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.

2.3.2.4 Step 3 Configure Agent-based ModelArts Access Authorization for the User

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. Therefore, in this example, the administrator needs to configure access authorization for all the IAM 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, choose **Settings**. The **Global Configuration** page is displayed.
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.

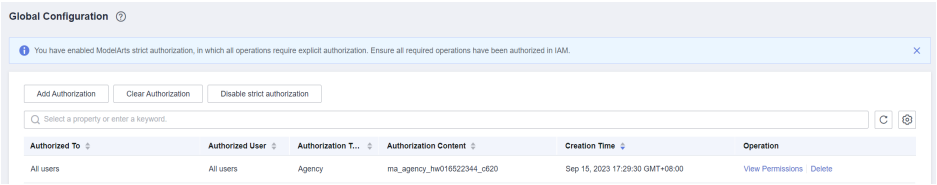
- **Common User**: You can use basic ModelArts functions, for example, accessing data and creating and managing training jobs, but not to manage resources. Select this option generally.
- **Custom**: You can flexibly assign permissions to the created agency. Select this option for refined permissions management. You can select permissions from the permission list as required.

Figure 2-29 Common user permissions

Policy/Role	Module	Description
OBS Administrator	Data Management | DevEnviron | Training Managem...	Object Storage Service Administrator
DLI FullAccess	Data Management	Full permissions for Data Lake Insight.
MRS Administrator	Data Management	MRS Administrator
DWS Administrator	Data Management	Data Warehouse Service Administrator
VPC Administrator	Training Management | AI Application Management |...	VPC Administrator
CES ReadOnlyAccess	AI Application Management | Service Deployment	Read-only permissions for Cloud Eye.
SMN Administrator	Training Management | AI Application Management |...	Simple Message Notification Administrator
EPS FullAccess	Training Management | AI Application Management |...	All operations on the Enterprise Project Management...
CTS Administrator	Data Management | DevEnviron | Training Managem...	CTS Administrator
SFS ReadOnlyAccess	Training Management	The read-only permissions to all Scalable File Service ...
LTS FullAccess	AI Application Management | Service Deployment	All permissions of Log Tank service.
ModelArts CommonOperations	Data Management | DevEnviron | Training Managem...	Common permissions of ModelArts service,except cre...
ECS FullAccess	DevEnviron	All permissions of ECS service.
SWR Admin	AI Application Management | Service Deployment | A...	Software Repository Admin

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

Figure 2-30 Configured agency authorization



2.3.2.5 Step 4 Verify User Permissions

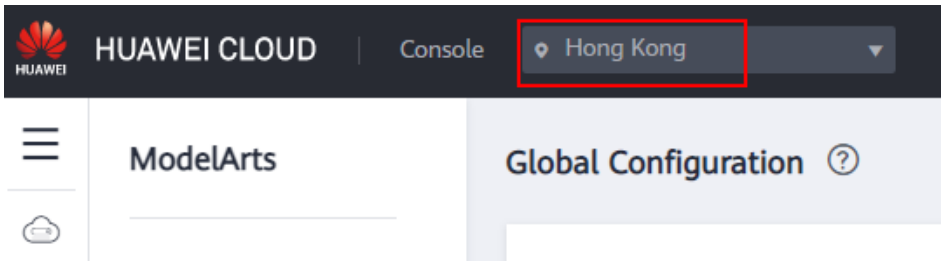
It takes 15 to 30 minutes for the permissions configured in **4** to take effect. Therefore, wait for 30 minutes after the configuration and then verify the configuration.

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.

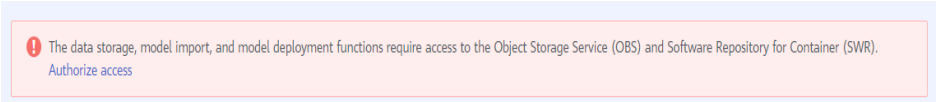
   a. Select the target region in the upper left corner, which must be the same as that in the authorization configuration.



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 the information shown in the following figure is displayed, the ModelArts agency authorization has not been configured. In this case, follow the instructions provided in **Step 3 Configure Agent-based ModelArts Access Authorization for the User** to configure the authorization.

Figure 2-31 Insufficient permissions



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.

2.3.3 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. By default, a developer account does not have the **te_admin** permission. The tenant

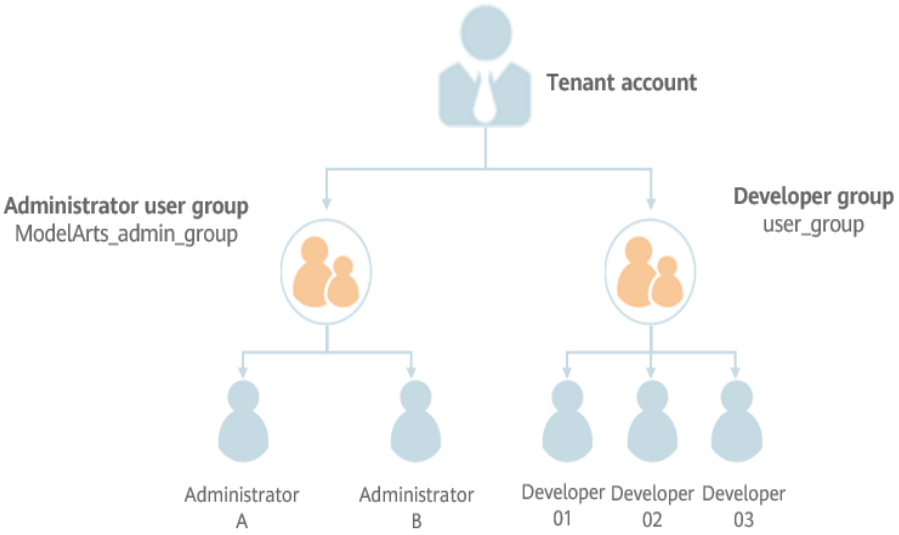
account must configure the required permissions. This section uses notebook as an example to describe how to assign different permissions to administrators and developers through custom policies.

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 2-32 Account relationships



Configuring Permissions for an Administrator

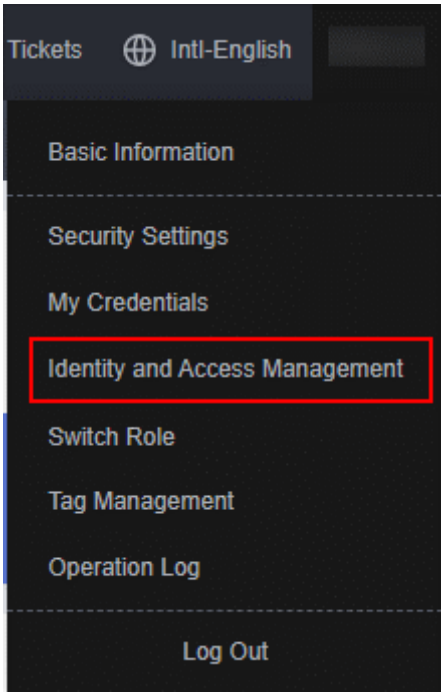
Assign full control permissions to administrators for using ModelArts dedicated resource pools and all notebook instances. The procedure is as follows:

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 Create a User Group and Add Users to the User Group**.

Step 2 Create a custom policy.

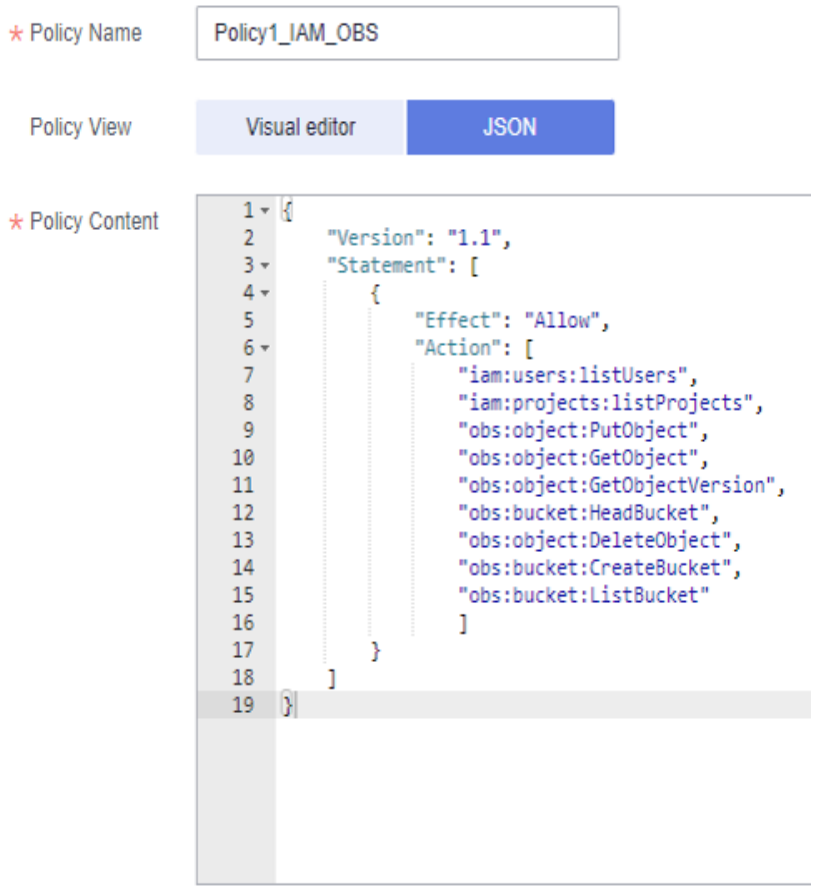
1. Log in to the management console using an administrator account, hover over your username in the upper right corner, and click **Identity and Access Management** from the drop-down list to switch to the IAM management console.

Figure 2-33 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**.

Figure 2-34 Custom policy 1



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**.

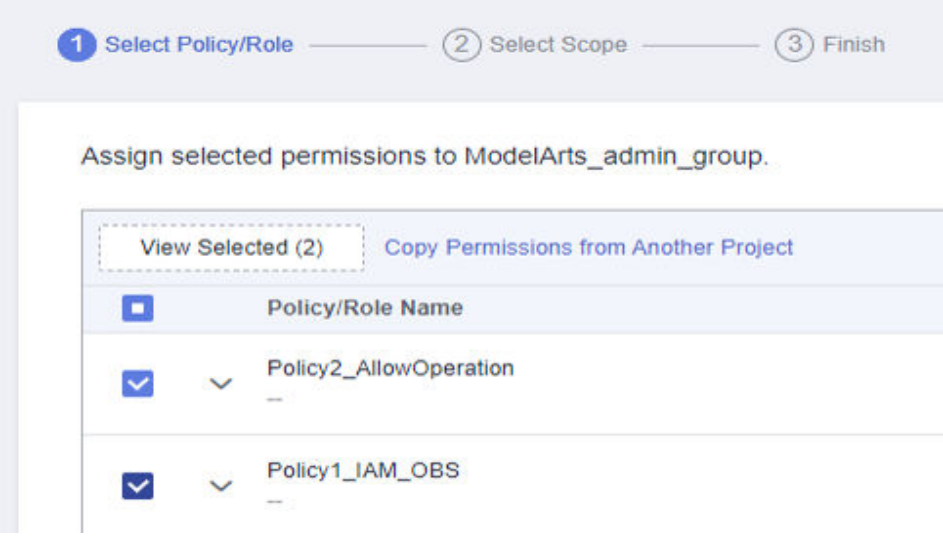
The custom policy **Policy2_AllowOperation** is as follows, which grants the user the permissions to perform operations on dependent services ECS, SWR, MRS, and SMN as well as ModelArts. You can directly copy and paste the content.

```
{
   "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**.

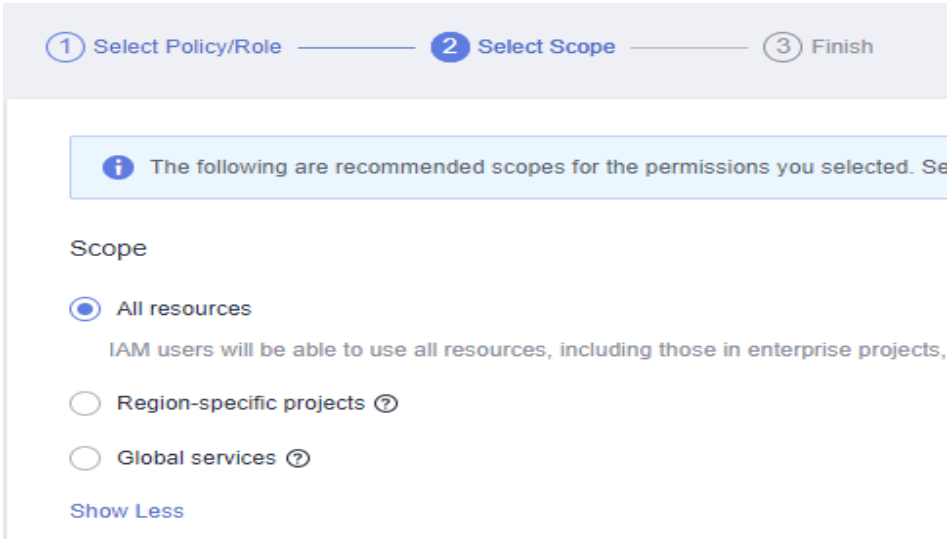
1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **ModelArts_admin_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS** and **Policy2_AllowOperation**. Click **Next**.

   Figure 2-35 Select Policy/Role

2. Specify the scope as **All resources** and click **OK**.

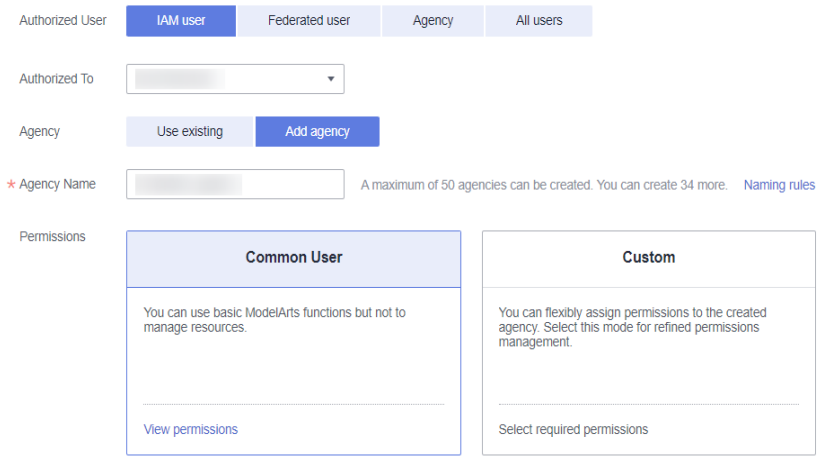
Figure 2-36 Select Scope



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 management console using a tenant account. In the navigation pane, choose **Settings**. The **Global Configuration** page is displayed.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select an administrator account for **Authorized To**, click **Add agency**, and select **Common User** for **Permissions**. Permissions control is not required for administrators, so use default setting **Common User**.

   Figure 2-37 Configuring authorization for an administrator



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

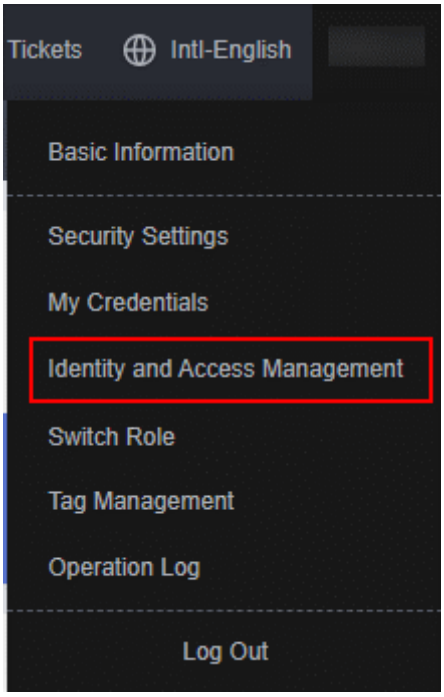
Use IAM for fine-grained control of developer permissions. The procedure is as follows:

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 Create a User Group and Add 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 2-38 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**.

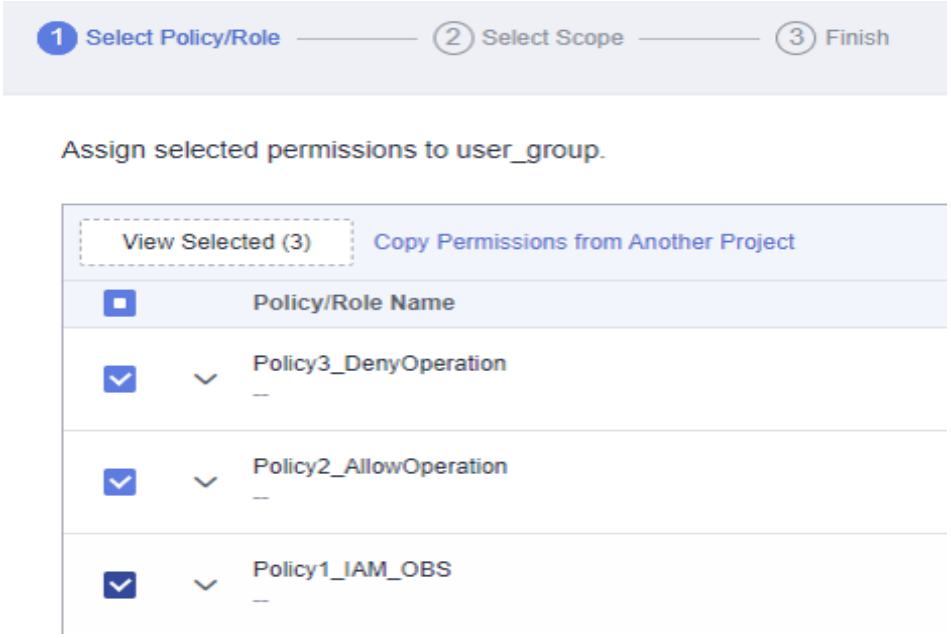
   The custom policy **Policy3_DenyOperation** is as follows. You can copy and paste the content.


```
{
   "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**.

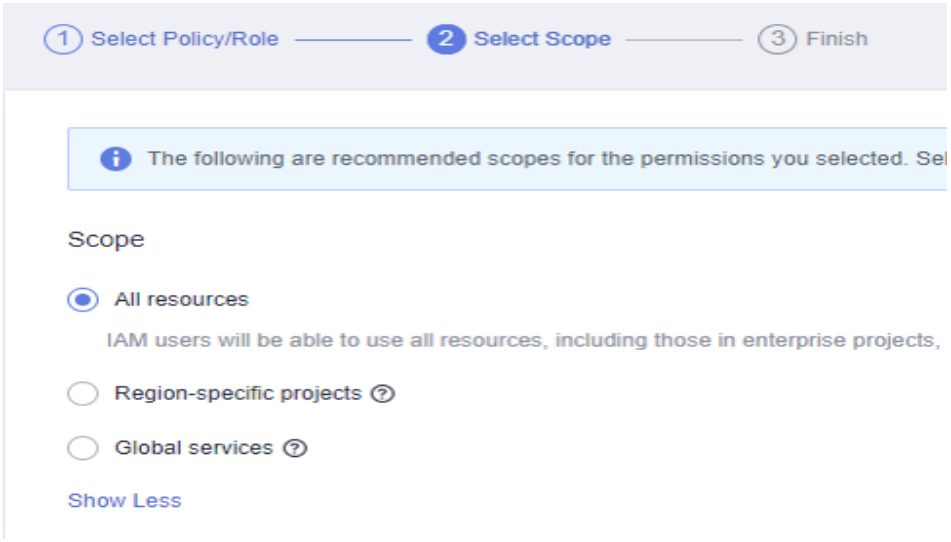
1. In the navigation pane of the IAM console, choose **User Groups**. On the **User Groups** page, locate the row that contains **user_group**, click **Authorize** in the **Operation** column, and select **Policy1_IAM_OBS**, **Policy2_AllowOperation**, and **Policy3_DenyOperation**. Click **Next**.

   Figure 2-39 Select Policy/Role

2. Specify the scope as **All resources** and click **OK**.

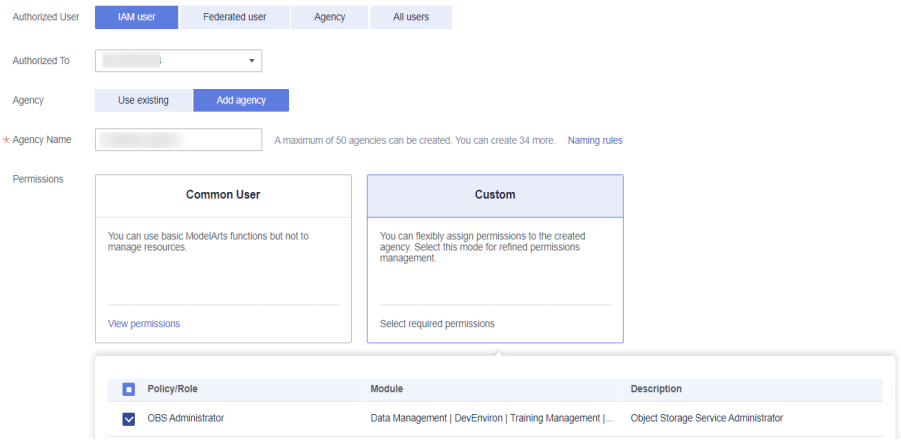
Figure 2-40 Select Scope



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 management console using a tenant account. In the navigation pane, choose **Settings**. The **Global Configuration** page is displayed.
2. Click **Add Authorization**. On the **Add Authorization** page, set **Authorized User** to **IAM user**, select a developer account for **Authorized To**, add a new agency, name it **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.

   Figure 2-41 Configuring authorization for a developer



3. Click **Create**.
4. On the **Global Configuration** 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.

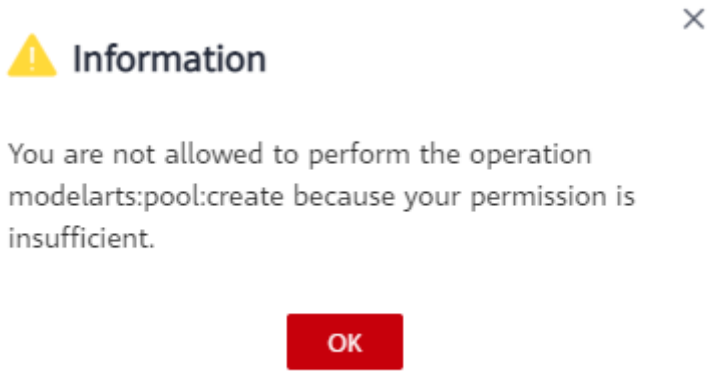
Step 5 Test developer permissions.

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.

   Figure 2-42 Insufficient permissions



----End

2.3.4 Viewing the Notebook Instances of All IAM Users Under One Tenant Account

Any IAM user granted with the **listAllNotebooks** and **listUsers** permissions can click **View all** on the notebook page to view the instances of all users in the current IAM project.

NOTE

Users granted with these permissions can also access OBS and SWR of all users in the current IAM project.

Assigning the Required Permissions

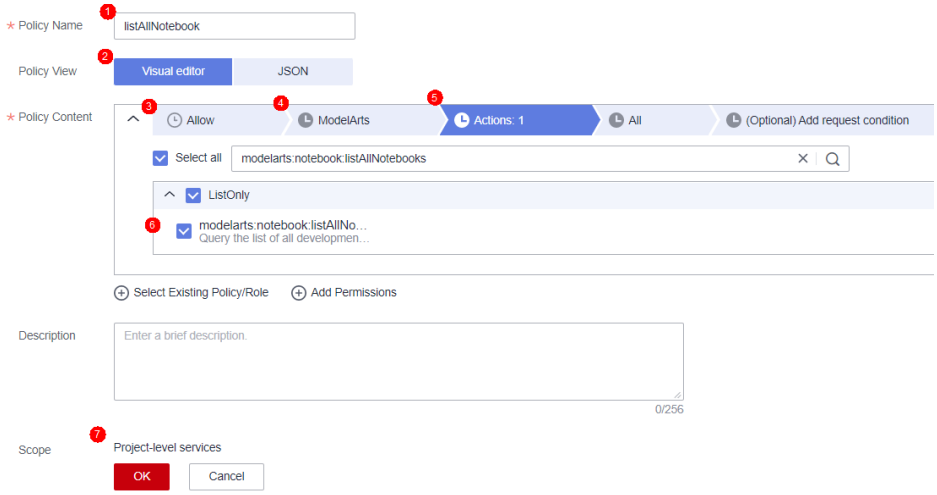
1. Log in to the management console as a tenant user, hover the cursor over your username in the upper right corner, and choose **Identity and Access Management** from the drop-down list to switch to the IAM management console.
2. On the IAM console, choose **Permissions** > **Policies/Roles** from the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

   Policy 1: Create a policy that allows users to view all notebook instances of an IAM project, as shown in **Figure 2-43**.

   - **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 2-43 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**. 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.

Enabling an IAM User to Start Other User's Notebook Instance

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.

2.3.5 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.

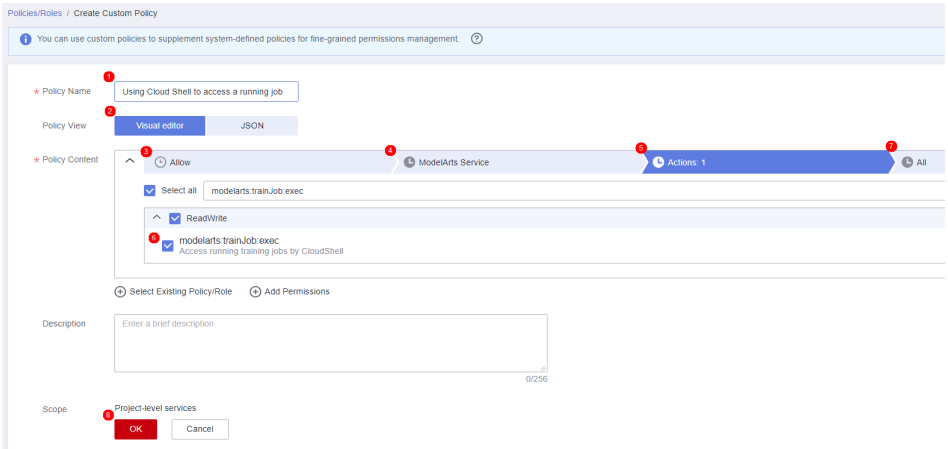
Constraints

You can use Cloud Shell to log in to a running training container using a dedicated resource pool.

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 2-44 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

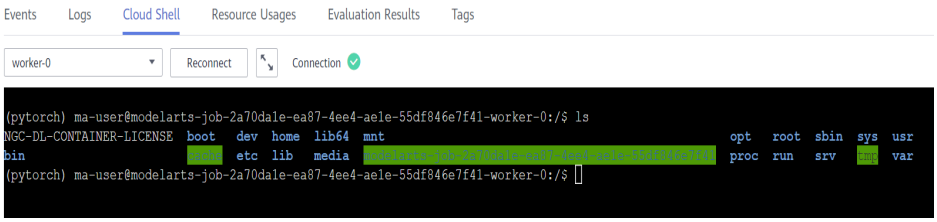
not in a user group, you can add the user to a user group through the user group management function.

Using Cloud Shell

1. Configure parameters based on **Preparation: Assigning the Cloud Shell Permission to an IAM User**.
2. On the ModelArts console, choose **Training Management** > **Training Jobs**. Go to the details page of the target training job and log in to the training container on the Cloud Shell tab.

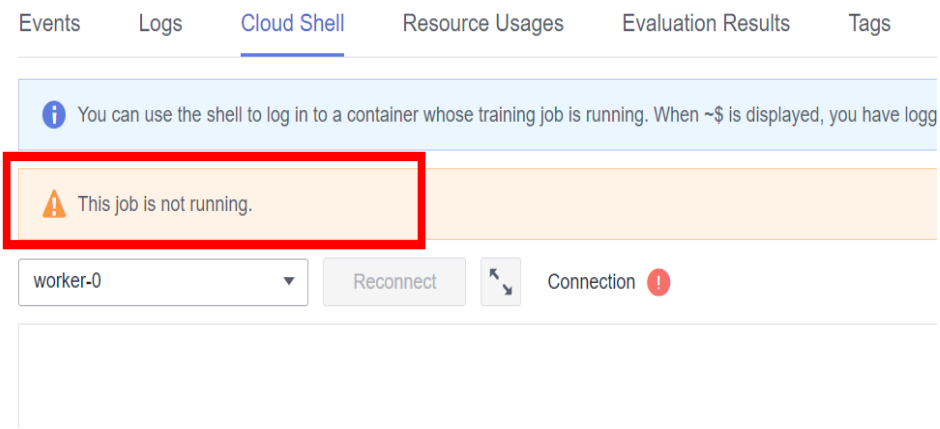
   Verify that the login is successful, as shown in the following figure.

   Figure 2-45 Cloud Shell

   If the job is not running or the permission is insufficient, CloudShell cannot be used. Locate the fault as prompted.

   Figure 2-46 Error message

2.3.6 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. On the **Permissions** tab page, 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 **Training Management** > **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 **DevEnviron** > **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 **Service Deployment** > **Real-Time Services**, and click **Deploy**. On the page for service deployment, only a dedicated resource pool can be selected for **Resource Pool**.

2.3.7 Granting SFS Turbo Folder Access Permissions to IAM Users

Scenarios

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

Constraints

- Ensure that you have enabled strict authorization. Log in to the ModelArts console and choose **Settings** from the navigation pane on the left. On the **Global Configuration** page, click **Enable strict authorization**.
- If ModelArts permissions have not been granted to IAM users, the IAM users may fail to use ModelArts after the strict authorization is enabled. Grant the permission to IAM users by referring to **Assigning Permissions to Individual Users for Using ModelArts**.

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": {
            "StringEquals": {
                "modelarts:sfsId": [
                    "<your_ssf_id>"
                ],
                "modelarts:sfsPath": [
                    "<sfs_path>"
                ],
                "modelarts:sfsOption": [
                    "<sfs_option>"
                ]
            }
        }
    }
  ]
}
```

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

Table 2-22 Parameter description

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

Parameter	Description

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:sfsOption	Type of the access permission. The following parameters are supported:
	• **readonly**: Read-only permission
	• **readwrite**: Read and write permission
	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:notebook:create"
	],
	"Condition": {
	"StringEquals": {
	"modelarts:sfsId": [
	"0e51c7d5-d90e-475a-b5d0-ecf896da3b0d"
	],
	"modelarts:sfsPath": [
	"/path1"
	],
	"modelarts:sfsOption": [
	"readonly"
	]
	}
	}
	},
	{
	"Effect": "Allow",
	"Action": [
	"modelarts:notebook:create"
	],
	"Condition": {
	"StringEquals": {
	"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 Create a User Group and Add Users to the User Group**.

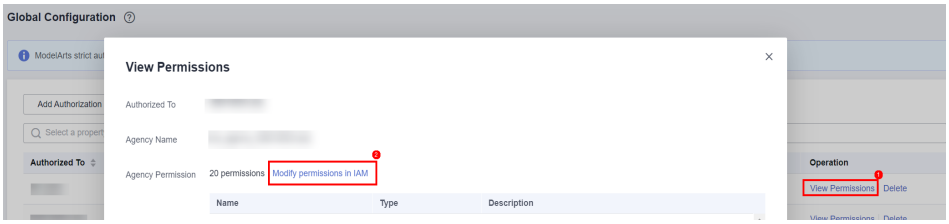
Step 4 Grant a policy to the user group. On the user group list page of IAM, click **Authorize** of the target user group. The **Authorize User Group** page is displayed. Select the **ma_sfs_turbo** policy created in **Step 2**. Click **Next** and then **OK**.

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

1. On the ModelArts management console, choose **Settings** from the navigation pane on the left. On the displayed page, locate the target agency, choose

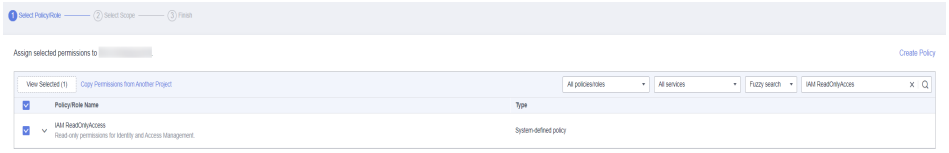
View Permissions in the **Operation** column, and click **Modify permission in IAM**.

Figure 2-47 Modifying permissions 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**.

Figure 2-48 IAM ReadOnlyAccess



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

2.4 FAQ

2.4.1 What Do I Do If a Message Indicating Insufficient Permissions Is Displayed When I Use ModelArts?

If a message indicating insufficient permissions is displayed when you use ModelArts, perform the operations described in this section to grant permissions for related services as needed.

The permissions to use ModelArts depend on OBS authorization. Therefore, ModelArts users require OBS system permissions as well.

- For details about how to grant a user full permissions for OBS and common operations permissions for ModelArts, see **Configuring Common Operations Permissions**.
- For details about how to manage user permissions on OBS and ModelArts in a refined manner and configure custom policies, see **Creating a Custom Policy for ModelArts**.

Configuring Common Operations Permissions

To use the basic functions of ModelArts, assign the **ModelArts CommonOperations** permission on project-level services to users. Since

ModelArts depends on OBS permissions, assign the **OBS Administrator** permission on global services to users.

The procedure is as follows:

Step 1 Create a user group.

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

Step 2 Configure permissions for the user group.

In the user group list, locate the user group created in **step 1**, click **Authorize** , and perform the following operations.

1. Assign the **ModelArts CommonOperations** permission on project-level services to the user group and click **OK**.

   Figure 2-49 Assigning the ModelArts CommonOperations permission

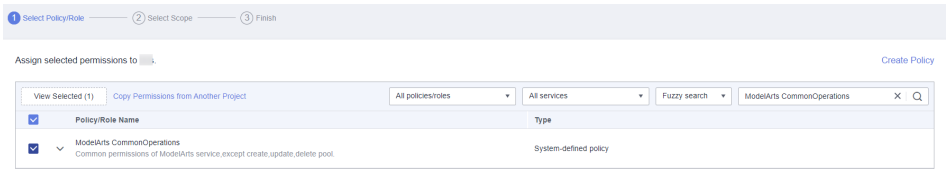
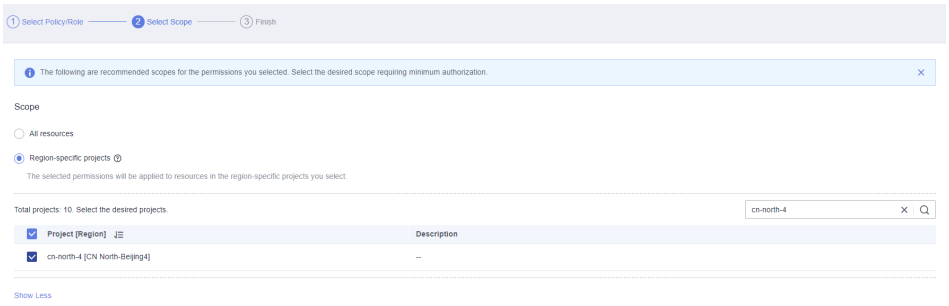
   

   Figure 2-50 Setting Scope to Region-specific projects

   NOTE

   The permission takes effect only in assigned regions. Assign permissions in all regions if the permission is required in all regions.

2. Assign the **OBS Administrator** permission on global services to the user group and click **OK**.

   Figure 2-51 Assigning the OBS Administrator permission

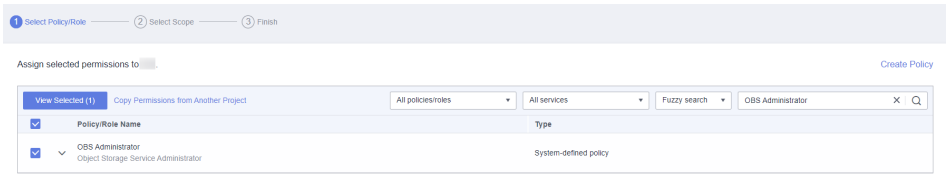
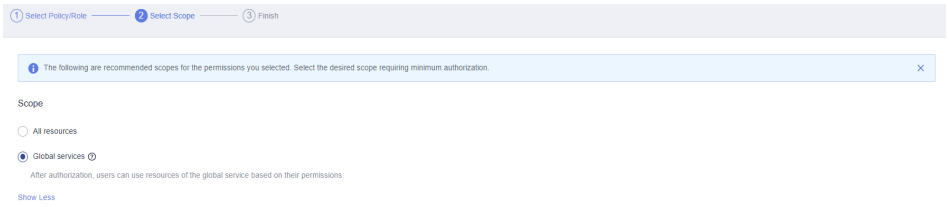
   

Figure 2-52 Setting Scope to Global services



Step 3 **Create a user and add it to the user group**.

Create a user on the IAM console and add the user to the user group created in step 1.

Step 4 **Log in** and verify permissions.

Log in to the ModelArts console as the created user, switch to the authorized region, and verify the **ModelArts CommonOperations** and **Tenant Administrator** policies are in effect.

- Choose **Service List** > **ModelArts**. Choose **Dedicated Resource Pools**. On the page that is displayed, select a resource pool type and click **Create**. You should not be able to create a new resource pool.
- Choose any other service in **Service List**. (Assume that the current policy contains only **ModelArts CommonOperations**.) If a message appears indicating that you have insufficient permissions to access the service, the **ModelArts CommonOperations** policy has already taken effect.
- Choose **Service List** > **ModelArts**. On the ModelArts console, choose **Data Management** > **Datasets** > **Create Dataset**. You should be able to access the corresponding OBS path.

----End

Creating a Custom Policy for ModelArts

In addition to the default system policies of ModelArts, you can create custom policies, which can address OBS permissions as well. For more information, see **Creating a Custom Policy**.

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

NOTE

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

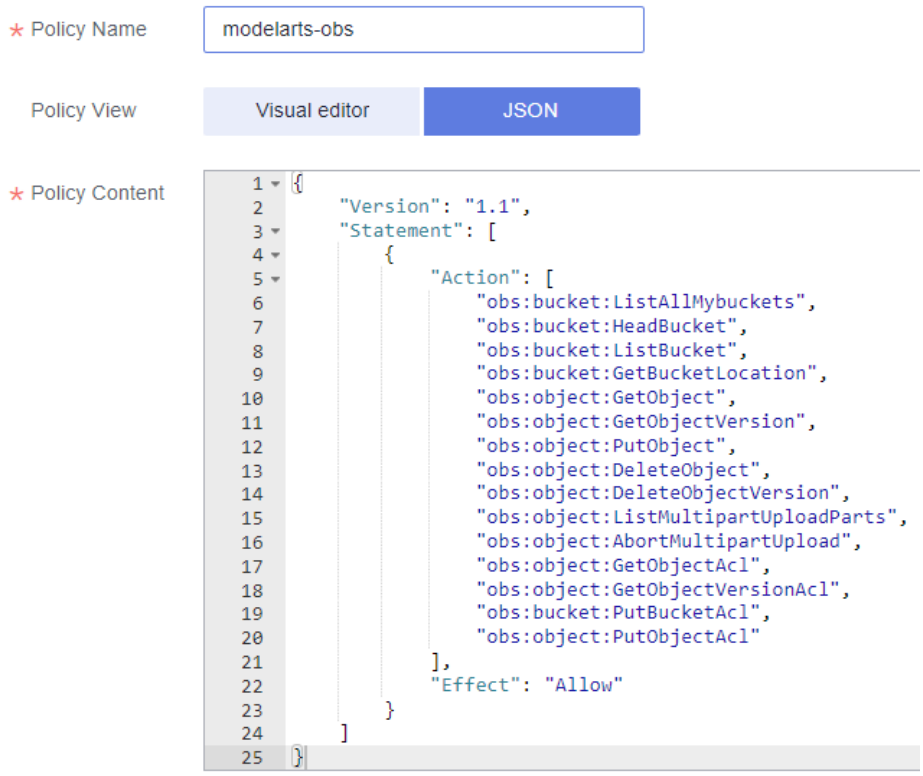
ModelArts is a project-level service, but OBS is a global service, so you need to create separate policies for the two services and then apply these policies to the users.

1. Create a custom policy for minimizing permissions for OBS that ModelArts depends on. See **Figure 2-53**.

   Log in to the IAM console, choose **Permissions** > **Policies/Roles**, and click **Create Custom Policy**. Configure the parameters as follows:

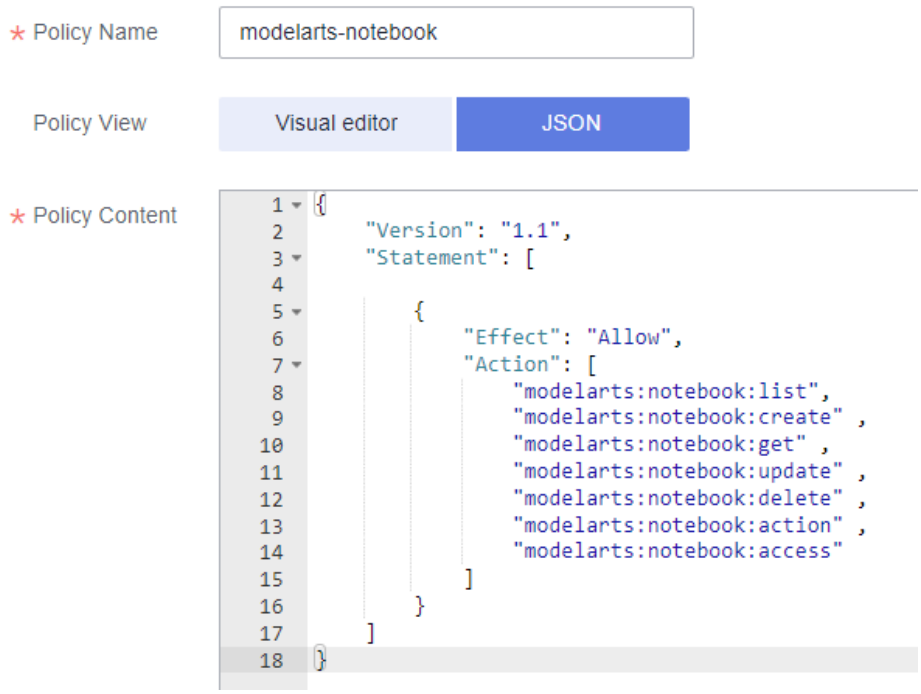
- **Policy Name**: Choose a custom policy name.
- **Policy View**: **JSON**
- **Policy Content**: Follow the instructions in **Example Custom Policies of OBS**. For more information about OBS system permissions, see **OBS Permissions Management**.

Figure 2-53 Minimum permissions for OBS



2. Create a custom policy for the permission to use the ModelArts development environment. See **Figure 2-54**. Configure the parameters as follows:
   - **Policy Name**: Choose a custom policy name.
   - **Policy View**: **JSON**
   - **Policy Content**: Follow the instructions in **Example Custom Policies for Using the ModelArts Development Environment**. For the actions that can be added for custom policies, see **ModelArts API Reference > Permissions Policies and Supported Actions**.

Figure 2-54 Permission to use the development environment



- For the system policies of other services, see **System Permissions**.

3. On the IAM console, **create a user group and grant required permissions.**

   After creating a user group on the IAM console, grant the custom policy created in **1** to the user group.

4. **Create a user and add it to the user group**.

   Create a user on the IAM console and add the user to the group created in **3**.

5. **Log in** and verify permissions.

   Log in to the ModelArts console as the created user, switch to the authorized region, and verify the **ModelArts CommonOperations** and **Tenant Administrator** policies are in effect.

   - Choose **Service List** > **ModelArts**. On the ModelArts console, choose **Data Management** > **Datasets**. If you cannot create a dataset, the permissions (for using the development environment) granted only to ModelArts users have taken effect.
   - Choose **Service List** > **ModelArts**. On the ModelArts console, choose **DevEnviron** > **Notebooks** > **Create**. You should be able to access the OBS path specified in **Storage Path**.

Example Custom Policies of OBS

The permissions to use ModelArts require OBS authorization. The following example shows the minimum OBS required, including the permissions for OBS buckets and objects. After being granted the minimum permissions for OBS, users can access OBS from ModelArts without restrictions.

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

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

Example Custom Policies for Using the ModelArts Development Environment

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

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


3 Notebook

3.1 Creating, Migrating, and Managing Conda Virtual Environments Based on SFS

This topic 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/
```

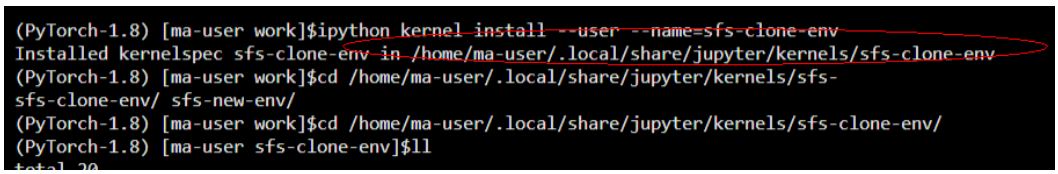
View the existing conda virtual environments. The new virtual environment is properly displayed, and you can switch to it by name.

```
# 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.



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
```


4 Model Training

4.1 Using a Custom Algorithm to Build a Handwritten Digit Recognition Model

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.

Step 1: Making Preparations

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

  Certain ModelArts functions require access to OBS, SWR, and IEF. Before using ModelArts, ensure your account has been authorized to access 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. On the **Add Authorization** page that is displayed, set required parameters as follows:

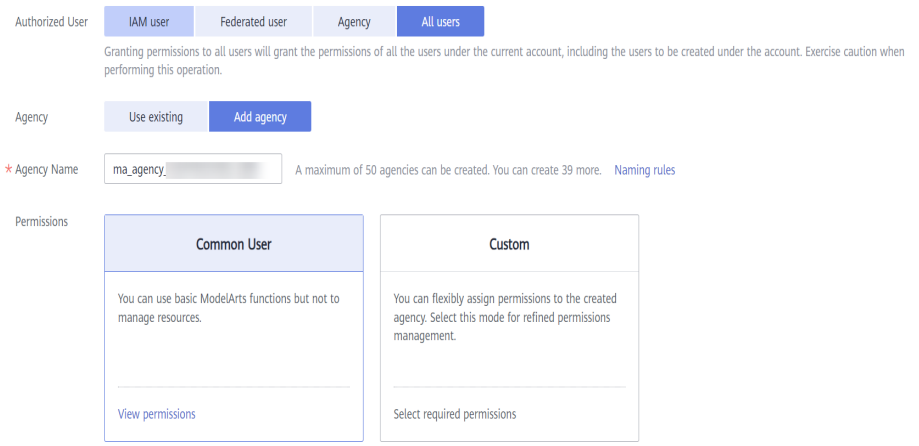
  Authorized User: Select **All users**.

  Agency: Select **Add agency**.

  Permissions: Select **Common User**.

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

Figure 4-1 Configuring the agency-based authorization



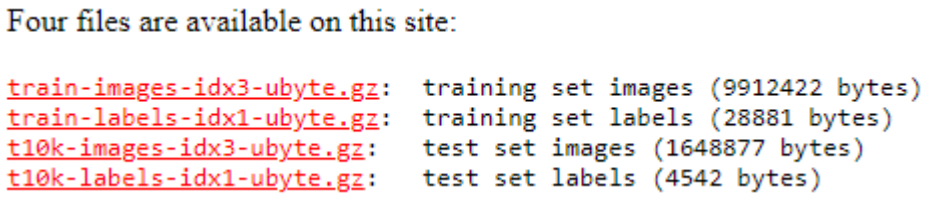
c. After the configuration, view the agency configurations of your account on the **Global Configuration** page.

Figure 4-2 Viewing agency configurations

Step 2: Preparing Training Data

An MNIST dataset downloaded from the **MNIST official website** is used in this case. Ensure that the four files are all downloaded.

Figure 4-3 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

NOTE

If you are asked to enter the login information after you click the MNIST official website link, copy and paste this link in the address box of your browser: http://yann.lecun.com/exdb/mnist/

The login information is required when you open the link in HTTPS mode, which is not required if you open the link in HTTP mode.

Step 3: 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 a 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 4: 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 the OBS console and create an OBS bucket and folder. **Figure 4-4** shows an example of the created objects. For details, see **Creating a Bucket** and **Creating a 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
   ```

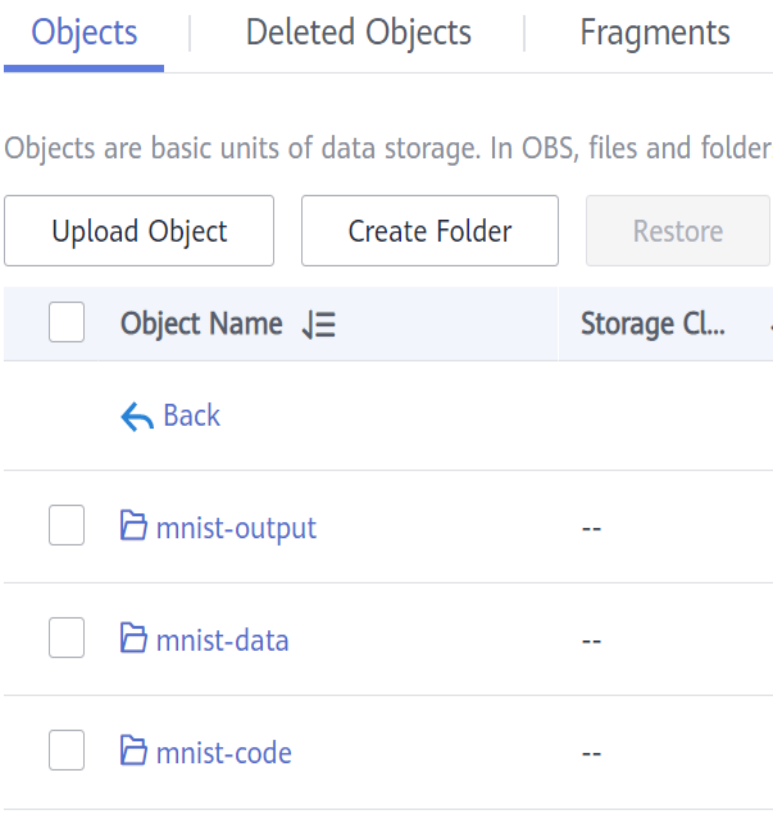


```
        - mnist-data      # OBS folder, which is used to store the training dataset. The folder name is customizable, for example, mnist-data.
        - mnist-code      # OBS folder, which is used to store training script train.py. The folder name is customizable, for example, mnist-code.
            - 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.

Figure 4-4 OBS file directory

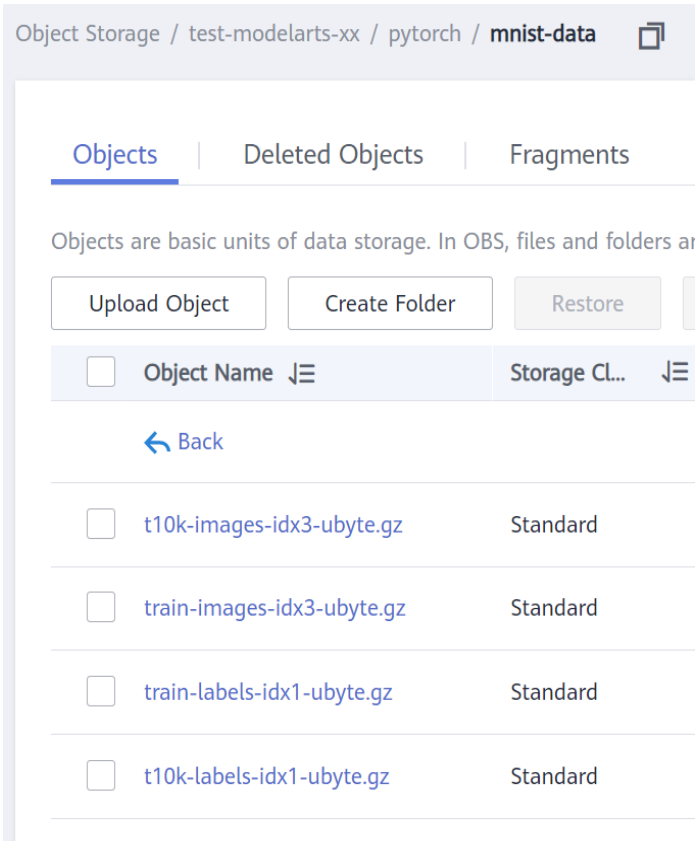


2. Upload the MNIST dataset package obtained in **Step 2: Preparing Training Data** to OBS. For details, see **Uploading an Object**.

CAUTION

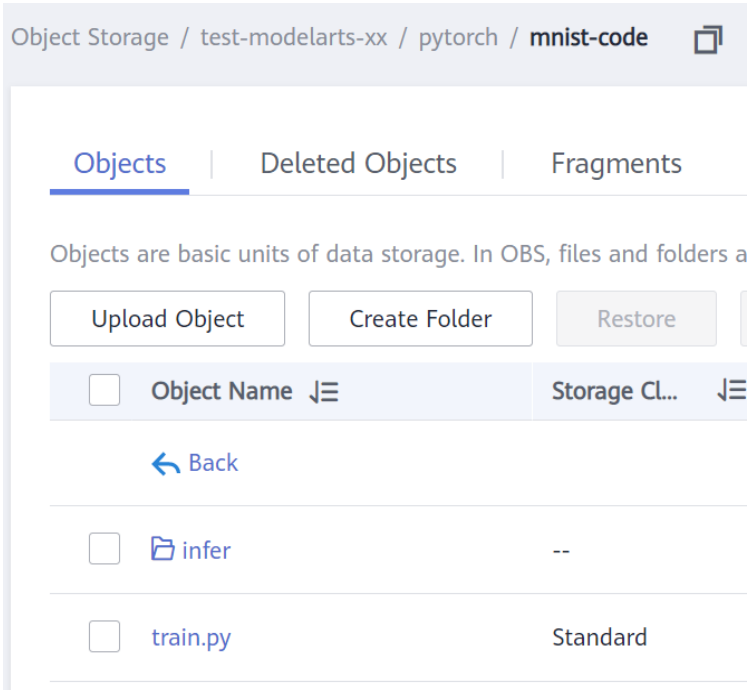
- When uploading data to OBS, do not encrypt the data. Otherwise, the training will fail.
- Files do not need to be decompressed. Directly upload compressed packages to OBS.

Figure 4-5 Uploading a dataset to the mnist-data folder



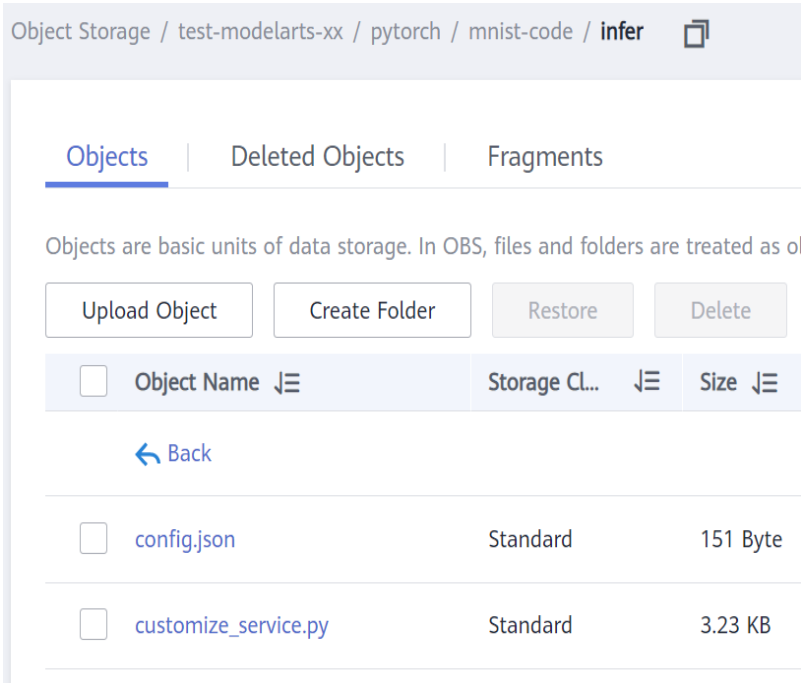
3. Upload the training script **train.py** to the **mnist-code** folder.

Figure 4-6 Uploading 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.

Figure 4-7 Uploading **customize_service.py** and **config.json** to the **infer** folder



Step 5: Creating 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 **Settings** and check whether access authorization has been configured for the current account. For details, see **Configuring Access Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
3. In the navigation pane, choose **Training Management** > **Training Jobs**. On the **Training Jobs** page that appears, 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 **GPU** and then **GPU: 1*NVIDIA-V100(16GB) | CPU: 8 vCPUs 64GB** (example). 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.

Figure 4-8 Training job settings

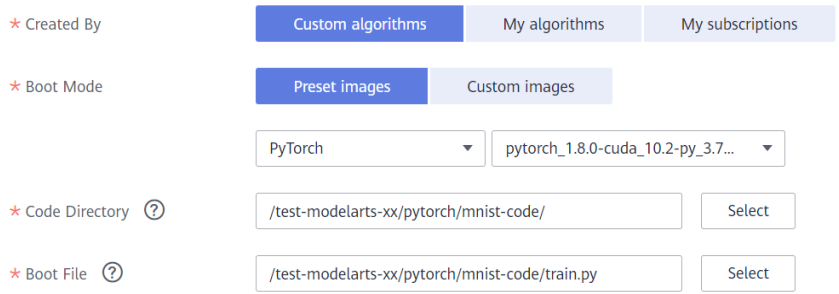


Figure 4-9 Setting training input and output

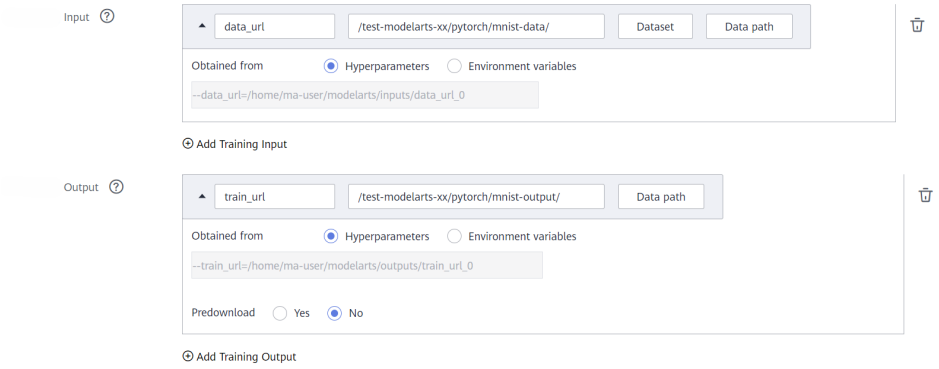
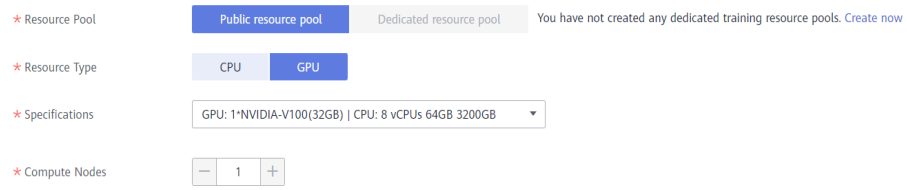


Figure 4-10 Configuring the resource type



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 4-11**). Then, check whether the **model** folder is available and whether there are any trained models in the folder (as shown in **Figure 4-12**). 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 4-11 Output path

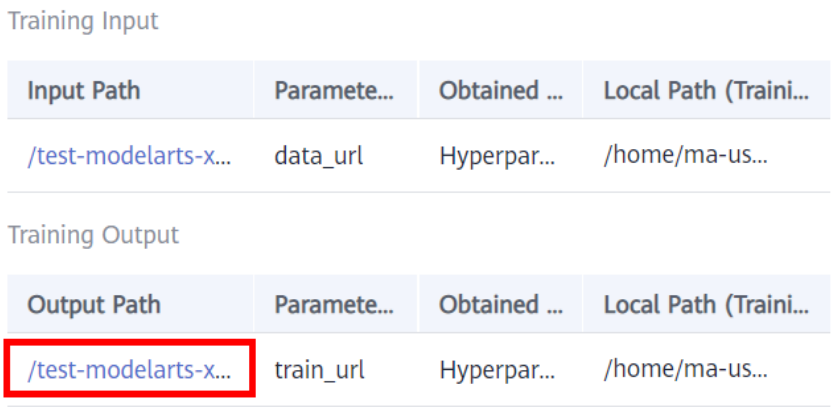
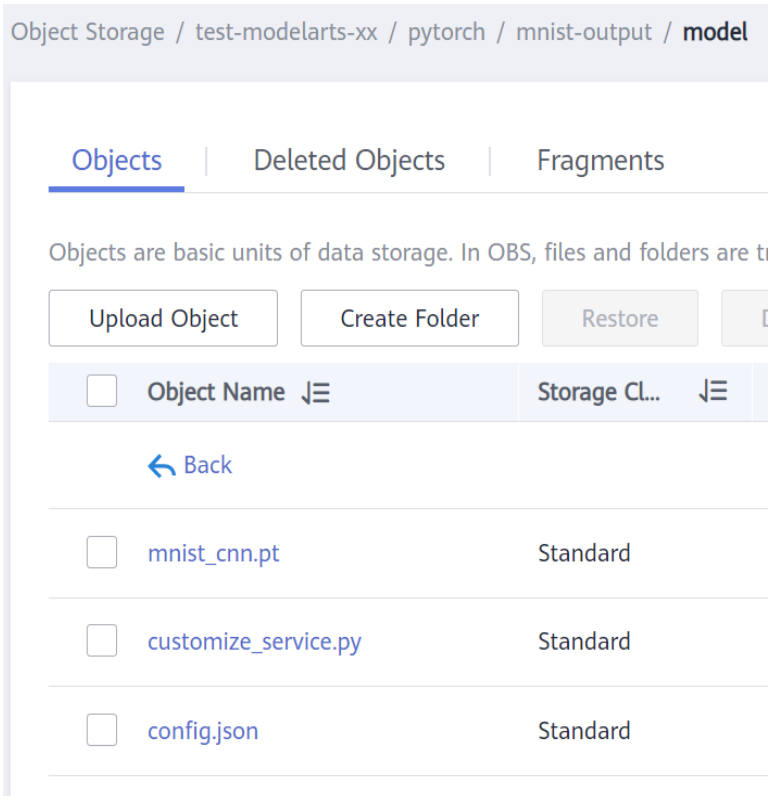


Figure 4-12 Trained model



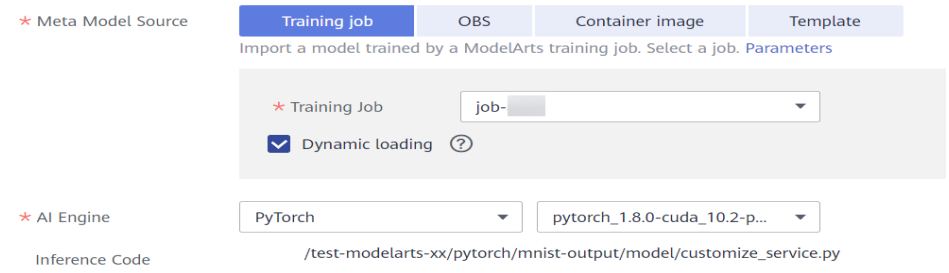
Step 6: Deploying an Inference Service

After the model training is complete, create an AI application and deploy it as a real-time service.

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

   Choose **Training Job** for **Meta Model Source**. Select the training job completed in **Step 5: Creating a Training Job** from the drop-down list and select **Dynamic loading**. The values of **AI Engine** will be automatically configured.

Figure 4-13 Meta Model Source

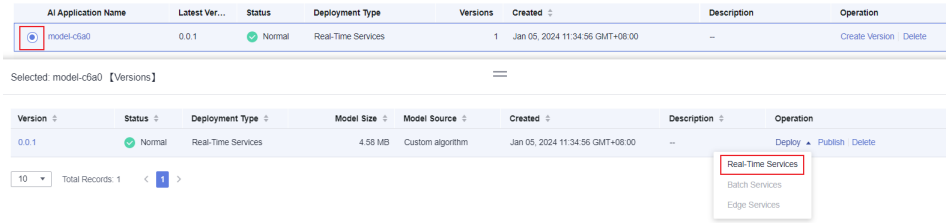


NOTE

If you have used **Training Jobs** of an old version, you can see both **Training Jobs** and **Training Jobs New** below **Training job**. In this case, select **Training Jobs New**.

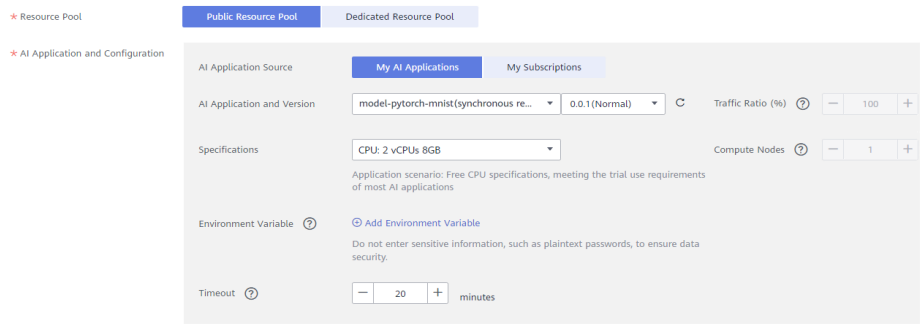
3. On the **AI Applications** page, if the application status changes to **Normal**, it has been created. Click the option button on the left of the AI application name to display the version list at the bottom of the list page, and choose **Deploy** > **Real-Time Services** in the **Operation** column to deploy the AI application as a real-time service.

Figure 4-14 Deploying a real-time service



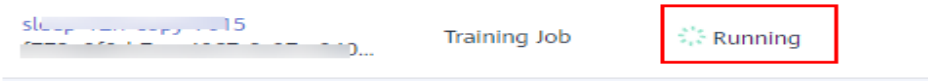
4. On the **Deploy** page, configure parameters and create a real-time service as prompted. In this example, use CPU specifications. If there are free CPU specifications, you can select them for deployment. (Each user can deploy only one real-time service for free. If you have deployed one, delete it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

Figure 4-15 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.

Figure 4-16 Deployed service



Step 7: 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**.

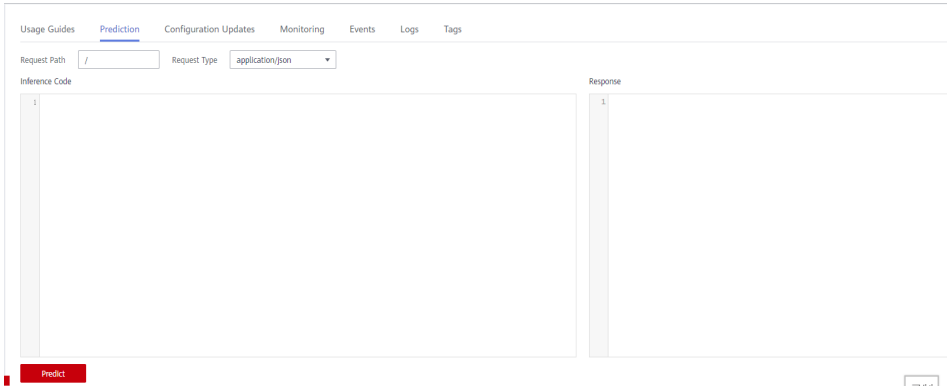
   After the prediction is complete, the prediction result is displayed in the **Test Result** pane. According to the prediction result, the digit on the image is **2**.

   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 4-17 Example

Figure 4-18 Prediction results



Step 8: 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 row containing the target service and click **Stop** or **Delete** in the **Operation** column.
- On the **AI Applications** page in **AI Application Management**, locate the row containing the target service and click **Delete** in the **Operation** column.
- On the **Training Jobs** page, click **Delete** in the **Operation** column to delete the finished 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**.

4.2 Example: Creating a Custom Image for Training (PyTorch + CPU/GPU)

This section describes how to create an image and use the image for training on the ModelArts platform. The AI engine used for training is PyTorch, and the resources are CPUs or GPUs.

NOTE

This section applies only to training jobs of the new version.

Scenarios

In this example, create a custom image by writing a Dockerfile on a Linux x86_64 host running the Ubuntu 18.04 operating system.

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training on ModelArts.

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience. The following is the detailed procedure:

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing the Training Script and Uploading It to OBS**
4. **Step 3 Preparing a Host**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading an Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

Prerequisites

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

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-1** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

Table 4-1 Folder to create

Name	Description
obs://test-modelarts/pytorch/demo-code/	Stores the training script.
obs://test-modelarts/pytorch/log/	Stores training log files.

Step 2 Preparing the Training Script and Uploading It to OBS

Prepare the training script **pytorch-verification.py** and upload it to the **obs://test-modelarts/pytorch/demo-code/** folder of the OBS bucket.

The **pytorch-verification.py** file contains the following information:

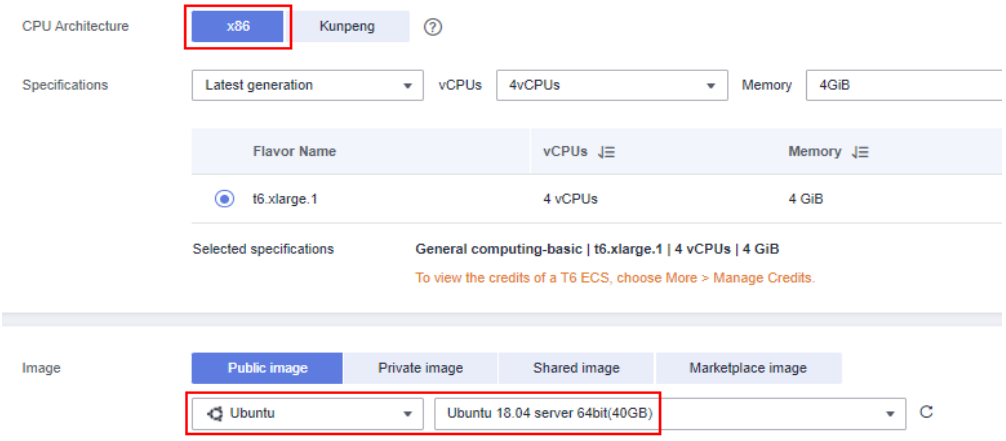
```
import torch
import torch.nn as nn

x = torch.randn(5, 3)
print(x)

available_dev = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
y = torch.randn(5, 3).to(available_dev)
print(y)
```

Step 3 Preparing a Host

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

Figure 4-19 Creating an ECS using a public image (x86)



Step 4 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- pytorch-1.8.1

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses the Linux x86_64 OS as an example to describe how to obtain the Docker installation package. For more details about how to install Docker, see **official Docker documents**.

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

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

2. Run the following command to check the Docker Engine version:

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

   The following information is displayed:

   ```
   ...
   Engine:
    Version:        18.09.0
   ```

   NOTE

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

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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


> **NOTE**
>
> In Huawei Mirrors **https://mirrors.huaweicloud.com/home**, search for **pypi** to obtain the pip.conf file.

5. Download the following .whl files from **https://download.pytorch.org/whl/torch_stable.html**:
   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   > **NOTE**
   >
   > The URL code of the + symbol is %2B. When searching for a file in the above website, replace the + symbol in the file name with %2B.
   >
   > For example, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

6. Download the Miniconda3-py37_4.12.0-Linux-x86_64.sh installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.
7. Store the pip source file, torch*.whl file, and Miniconda3 installation file in the **context** folder, which is as follows:

```
context
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

8. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The host must be connected to the public network for creating a container image.

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

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

# Use the PyPI configuration provided by Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# Install Miniconda3 to the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install torch*.whl using the default Miniconda3 Python environment in /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
```



```
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# Create the final container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim and cURL in Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    apt-get update && \
    apt-get install -y vim curl && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the base container image exists. User ma-user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

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

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

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

```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

10. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **pytorch:1.8.1-cuda11.1**:

```
docker build . -t pytorch:1.8.1-cuda11.1
```

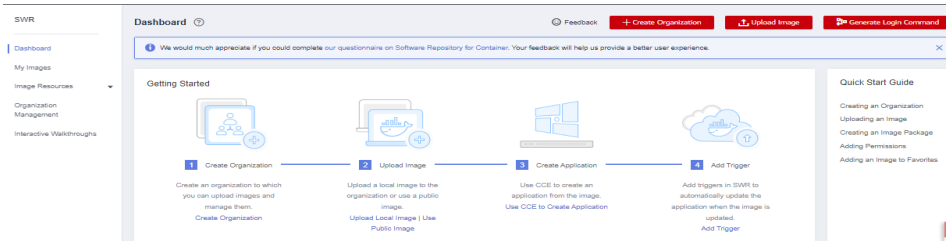
The following log information displayed during image creation indicates that the image has been created.

```
Successfully tagged pytorch:1.8.1-cuda11.1
```

Step 5 Uploading an Image to SWR

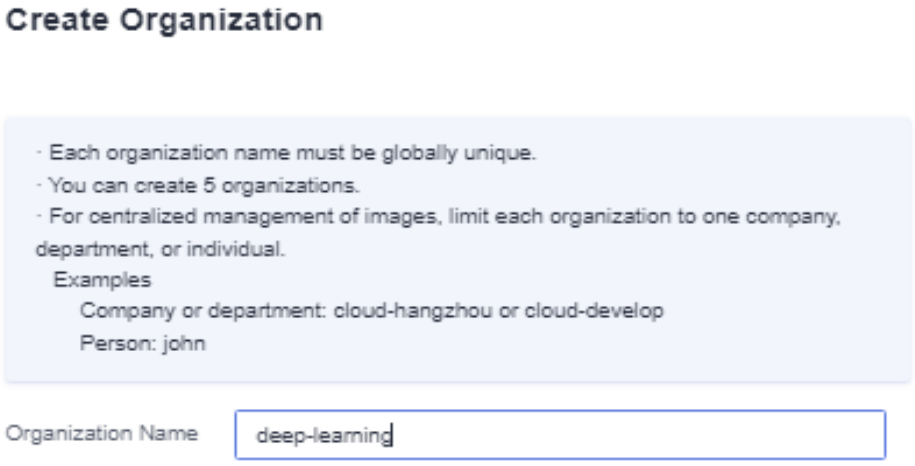
1. Log in to the SWR console and select the target region.

Figure 4-20 SWR console



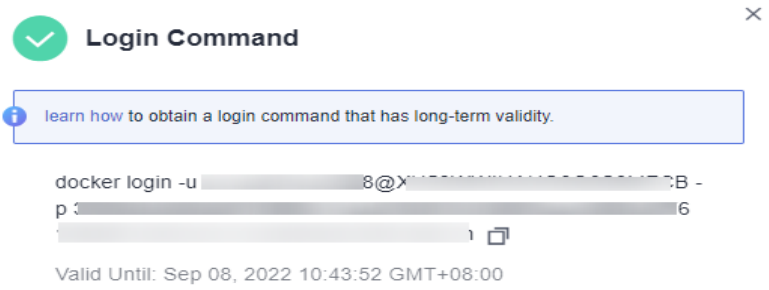
2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

   Figure 4-21 Creating an organization

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

   Figure 4-22 Login Command

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Run the following command to tag the uploaded image:
      ```
      #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
      sudo docker tag pytorch:1.8.1-cuda11.1 swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
      ```
   b. Run the following command to upload the image:
      ```
      #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
      sudo docker push swr.{region-id}.{domain}/deep-learning/pytorch:1.8.1-cuda11.1
      ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

Step 6 Creating a Training Job on ModelArts

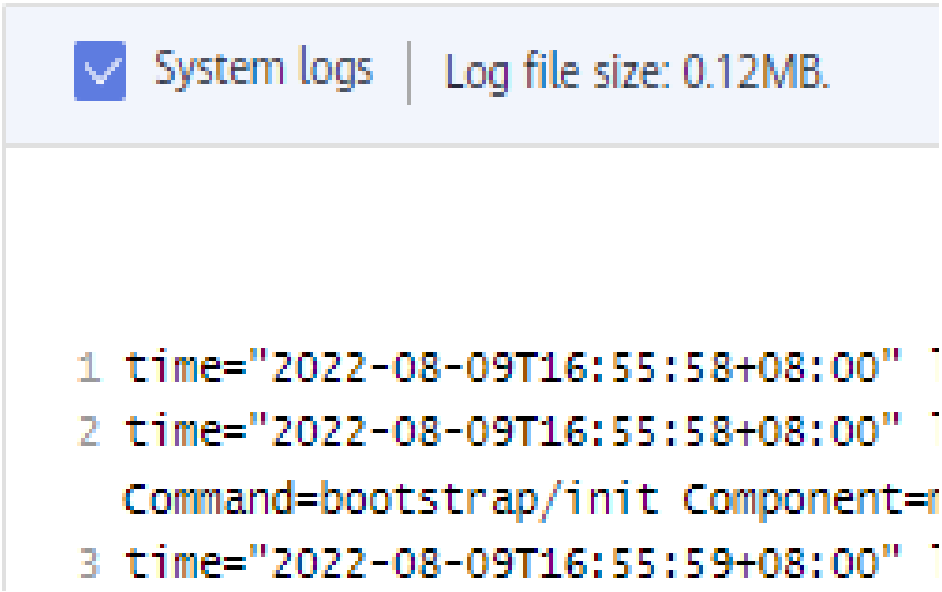
1. Log in to the ModelArts management console and check whether access authorization has been configured for your account. For details, see

Configuring Agency Authorization. If you have been authorized using access keys, clear the authorization and configure agency authorization.

2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. On the **Create Training Job** page, set required parameters and click **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Uploading an Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/pytorch/demo-code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **/home/ma-user/miniconda3/bin/python ${MA_JOB_DIR}/demo-code/pytorch-verification.py**. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Resource Pool**: **Public resource pools**
   - **Resource Type**: Select **CPU** or **GPU**.
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: Set this parameter to the OBS path for storing training logs, for example, **obs://test-modelarts/pytorch/log/**.
4. Check the parameters of the training job and click **Submit**.
5. Wait until the training job is completed.

   After a training job is created, the operations such as container image downloading, code directory downloading, and boot command execution are automatically performed in the backend. Generally, the training duration ranges from dozens of minutes to several hours, depending on the training procedure and selected resources. After the training job is executed, the log similar to the following is output.

Figure 4-23 Run logs of training jobs with GPU specifications



4.3 Example: Creating a Custom Image for Training (MPI + CPU/GPU)

This section describes how to create an image and use the image for training on the ModelArts platform. The AI engine used for training is MPI, and the resources are CPUs or GPUs.

NOTE

This section applies only to training jobs of the new version.

Scenarios

In this example, create a custom image by writing a Dockerfile on a Linux x86_64 host running the Ubuntu 18.04 operating system.

Objective: Build and install container images of the following software and use the images and CPUs/GPUs for training on ModelArts.

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- openmpi-3.0.0

Procedure

Before using a custom image to create a training job, get familiar with Docker and have development experience. The following is the detailed procedure:

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing Script Files and Uploading Them to OBS**
4. **Step 3 Preparing an Image Server**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading an Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

Prerequisites

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

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-2** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

Table 4-2 Folder to create

Name	Description
obs://test-modelarts/mpi/demo-code/	Stores the MPI boot script and training script file.
obs://test-modelarts/mpi/log/	Stores training log files.

Step 2 Preparing Script Files and Uploading Them to OBS

Prepare the MPI boot script **run_mpi.sh** and training script **mpi-verification.py** and upload them to the **obs://test-modelarts/mpi/demo-code/** folder of the OBS bucket.

- The content of the MPI boot script **run_mpi.sh** is as follows:

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"38888"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"
```



```
        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-
p ${MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG -x NCCL_SOCKET_IFNAME -x NCCL_IB_HCA -x NCCL_IB_TIMEOUT -x
NCCL_IB_GID_INDEX -x NCCL_IB_TC \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x PATH -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
```



```
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

NOTE

The script **run_mpi.sh** uses LF line endings. If CRLF line endings are used, executing the training job will fail, and the error "$'\r': command not found" will be displayed in logs.

- The content of the training script **mpi-verification.py** is as follows:

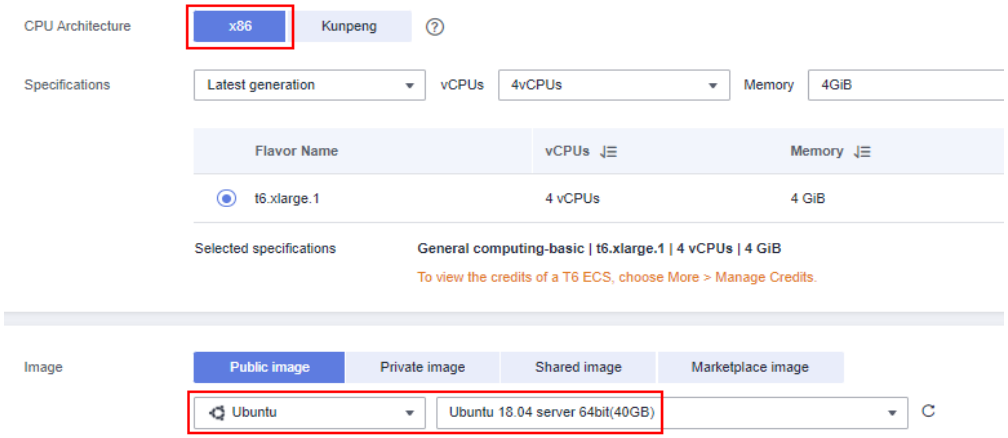
```
import os
import socket

if __name__ == '__main__':
    print(socket.gethostname())

    # https://www.open-mpi.org/faq/?category=running#mpi-environmental-variables
    print('OMPI_COMM_WORLD_SIZE: ' + os.environ['OMPI_COMM_WORLD_SIZE'])
    print('OMPI_COMM_WORLD_RANK: ' + os.environ['OMPI_COMM_WORLD_RANK'])
    print('OMPI_COMM_WORLD_LOCAL_RANK: ' + os.environ['OMPI_COMM_WORLD_LOCAL_RANK'])
```

Step 3 Preparing an Image Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

Figure 4-24 Creating an ECS using a public image (x86)



Step 4 Creating a Custom Image

Objective: Build and install container images of the following software and use the ModelArts training service to run the images.

- ubuntu-18.04
- cuda-11.1
- python-3.7.13

- openmpi-3.0.0

The following describes how to create a custom image by writing a Dockerfile.

1. Install Docker.

   The following uses the Linux x86_64 OS as an example to describe how to obtain a Docker installation package. For more details, see **Docker official documents**. Run the following commands to install Docker:

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

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

2. Check the Docker engine version. Run the following command:

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

   The following information is displayed:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTE

   You are advised to use Docker Engine of this version or later to create a custom image.

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

5. Download the openmpi 3.0.0 installation file.

   Download the openmpi 3.0.0 file edited using Horovod v0.22.1 from **https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz**.

6. Store the Miniconda3 and openmpi 3.0.0 files in the **context** folder. The following shows the **context** folder:

   ```
   context
   ├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
   └── openmpi-3.0.0-bin.tar.gz
   ```

7. Write the Dockerfile of the container image.

   Create an empty file named **Dockerfile** in the **context** folder and write the following content to the file:

   ```
   # The host must be connected to the public network for creating a container image.

   # Basic container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
   #
   # https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
   # require Docker Engine >= 17.05
   #
   # builder stage
   FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04 AS builder

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

   # Copy the Miniconda3 (Python 3.7.13) installation files to the /tmp directory of the basic container image.
   COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp

   # Install Miniconda3 to the /home/ma-user/miniconda3 directory of the basic container image.
   ```



```
# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Create the final container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

# Install vim, cURL, net-tools, and the SSH tool in Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# Install the Open MPI 3.0.0 file written using Horovod v0.22.1.
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 of the basic container image exists. User ma-user can directly use it.
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to avoid log loss.
ENV PATH=$PATH:/home/ma-user/miniconda3/bin \
    PYTHONUNBUFFERED=1

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

# Configure sshd to support SSH password-free login.
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
 StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config
```

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

8. Verify that the Dockerfile has been created. The following shows the **context** folder:


```
context
├── Dockerfile
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
└── openmpi-3.0.0-bin.tar.gz
```

9. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **mpi:3.0.0-cuda11.1**:

```
docker build . -t mpi:3.0.0-cuda11.1
```

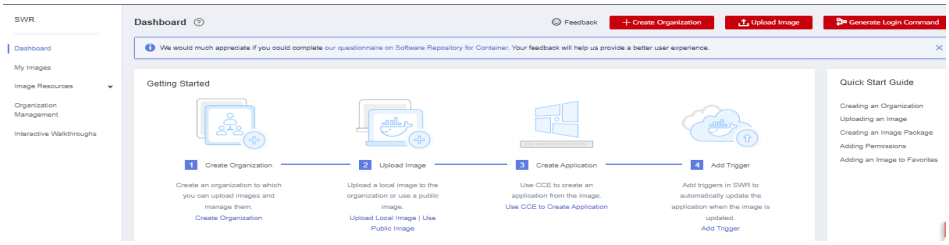
The following log information displayed during image creation indicates that the image has been created.

```
naming to docker.io/library/mpi:3.0.0-cuda11.1
```

Step 5 Uploading an Image to SWR

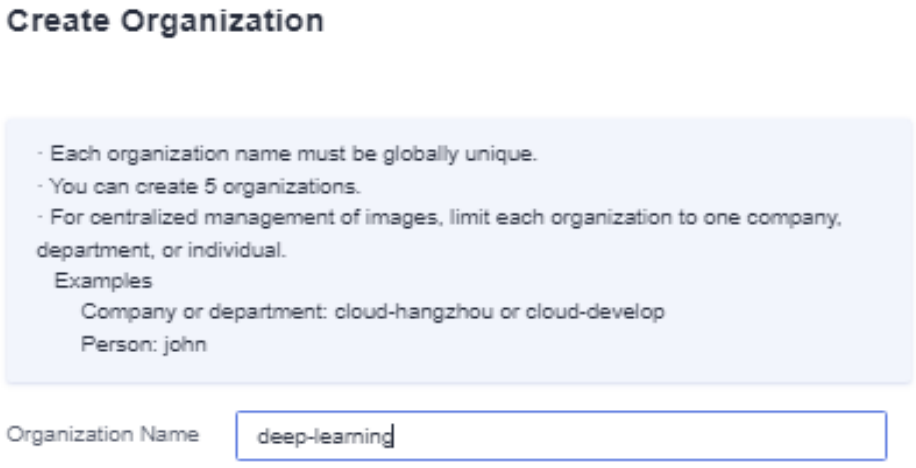
1. Log in to the SWR console and select the target region.

Figure 4-25 SWR console



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

Figure 4-26 Creating an organization



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

Figure 4-27 Login Command

4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Run the following command to tag the uploaded image:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker tag mpi:3.0.0-cuda11.1 swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
   b. Run the following command to upload the image:
   ```
   #Replace the region and domain information with the actual values, and replace the organization name deep-learning with your custom value.
   sudo docker push swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1
   ```
6. After the image is uploaded, choose **My Images** on the left navigation pane of the SWR console to view the uploaded custom images.

   swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1 is the SWR URL of the custom image.

Step 6 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. Log in to the ModelArts management console. In the left navigation pane, choose **Training Management** > **Training Jobs (New)**.
3. On the **Create Training Job** page, configure parameters and click **Submit**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: **swr.cn-north-4.myhuaweicloud.com/deep-learning/mpi:3.0.0-cuda11.1**
   - **Code Directory**: OBS path to the boot script, for example, **obs://test-modelarts/mpi/demo-code/**.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/mpi-verification.py**
   - **Environment Variable**: Add **MY_SSHD_PORT = 38888**.

Figure 4-28 Adding an environment variable

- **Resource Pool**: **Public resource pools**
- **Resource Type**: Select **GPU**.
- **Compute Nodes**: Enter **1** or **2**.
- **Persistent Log Saving**: enabled
- **Job Log Path**: Set this parameter to the OBS path for storing training logs, for example, **obs://test-modelarts/mpi/log/**.

4. Check the parameters of the training job and click **Submit**.
5. Wait until the training job is completed.

   After a training job is created, the operations such as container image downloading, code directory downloading, and boot command execution are automatically performed in the backend. Generally, the training duration ranges from dozens of minutes to several hours, depending on the training procedure and selected resources. After the training job is executed, the log similar to the following is output.

   Figure 4-29 Run logs of worker-0 with one compute node and GPU specifications

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      60/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      60/sshd
172.16.0.122 slots=1
modelarts-job-8cf8a682-21cb-4d73-9bb3-789cecdc458b-worker-0
OMPI_COMM_WORLD_SIZE: 1
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
```

   Set **Compute Nodes** to **2** and run the training job. **Figure 4-30** and **Figure 4-31** show the log information.

Figure 4-30 Run logs of worker-0 with two compute nodes and GPU specifications

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 0
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      61/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      61/sshd
172.16.0.39 slots=1
172.16.0.123 slots=1
Warning: Permanently added '[172.16.0.123]:38888' (RSA) to the list of known hosts.
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-0
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 0
OMPI_COMM_WORLD_LOCAL_RANK: 0
modelarts-job-31732752-6857-4e33-96ff-7a28afae26fb-worker-1
OMPI_COMM_WORLD_SIZE: 2
OMPI_COMM_WORLD_RANK: 1
OMPI_COMM_WORLD_LOCAL_RANK: 0
```

Figure 4-31 Run logs of worker-1 with two compute nodes and GPU specifications

```
MY_HOME: /home/ma-user
MY_SSHD_PORT: 38888
MY_MPI_BTL_TCP_IF: eth0,bond0
MY_TASK_INDEX: 1
MY_MPI_SLOTS: 1
tcp        0      0 0.0.0.0:38888           0.0.0.0:*               LISTEN      62/sshd
tcp6       0      0 :::38888                :::*                    LISTEN      62/sshd
/home/ma-user/modelarts/user-job-dir/[illegible]e/run_mpi.sh: line 109:    66 Terminated              sleep 365d
```

4.4 Example: Creating a Custom Image for Training (Horovod-PyTorch and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is horovod_0.22.1-pytorch_1.8.1, and the resources used for training are GPUs.

NOTE

This section applies only to training jobs of the new version.

Scenario

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

Create a container image with the following configurations and use the image to create a CPU- or GPU-powered training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1

- horovod-0.22.1

Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Preparing the Training Script and Uploading It to OBS**
4. **Step 3 Preparing a Server**
5. **Step 4 Creating a Custom Image**
6. **Step 5 Uploading the Image to SWR**
7. **Step 6 Creating a Training Job on ModelArts**

Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-3** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details about how to create an OBS bucket and folder, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS directory you use and ModelArts are in the same region.

Table 4-3 Folder to create

Name	Description
obs://test-modelarts/pytorch/demo-code/	Stores the training script.
obs://test-modelarts/pytorch/log/	Stores training log files.

Step 2 Preparing the Training Script and Uploading It to OBS

Obtain training scripts **pytorch_synthetic_benchmark.py** and **run_mpi.sh** and upload them to **obs://test-modelarts/horovod/demo-code/** in the OBS bucket.

pytorch_synthetic_benchmark.py is as follows:

```
import argparse
import torch.backends.cudnn as cudnn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data.distributed
from torchvision import models
```



```
import horovod.torch as hvd
import timeit
import numpy as np

# Benchmark settings
parser = argparse.ArgumentParser(description='PyTorch Synthetic Benchmark',
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='resnet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

parser.add_argument('--use-adasum', action='store_true', default=False,
                    help='use adasum algorithm to do reduction')

args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

hvd.init()

if args.cuda:
    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

cudnn.benchmark = True

# Set up standard model.
model = getattr(models, args.model)()

# By default, Adasum doesn't need scaling up learning rate.
lr_scaler = hvd.size() if not args.use_adasum else 1

if args.cuda:
    # Move model to GPU.
    model.cuda()
    # If using GPU Adasum allreduce, scale learning rate by local_size.
    if args.use_adasum and hvd.nccl_built():
        lr_scaler = hvd.local_size()

optimizer = optim.SGD(model.parameters(), lr=0.01 * lr_scaler)

# Horovod: (optional) compression algorithm.
compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(optimizer,
                                     named_parameters=model.named_parameters(),
                                     compression=compression,
                                     op=hvd.Adasum if args.use_adasum else hvd.Average)

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)

# Set up fixed fake data
data = torch.randn(args.batch_size, 3, 224, 224)
```



```
target = torch.LongTensor(args.batch_size).random_() % 1000
if args.cuda:
    data, target = data.cuda(), target.cuda()


def benchmark_step():
    optimizer.zero_grad()
    output = model(data)
    loss = F.cross_entropy(output, target)
    loss.backward()
    optimizer.step()


def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')


log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

# Warm-up
log('Running warmup...')
timeit.timeit(benchmark_step, number=args.num_warmup_batches)

# Benchmark
log('Running benchmark...')
img_secs = []
for x in range(args.num_iters):
    time = timeit.timeit(benchmark_step, number=args.num_batches_per_iter)
    img_sec = args.batch_size * args.num_batches_per_iter / time
    log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
    img_secs.append(img_sec)

# Results
img_sec_mean = np.mean(img_secs)
img_sec_conf = 1.96 * np.std(img_secs)
log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
log('Total img/sec on %d %s(s): %.1f +-%.1f' %
    (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))
```

run_mpi.sh is as follows:

```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
```



```
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi

RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?
```



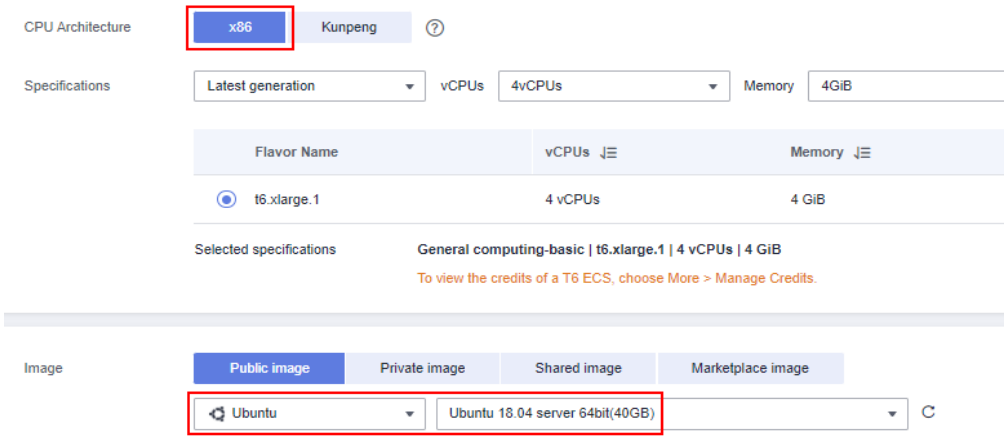
```
    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

Step 3 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

Figure 4-32 Creating an ECS using a public image (x86)



Step 4 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- pytorch-1.8.1
- horovod-0.22.1

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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

   If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:

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

   The following information is displayed:

   ```
   Engine:
    Version:          18.09.0
   ```

   NOTE

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

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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

   NOTE

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

5. Download the source Horovod code file.

   Download **horovod-0.22.1.tar.gz** from https://pypi.org/project/horovod/0.22.1/#files.

6. Download .whl files.

   Download the following .whl files from https://download.pytorch.org/whl/torch_stable.html.

   - torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
   - torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
   - torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl

   NOTE

   The URL code of the plus sign (+) is %2B. When searching for files in the preceding websites, replace the plus sign (+) in the file name with %2B, for example, **torch-1.8.1%2Bcu111-cp37-cp37m-linux_x86_64.whl**.

7. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

8. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The server on which the container image is created must access the Internet.

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder

# Install CMake obtained from Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y build-essential cmake g++-7 && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

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

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl /tmp
COPY torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl /tmp
COPY openmpi-3.0.0-bin.tar.gz /tmp
COPY horovod-0.22.1.tar.gz /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# Environment variables required for building Horovod with PyTorch
ENV HOROVOD_NCCL_INCLUDE=/usr/include \
    HOROVOD_NCCL_LIB=/usr/lib/x86_64-linux-gnu \
    HOROVOD_MPICXX_SHOW="/usr/local/openmpi/bin/mpicxx -show" \
    HOROVOD_GPU_OPERATIONS=NCCL \
    HOROVOD_WITH_PYTORCH=1

# Install the .whl files using default Miniconda3 Python environment /home/ma-user/
miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
```



```
    /tmp/torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl \
    /tmp/torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl

# Build and install horovod-0.22.1.tar.gz using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/horovod-0.22.1.tar.gz

# Create the container image.
FROM nvidia/cuda:11.1.1-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1 libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

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

# Configure sshd to support SSH password-free login.
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
```



```
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config

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

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

9. Download the MLNX_OFED installation package.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, in the **Download** tab, select a proper installation package from **Current Versions** or **Archive Versions**. In this example, choose **Archive Versions**, set **Version** to **5.4-3.5.8.0-LTS**, **OS Distribution** to **Ubuntu**, **OS Distribution Version** to **Ubuntu 18.04**, **Architecture** to **x86_64**, and download the **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz** installation package.

10. Download **openmpi-3.0.0-bin.tar.gz**.

    Download **openmpi-3.0.0-bin.tar.gz** from https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

11. Store the pip source file, .whl files, and Miniconda3 installation file in the **context** folder, which is as follows:

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── horovod-0.22.1.tar.gz
├── openmpi-3.0.0-bin.tar.gz
├── pip.conf
├── torch-1.8.1+cu111-cp37-cp37m-linux_x86_64.whl
├── torchaudio-0.8.1-cp37-cp37m-linux_x86_64.whl
└── torchvision-0.9.1+cu111-cp37-cp37m-linux_x86_64.whl
```

12. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1**:

```
docker build . -t horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

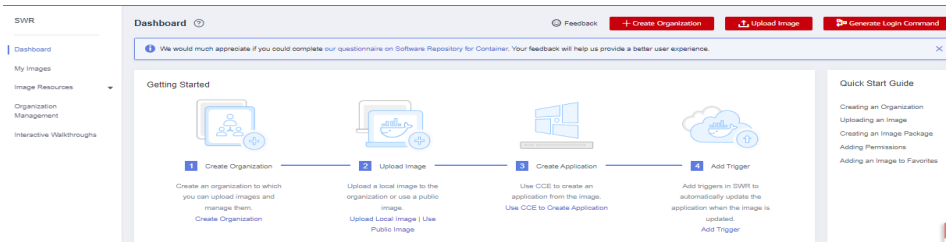
    The following log shows that the image has been created.

```
Successfully tagged horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

Step 5 Uploading the Image to SWR

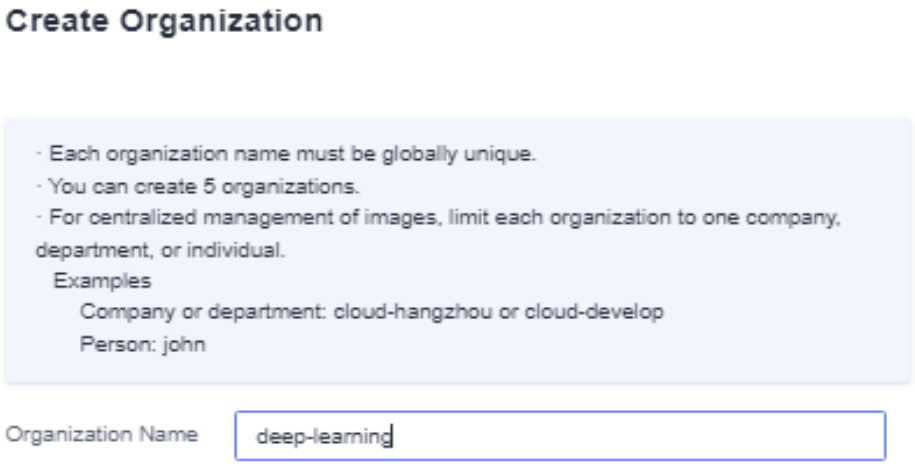
1. Log in to the SWR console and select the target region.

Figure 4-33 SWR console



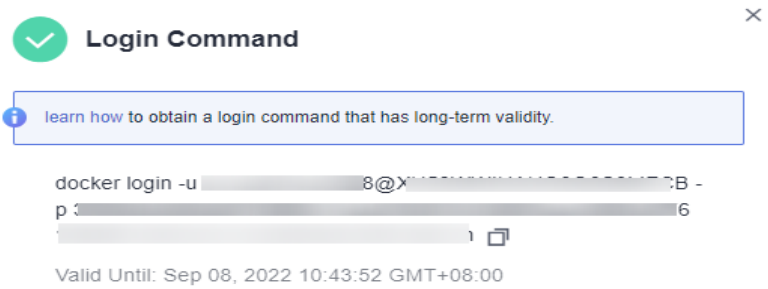
2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

Figure 4-34 Creating an organization



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

Figure 4-35 Login Command



4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.

```
# Replace the region, domain, as well as organization name deep-learning with the actual values.
sudo docker tag horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```


b. Run the following command to upload the image:

```
# Replace the region, domain, as well as organization name deep-learning with the actual values.
sudo docker push swr.{region-id}.{domain}/deep-learning/horovod-pytorch:0.22.1-1.8.1-ofed-cuda11.1
```

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

Step 6 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Uploading the Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/pytorch/demo-code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/demo-code** directory of the training container. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **bash ${MA_JOB_DIR}/demo-code/run_mpi.sh python ${MA_JOB_DIR}/demo-code/pytorch_synthetic_benchmark.py**. **demo-code** (customizable) is the last-level directory of the OBS path.
   - **Environment Variable**: Click **Add Environment Variable** and add the environment variable **MY_SSHD_PORT=38888**.
   - **Resource Pool**: Select **Public resource pools**.
   - **Resource Type**: Select **GPU**.
   - **Compute Nodes**: 1 or 2
   - **Persistent Log Saving**: enabled
   - **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/pytorch/log/**
4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

Figure 4-36 Run logs of training jobs with GPU specifications (one compute node)

```
58 Iter #0: 342.4 img/sec per GPU
59 Iter #1: 342.7 img/sec per GPU
60 Iter #2: 342.8 img/sec per GPU
61 Iter #3: 342.5 img/sec per GPU
62 Iter #4: 342.9 img/sec per GPU
63 Iter #5: 342.8 img/sec per GPU
64 Iter #6: 342.9 img/sec per GPU
65 Iter #7: 343.0 img/sec per GPU
66 Iter #8: 342.6 img/sec per GPU
67 Iter #9: 342.7 img/sec per GPU
68 Img/sec per GPU: 342.7 +-0.3
69 Total img/sec on 1 GPU(s): 342.7 +-0.3
```

Figure 4-37 Run logs of training jobs with GPU specifications (two compute nodes)

```
84 Iter #0: 115.1 img/sec per GPU
85 Iter #1: 123.5 img/sec per GPU
86 Iter #2: 115.7 img/sec per GPU
87 Iter #3: 117.4 img/sec per GPU
88 Iter #4: 120.6 img/sec per GPU
89 Iter #5: 126.9 img/sec per GPU
90 Iter #6: 119.4 img/sec per GPU
91 Iter #7: 118.4 img/sec per GPU
92 Iter #8: 122.0 img/sec per GPU
93 Iter #9: 118.2 img/sec per GPU
94 Img/sec per GPU: 119.7 +-6.8
95 Total img/sec on 2 GPU(s): 239.4 +-13.5
```

4.5 Example: Creating a Custom Image for Training (MindSpore and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is MindSpore, and the resources used for training are GPUs.

NOTE

This section applies only to training jobs of the new version.

Scenario

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

Create a container image with the following configurations and use the image to create a GPU-powered training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

- **Prerequisites**
- **Step 1 Creating an OBS Bucket and Folder**
- **Step 2 Creating a Dataset and Uploading It to OBS**
- **Step 3 Preparing the Training Script and Uploading It to OBS**
- **Step 4 Preparing a Server**
- **Step 5 Creating a Custom Image**
- **Step 6 Uploading the Image to SWR**
- **Step 7 Creating a Training Job on ModelArts**

Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-4** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS and ModelArts are in the same region.

Table 4-4 Required OBS folders

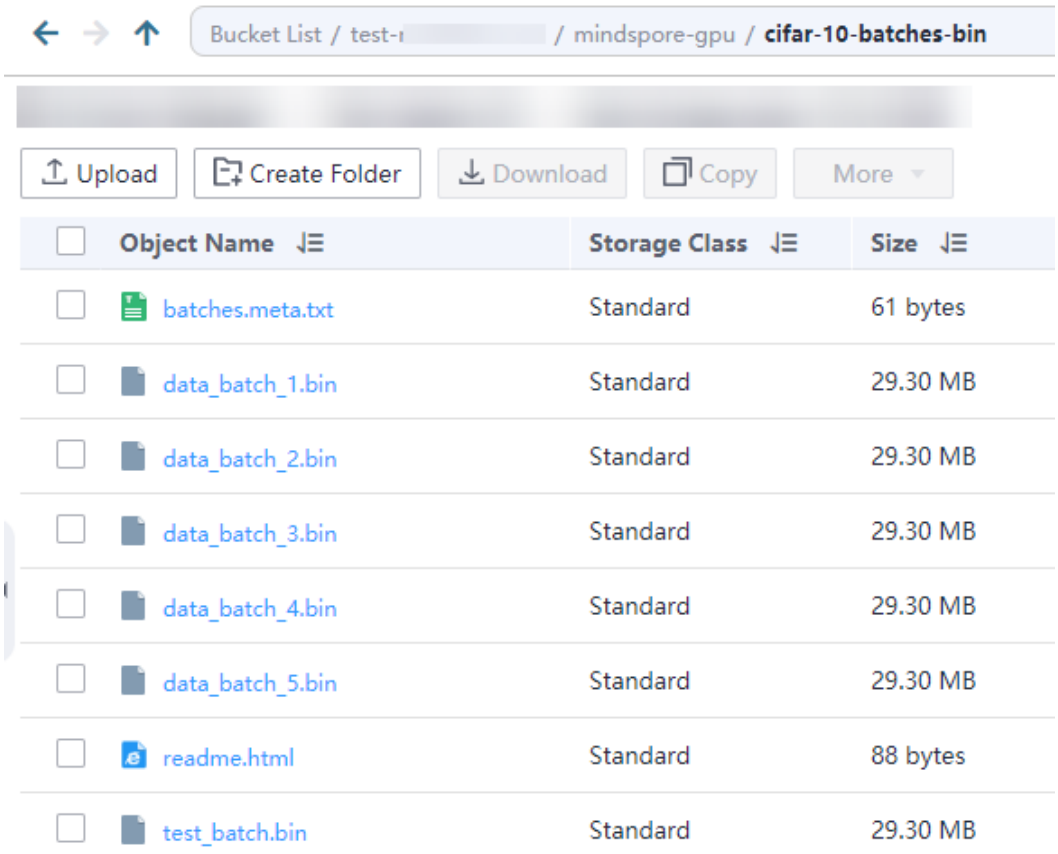
Folder	Description
obs://test-modelarts/mindspore-gpu/resnet/	Stores the training script.

Folder	Description
obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/	Stores dataset files.
obs://test-modelarts/mindspore-gpu/output/	Stores training output files.
obs://test-modelarts/mindspore-gpu/log/	Store training log files.

Step 2 Creating a Dataset and Uploading It to OBS

Go to http://www.cs.toronto.edu/~kriz/cifar.html, download **CIFAR-10 binary version (suitable for C programs)**, decompress it, and upload the decompressed data to **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/** in the OBS bucket, which is as follows.

Figure 4-38 Datasets



Step 3 Preparing the Training Script and Uploading It to OBS

Obtain the ResNet file and script **run_mpi.sh** and upload them to **obs://test-modelarts/mindspore-gpu/resnet/** in the OBS bucket.

Download the ResNet file from **https://gitee.com/mindspore/models/tree/r1.8/official/cv/resnet**.

run_mpi.sh is as follows:

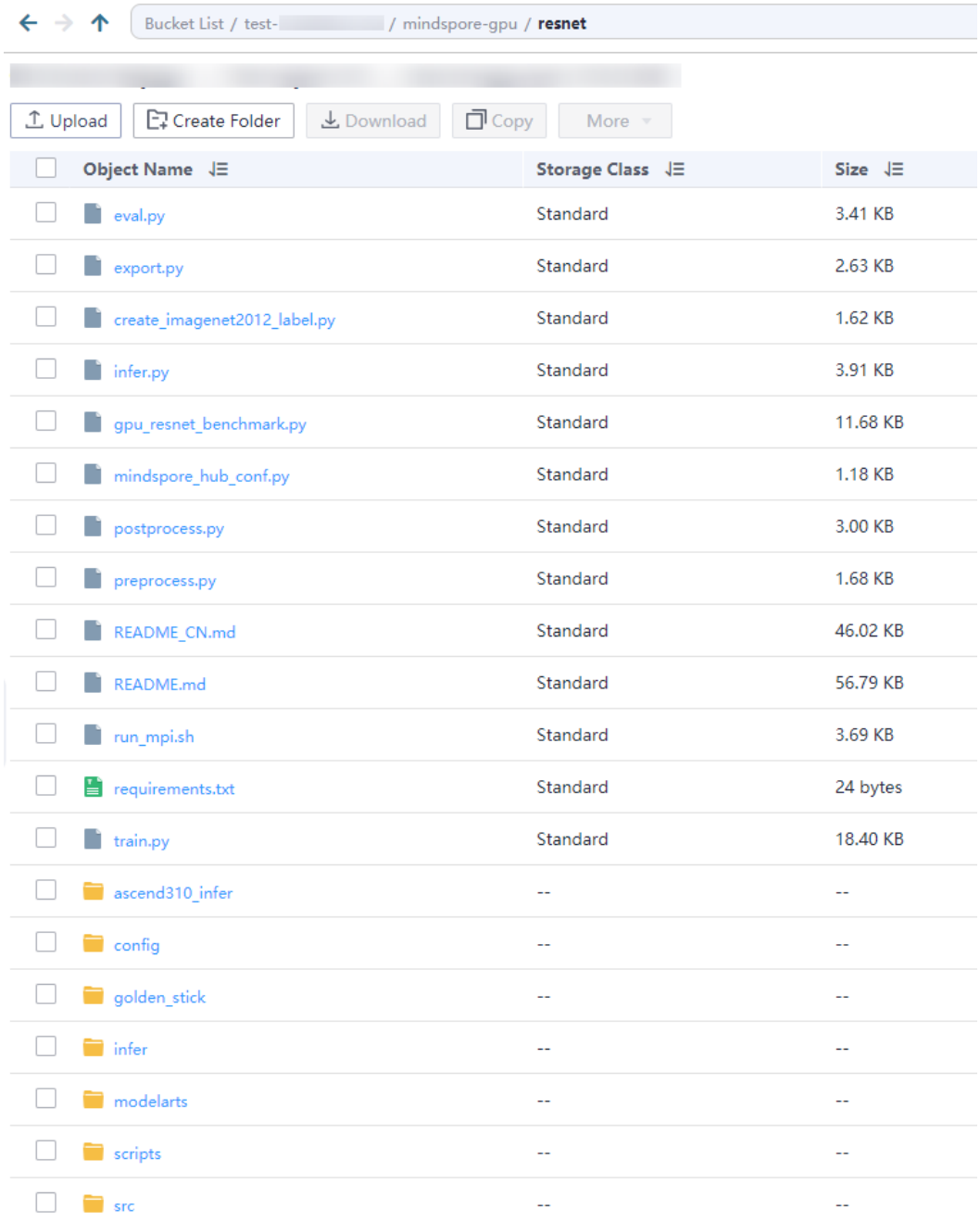
```
#!/bin/bash
MY_HOME=/home/ma-user

MY_SSHD_PORT=${MY_SSHD_PORT:-"36666"}

MY_MPI_BTL_TCP_IF=${MY_MPI_BTL_TCP_IF:-"eth0,bond0"}

MY_TASK_INDEX=${MA_TASK_INDEX:-${VC_TASK_INDEX:-${VK_TASK_INDEX}}}

MY_MPI_SLOTS=${MY_MPI_SLOTS:-"${MA_NUM_GPUS}"}

MY_MPI_TUNE_FILE="${MY_HOME}/env_for_user_process"

if [ -z ${MY_MPI_SLOTS} ]; then
    echo "[run_mpi] MY_MPI_SLOTS is empty, set it be 1"
    MY_MPI_SLOTS="1"
fi

printf "MY_HOME: ${MY_HOME}\nMY_SSHD_PORT: ${MY_SSHD_PORT}\nMY_MPI_BTL_TCP_IF: $
{MY_MPI_BTL_TCP_IF}\nMY_TASK_INDEX: ${MY_TASK_INDEX}\nMY_MPI_SLOTS: ${MY_MPI_SLOTS}\n"

env | grep -E '^MA_|^SHARED_|^S3_|^PATH|^VC_WORKER_|^SCC|^CRED' | grep -v '=$' > $
{MY_MPI_TUNE_FILE}
# add -x to each line
sed -i 's/^/-x /' ${MY_MPI_TUNE_FILE}

sed -i "s|{{MY_SSHD_PORT}}|${MY_SSHD_PORT}|g" ${MY_HOME}/etc/ssh/sshd_config

# start sshd service
bash -c "$(which sshd) -f ${MY_HOME}/etc/ssh/sshd_config"

# confirm the sshd is up
netstat -anp | grep LIS | grep ${MY_SSHD_PORT}

if [ $MY_TASK_INDEX -eq 0 ]; then
    # generate the hostfile of mpi
    for ((i=0; i<$MA_NUM_HOSTS; i++))
    do
        eval hostname=${MA_VJ_NAME}-${MA_TASK_NAME}-${i}.${MA_VJ_NAME}
        echo "[run_mpi] hostname: ${hostname}"

        ip=""
        while [ -z "$ip" ]; do
            ip=$(ping -c 1 ${hostname} | grep "PING" | sed -E 's/PING .* .([0-9.]+). .*/\1/g')
            sleep 1
        done
        echo "[run_mpi] resolved ip: ${ip}"

        # test the sshd is up
        while :
        do
            if [ cat < /dev/null >/dev/tcp/${ip}/${MY_SSHD_PORT} ]; then
                break
            fi
            sleep 1
        done

        echo "[run_mpi] the sshd of ip ${ip} is up"

        echo "${ip} slots=$MY_MPI_SLOTS" >> ${MY_HOME}/hostfile
    done

    printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"
fi
```



```
RET_CODE=0

if [ $MY_TASK_INDEX -eq 0 ]; then

    echo "[run_mpi] start exec command time: "$(date +"%Y-%m-%d-%H:%M:%S")

    np=$(( ${MA_NUM_HOSTS} * ${MY_MPI_SLOTS} ))

    echo "[run_mpi] command: mpirun -np ${np} -hostfile ${MY_HOME}/hostfile -mca plm_rsh_args \"-p $
{MY_SSHD_PORT}\" -tune ${MY_MPI_TUNE_FILE} ... $@"

    # execute mpirun at worker-0
    # mpirun
    mpirun \
        -np ${np} \
        -hostfile ${MY_HOME}/hostfile \
        -mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -tune ${MY_MPI_TUNE_FILE} \
        -bind-to none -map-by slot \
        -x NCCL_DEBUG=INFO -x NCCL_SOCKET_IFNAME=${MY_MPI_BTL_TCP_IF} -x
NCCL_SOCKET_FAMILY=AF_INET \
        -x HOROVOD_MPI_THREADS_DISABLE=1 \
        -x LD_LIBRARY_PATH \
        -mca pml ob1 -mca btl ^openib -mca plm_rsh_no_tree_spawn true \
        "$@"

    RET_CODE=$?

    if [ $RET_CODE -ne 0 ]; then
        echo "[run_mpi] exec command failed, exited with $RET_CODE"
    else
        echo "[run_mpi] exec command successfully, exited with $RET_CODE"
    fi

    # stop 1...N worker by killing the sleep proc
    sed -i '1d' ${MY_HOME}/hostfile
    if [ `cat ${MY_HOME}/hostfile | wc -l` -ne 0 ]; then
        echo "[run_mpi] stop 1 to (N - 1) worker by killing the sleep proc"

        sed -i 's/${MY_MPI_SLOTS}/1/g' ${MY_HOME}/hostfile
        printf "[run_mpi] hostfile:\n`cat ${MY_HOME}/hostfile`\n"

        mpirun \
        --hostfile ${MY_HOME}/hostfile \
        --mca btl_tcp_if_include ${MY_MPI_BTL_TCP_IF} \
        --mca plm_rsh_args "-p ${MY_SSHD_PORT}" \
        -x PATH -x LD_LIBRARY_PATH \
        pkill sleep \
        > /dev/null 2>&1
    fi

    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
else
    echo "[run_mpi] the training log is in worker-0"
    sleep 365d
    echo "[run_mpi] exit time: "$(date +"%Y-%m-%d-%H:%M:%S")
fi

exit $RET_CODE
```

The following figure shows **obs://test-modelarts/mindspore-gpu/resnet/**, including the ResNet file and **run_mpi.sh**.

Figure 4-39 ResNet file and run_mpi.sh

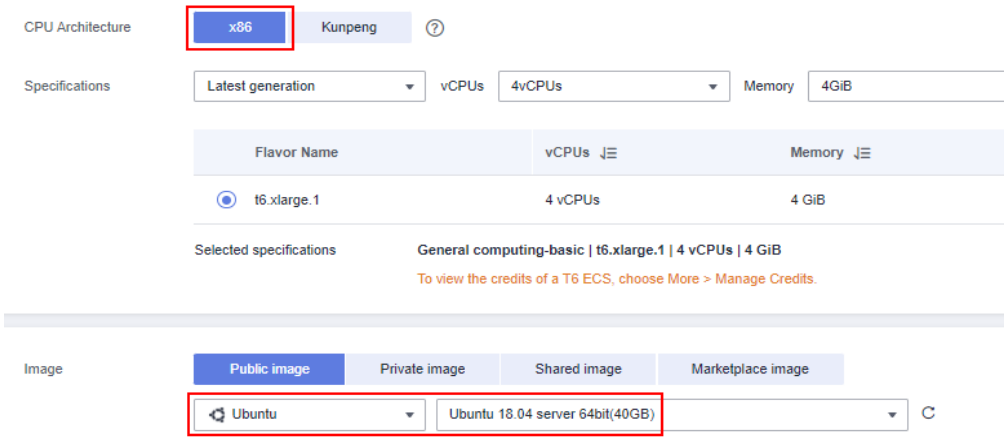


Step 4 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

Figure 4-40 Creating an ECS using a public image (x86)



Step 5 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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

   If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:

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

   The following information is displayed:

   ```
   Engine:
    Version:        18.09.0
   ```

   NOTE

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

3. Create a folder named **context**.

   ```
   mkdir -p context
   ```

4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:

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


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

5. Download **mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl** from **https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.8.1/MindSpore/gpu/x86_64/cuda-11.1/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl**.
6. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.
7. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:

```
# The server on which the container image is created must access the Internet.

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.1.1-devel-ubuntu18.04 AS builder

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

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the whl file using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl \
    easydict PyYAML

# Create the container image.
FROM nvidia/cuda:11.1.1-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# Install the vim, cURL, net-tools, MLNX_OFED, and SSH tools obtained from Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping libfile-find-rule-perl-perl \
    openssh-client openssh-server && \
    ssh -V && \
    mkdir -p /run/sshd && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1
```



```
libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# Install the Open MPI 3.0.0 file obtained from Horovod v0.22.1.
# https://github.com/horovod/horovod/blob/v0.22.1/docker/horovod/Dockerfile
# https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz
COPY openmpi-3.0.0-bin.tar.gz /tmp
RUN cd /usr/local && \
    tar -zxf /tmp/openmpi-3.0.0-bin.tar.gz && \
    ldconfig && \
    mpirun --version

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the basic container image. User ma-user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

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

# Configure sshd to support SSH password-free login.
RUN MA_HOME=/home/ma-user && \
    # setup sshd dir
    mkdir -p ${MA_HOME}/etc && \
    ssh-keygen -f ${MA_HOME}/etc/ssh_host_rsa_key -N '' -t rsa  && \
    mkdir -p ${MA_HOME}/etc/ssh ${MA_HOME}/var/run  && \
    # setup sshd config (listen at {{MY_SSHD_PORT}} port)
    echo "Port {{MY_SSHD_PORT}}\n\
HostKey ${MA_HOME}/etc/ssh_host_rsa_key\n\
AuthorizedKeysFile ${MA_HOME}/.ssh/authorized_keys\n\
PidFile ${MA_HOME}/var/run/sshd.pid\n\
StrictModes no\n\
UsePAM no" > ${MA_HOME}/etc/ssh/sshd_config && \
    # generate ssh key
    ssh-keygen -t rsa -f ${MA_HOME}/.ssh/id_rsa -P '' && \
    cat ${MA_HOME}/.ssh/id_rsa.pub >> ${MA_HOME}/.ssh/authorized_keys && \
    # disable ssh host key checking for all hosts
    echo "Host *\n\
  StrictHostKeyChecking no" > ${MA_HOME}/.ssh/config

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

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

8. Download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, click **Download**, set **Version** to **5.4-3.5.8.0-LTS**, **OSDistributionVersion** to **Ubuntu 18.04**, and **Architecture** to **x86_64**, and download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Download **openmpi-3.0.0-bin.tar.gz**.

Download **openmpi-3.0.0-bin.tar.gz** from https://github.com/horovod/horovod/files/1596799/openmpi-3.0.0-bin.tar.gz.

10. Store the Dockerfile and Miniconda3 installation file in the **context** folder, which is as follows:

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── mindspore_gpu-1.8.1-cp37-cp37m-linux_x86_64.whl
├── openmpi-3.0.0-bin.tar.gz
└── pip.conf
```

11. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **mindspore:1.8.1-ofed-cuda11.1**:

```
docker build . -t mindspore:1.8.1-ofed-cuda11.1
```

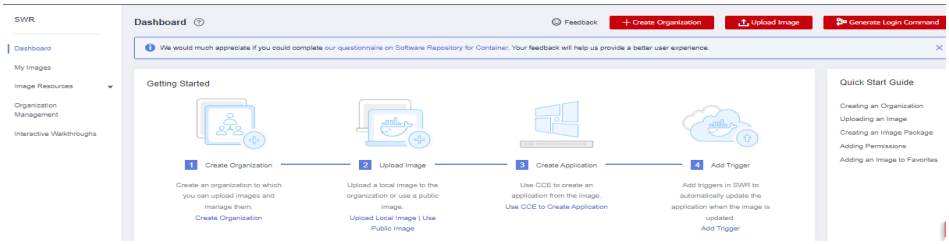
The following log shows that the image has been created.

```
Successfully tagged mindspore:1.8.1-ofed-cuda11.1
```

Step 6 Uploading the Image to SWR

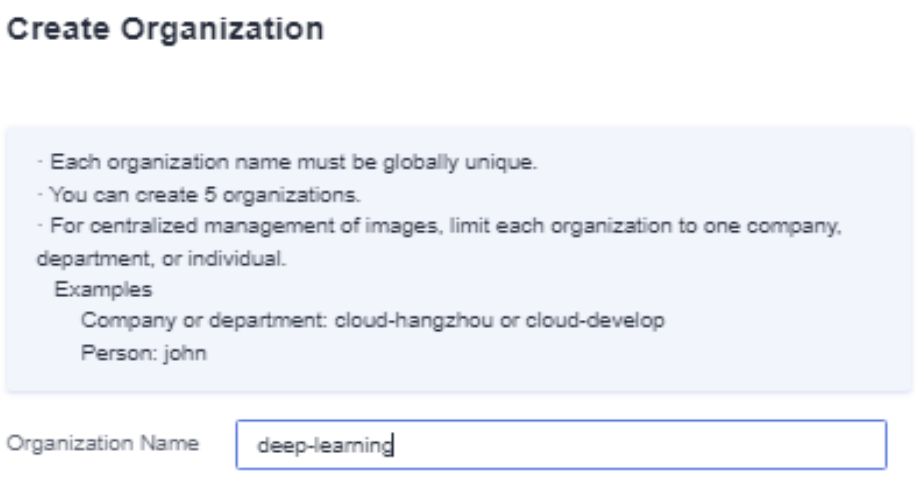
1. Log in to the SWR console and select the target region.

Figure 4-41 SWR console



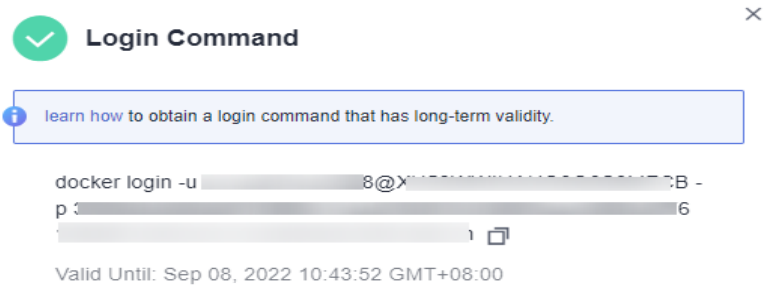
2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

Figure 4-42 Creating an organization



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

Figure 4-43 Login Command



4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag mindspore:1.8.1-ofed-cuda11.1 swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
   ```
   b. Run the following command to upload the image:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/mindspore:1.8.1-ofed-cuda11.1
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

Step 7 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 6 Uploading the Image to SWR**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/mindspore-gpu/resnet/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/resnet** directory of the training container. **resnet** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **bash ${MA_JOB_DIR}/resnet/run_mpi.sh python ${MA_JOB_DIR}/resnet/train.py**. **resnet** (customizable) is the last-level directory of the OBS path.
   - **Training Input**: Click **Add Training Input**. Enter **data_path** for the name, select the OBS path to the target dataset, for example, **obs://test-modelarts/mindspore-gpu/cifar-10-batches-bin/**, and set **Obtained from** to **Hyperparameters**.

- **Training Output**: Click **Add Training Output**. Enter **output_path** for the name, select an OBS path for storing training outputs, for example, **obs://test-modelarts/mindspore-gpu/output/**, and set **Obtained from** to **Hyperparameters** and **Predownload** to **No**.
- **Hyperparameters**: Click **Add Hyperparameter** and add the following hyperparameters:
  - run_distribute=True
  - device_num=1 (Set this parameter based on the number of GPUs in the instance flavors.)
  - device_target=GPU
  - epoch_size=2
- **Environment Variable**: Click **Add Environment Variable** and add the environment variable **MY_SSHD_PORT=38888**.
- **Resource Pool**: Select **Public resource pools**.
- **Resource Type**: Select **GPU**.
- **Compute Nodes**: 1 or 2
- **Persistent Log Saving**: enabled
- **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mindspore-gpu/log/**

4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

   Figure 4-44 Run logs of training jobs with GPU specifications (one compute node)

```
127 epoch: 1 step: 1875, loss is 1.4800076
128 Train epoch time: 369422.027 ms, per step time: 197.025 ms
129 epoch: 2 step: 1875, loss is 1.0306032
130 Train epoch time: 66996.087 ms, per step time: 35.731 ms
```


Figure 4-45 Run logs of training jobs with GPU specifications (two compute nodes)

```
epoch: 1 step: 937, loss is 1.8482083
epoch: 1 step: 937, loss is 1.342748
Train epoch time: 488492.170 ms, per step time: 521.336 ms
Train epoch time: 488490.528 ms, per step time: 521.335 ms
epoch: 2 step: 937, loss is 0.9150252
Train epoch time: 180200.654 ms, per step time: 192.317 ms
epoch: 2 step: 937, loss is 0.9933052
Train epoch time: 180199.969 ms, per step time: 192.316 ms
172.16.0.45 slots=1
```

4.6 Example: Creating a Custom Image for Training (TensorFlow and GPUs)

This section describes how to create an image and use it for training on ModelArts. The AI engine used in the image is TensorFlow, and the resources used for training are GPUs.

NOTE

This section applies only to training jobs of the new version.

Scenario

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

Create a container image with the following configurations and use the image to create a GPU-powered training job on ModelArts:

- ubuntu-18.04
- cuda-11.2
- python-3.7.13
- mlnx ofed-5.4
- tensorflow gpu-2.10.0

Procedure

Before using a custom image to create a training job, you need to be familiar with Docker and have development experience.

1. **Prerequisites**
2. **Step 1 Creating an OBS Bucket and Folder**
3. **Step 2 Creating a Dataset and Uploading It to OBS**
4. **Step 3 Preparing the Training Script and Uploading It to OBS**
5. **Step 4 Preparing a Server**
6. **Step 5 Creating a Custom Image**
7. **Step 6 Uploading the Image to SWR**

8. **Step 7 Creating a Training Job on ModelArts**

Prerequisites

You have registered a Huawei Cloud account. The account is not in arrears or frozen.

Step 1 Creating an OBS Bucket and Folder

Create a bucket and folders in OBS for storing the sample dataset and training code. **Table 4-5** lists the folders to be created. Replace the bucket name and folder names in the example with actual names.

For details, see **Creating a Bucket** and **Creating a Folder**.

Ensure that the OBS and ModelArts are in the same region.

Table 4-5 Required OBS folders

Folder	Description
obs://test-modelarts/tensorflow/code/	Stores the training script.
obs://test-modelarts/tensorflow/data/	Stores dataset files.
obs://test-modelarts/tensorflow/log/	Store training log files.

Step 2 Creating a Dataset and Uploading It to OBS

Download **mnist.npz** from **https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz**, and upload it to **obs://test-modelarts/tensorflow/data/** in the OBS bucket.

Step 3 Preparing the Training Script and Uploading It to OBS

Obtain the training script **mnist.py** and upload it to **obs://test-modelarts/tensorflow/code/** in the OBS bucket.

mnist.py is as follows:

```
import argparse
import tensorflow as tf

parser = argparse.ArgumentParser(description='TensorFlow quick start')
parser.add_argument('--data_url', type=str, default="./Data", help='path where the dataset is saved')
args = parser.parse_args()

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data(args.data_url)
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
```



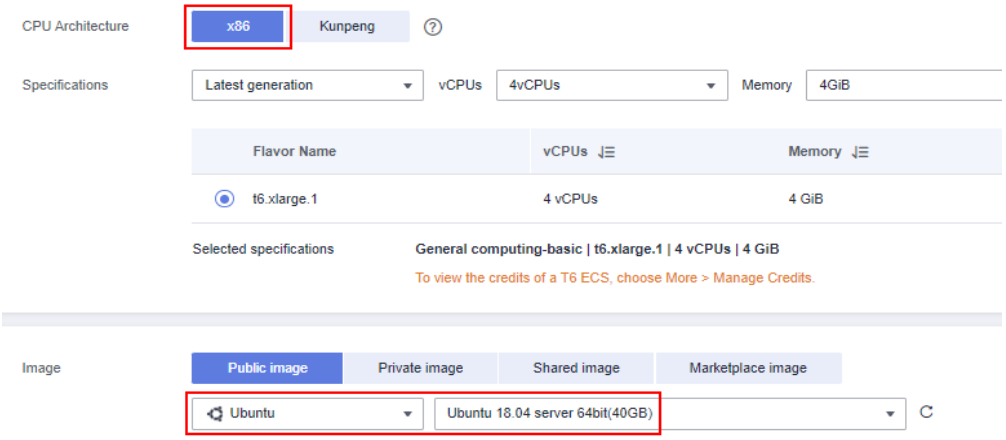
```
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
```

Step 4 Preparing a Server

Obtain a Linux x86_64 server running Ubuntu 18.04. Either an ECS or your local PC will do.

For details about how to purchase an ECS, see **Purchasing and Logging In to a Linux ECS**. Select a public image. An Ubuntu 18.04 image is recommended.

Figure 4-46 Creating an ECS using a public image (x86)



Step 5 Creating a Custom Image

Create a container image with the following configurations and use the image to create a training job on ModelArts:

- ubuntu-18.04
- cuda-11.1
- python-3.7.13
- mlnx ofed-5.4
- mindspore gpu-1.8.1

This section describes how to write a Dockerfile to create a custom image.

1. Install Docker.

   The following uses Linux x86_64 as an example to describe how to obtain a Docker installation package. For more details about how to install Docker, see **official Docker documents**. Run the following command to install Docker:

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


If the **docker images** command can be executed, Docker has been installed. In this case, skip this step.

2. Check the Docker Engine version. Run the following command:
```
docker version | grep -A 1 Engine
```
The following information is displayed:
```
Engine:
 Version:       18.09.0
```

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

3. Create a folder named **context**.
```
mkdir -p context
```
4. Obtain the **pip.conf** file. In this example, the pip source provided by Huawei Mirrors is used, which is as follows:
```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

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

5. Download **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl**.

   Download **tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl** from **https://pypi.org/project/tensorflow-gpu/2.10.0/#files**.

6. Download the Miniconda3 installation file.

   Download the Miniconda3 py37 4.12.0 installation file (Python 3.7.13) from **https://repo.anaconda.com/miniconda/Miniconda3-py37_4.12.0-Linux-x86_64.sh**.

7. Write the container image Dockerfile.

   Create an empty file named **Dockerfile** in the **context** folder and copy the following content to the file:
```
# The server on which the container image is created must access the Internet.

# Base container image at https://github.com/NVIDIA/nvidia-docker/wiki/CUDA
#
# https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds
# require Docker Engine >= 17.05
#
# builder stage
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04 AS builder

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

# Use the PyPI configuration obtained from Huawei Mirrors.
RUN mkdir -p /root/.pip/
COPY pip.conf /root/.pip/pip.conf

# Copy the installation files to the /tmp directory in the base container image.
COPY Miniconda3-py37_4.12.0-Linux-x86_64.sh /tmp
COPY tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl /tmp

# https://conda.io/projects/conda/en/latest/user-guide/install/linux.html#installing-on-linux
# Install Miniconda3 in the /home/ma-user/miniconda3 directory of the base container image.
```



```
RUN bash /tmp/Miniconda3-py37_4.12.0-Linux-x86_64.sh -b -p /home/ma-user/miniconda3

# Install the TensorFlow .whl file using default Miniconda3 Python environment /home/ma-user/miniconda3/bin/pip.
RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir \
    /tmp/tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl

RUN cd /tmp && \
    /home/ma-user/miniconda3/bin/pip install --no-cache-dir keras==2.10.0

# Create the container image.
FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu18.04

COPY MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz /tmp

# Install the vim, cURL, net-tools, and MLNX_OFED tools obtained from Huawei Mirrors.
RUN cp -a /etc/apt/sources.list /etc/apt/sources.list.bak && \
    sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g" /etc/apt/sources.list && \
    echo > /etc/apt/apt.conf.d/00skip-verify-peer.conf "Acquire { https::Verify-Peer false }" && \
    apt-get update && \
    apt-get install -y vim curl net-tools iputils-ping && \
    # mlnx ofed
    apt-get install -y python libfuse2 dpatch libnl-3-dev autoconf libnl-route-3-dev pciutils libnuma1 libpci3 m4 libelf1 debhelper automake graphviz bison lsof kmod libusb-1.0-0 swig libmnl0 autotools-dev flex chrpath libltdl-dev && \
    cd /tmp && \
    tar -xvf MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz && \
    MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64/mlnxofedinstall --user-space-only --basic --without-fw-update -q && \
    cd - && \
    rm -rf /tmp/* && \
    apt-get clean && \
    mv /etc/apt/sources.list.bak /etc/apt/sources.list && \
    rm /etc/apt/apt.conf.d/00skip-verify-peer.conf

# Add user ma-user (UID = 1000, GID = 100).
# A user group whose GID is 100 exists in the base container image. User ma-user can directly run the following command:
RUN useradd -m -d /home/ma-user -s /bin/bash -g 100 -u 1000 ma-user

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

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

# Configure the preset environment variables of the container image.
# Set PYTHONUNBUFFERED to 1 to prevent log loss.
ENV PATH=/home/ma-user/miniconda3/bin:$PATH \
    LD_LIBRARY_PATH=/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:$LD_LIBRARY_PATH \
    PYTHONUNBUFFERED=1
```

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

8. Download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

   Go to https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/, click **Download**, set **Version** to **5.4-3.5.8.0-LTS**, **OSDistributionVersion** to **Ubuntu 18.04**, and **Architecture** to **x86_64**, and download **MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz**.

9. Store the Dockerfile and Miniconda3 installation file in the **context** folder, which is as follows:

```
context
├── Dockerfile
├── MLNX_OFED_LINUX-5.4-3.5.8.0-ubuntu18.04-x86_64.tgz
```



```
├── Miniconda3-py37_4.12.0-Linux-x86_64.sh
├── pip.conf
└── tensorflow_gpu-2.10.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
```

10. Create the container image. Run the following command in the directory where the Dockerfile is stored to build the container image **tensorflow:2.10.0-ofed-cuda11.2**:

```
docker build . -t tensorflow:2.10.0-ofed-cuda11.2
```

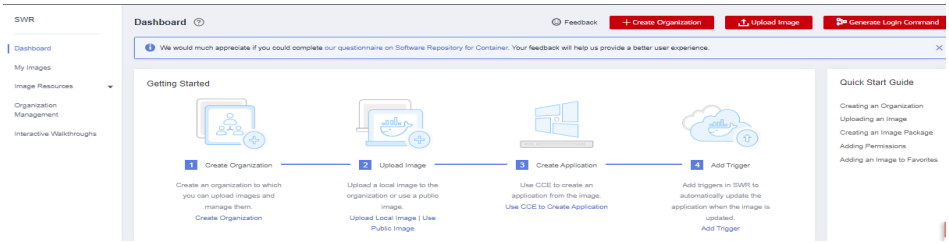
The following log shows that the image has been created.

```
Successfully tagged tensorflow:2.10.0-ofed-cuda11.2
```

Step 6 Uploading the Image to SWR

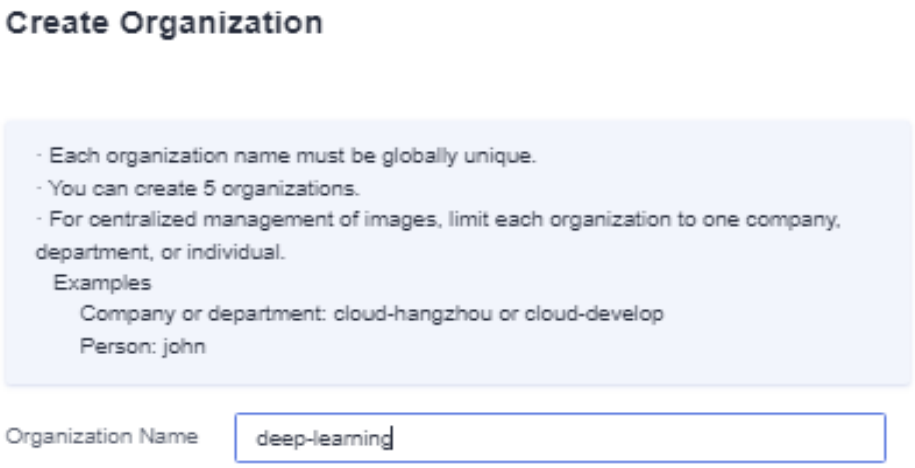
1. Log in to the SWR console and select the target region.

Figure 4-47 SWR console



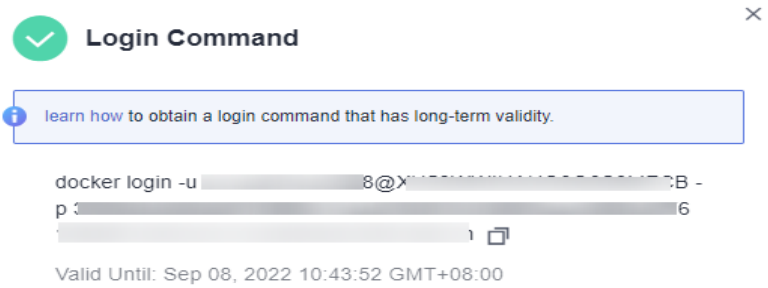
2. Click **Create Organization** in the upper right corner and enter an organization name to create an organization. Customize the organization name. Replace the organization name **deep-learning** in subsequent commands with the actual organization name.

Figure 4-48 Creating an organization



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

Figure 4-49 Login Command



4. Log in to the local environment as the **root** user and enter the login command.
5. Upload the image to SWR.
   a. Tag the uploaded image.
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker tag tensorflow:2.10.0-ofed-cuda11.2 swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
   b. Run the following command to upload the image:
   ```
   # Replace the region, domain, as well as organization name deep-learning with the actual values.
   sudo docker push swr.{region-id}.{domain}/deep-learning/tensorflow:2.10.0-ofed-cuda11.2
   ```
6. After the image is uploaded, choose **My Images** in navigation pane on the left of the SWR console to view the uploaded custom images.

Step 7 Creating a Training Job on ModelArts

1. Log in to the ModelArts management console, check whether access authorization has been configured for your account. For details, see **Configuring Agency Authorization**. If you have been authorized using access keys, clear the authorization and configure agency authorization.
2. In the navigation pane, choose **Training Management** > **Training Jobs**. The training job list is displayed by default.
3. Click **Create Training Job**. On the page that is displayed, configure parameters and click **Next**.
   - **Created By**: **Custom algorithms**
   - **Boot Mode**: **Custom images**
   - Image path: image created in **Step 5 Creating a Custom Image**.
   - **Code Directory**: directory where the boot script file is stored in OBS, for example, **obs://test-modelarts/tensorflow/code/**. The training code is automatically downloaded to the **${MA_JOB_DIR}/code** directory of the training container. **code** (customizable) is the last-level directory of the OBS path.
   - **Boot Command**: **python ${MA_JOB_DIR}/code/mnist.py**. **code** (customizable) is the last-level directory of the OBS path.
   - **Training Input**: Click **Add Training Input**. Enter **data_path** for the name, select the OBS path to **mnist.npz**, for example, **obs://test-modelarts/tensorflow/data/mnist.npz**, and set **Obtained from** to **Hyperparameters**.
   - **Resource Pool**: Select **Public resource pools**.

- **Resource Type**: Select **GPU**.
- **Compute Nodes**: Enter **1**.
- **Persistent Log Saving**: enabled
- **Job Log Path**: OBS path to stored training logs, for example, **obs://test-modelarts/mindspore-gpu/log/**

4. Confirm the configurations of the training job and click **Submit**.
5. Wait until the training job is created.

   After you submit the job creation request, the system will automatically perform operations on the backend, such as downloading the container image and code directory and running the boot command. A training job requires a certain period of time for running. The duration ranges from dozens of minutes to several hours, varying depending on the service logic and selected resources. After the training job is executed, the log similar to the following is output.

   Figure 4-50 Run logs of training jobs with GPU specifications

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


5 Model Inference

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

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

The process is as follows:

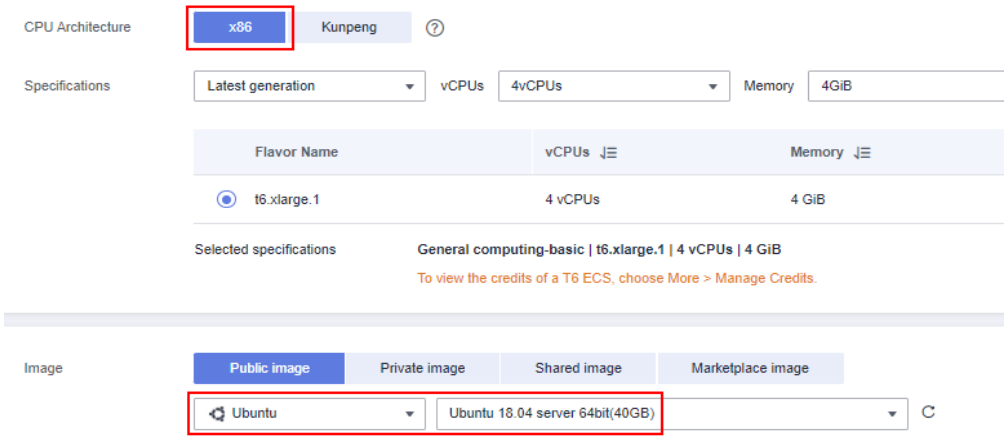
1. **Building an Image Locally**: Create a custom image package locally. For details, see **Custom Image Specifications for Creating AI Applications**.
2. **Verifying the Image Locally and Uploading It to SWR**: Verify the APIs of the custom image and upload the custom image to SWR.
3. **Using the Custom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

Building an Image Locally

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

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 5-1 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 .
```

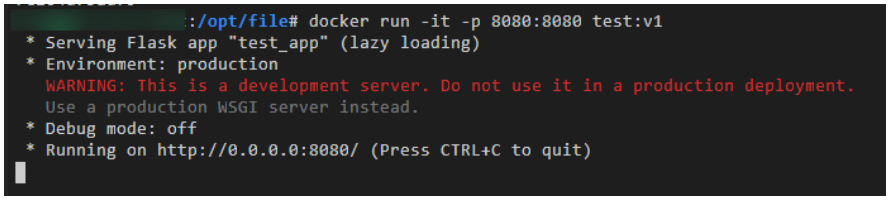
5. Run **docker images** to view the custom image you have created.

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 5-2 Starting a custom image

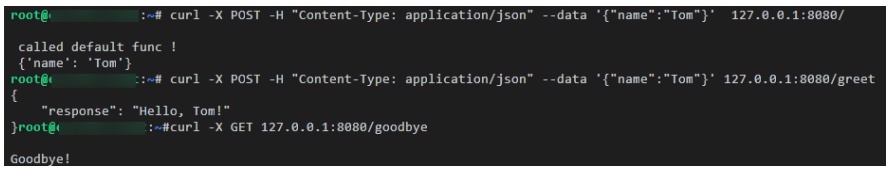


2. Open another terminal and run the following commands to test the functions of the three APIs of the custom image:

```
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
```

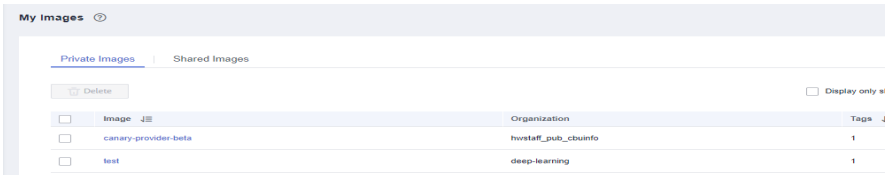
If information similar to the following is displayed, the function verification is successful.

Figure 5-3 Testing API functions



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

Figure 5-4 Uploaded images

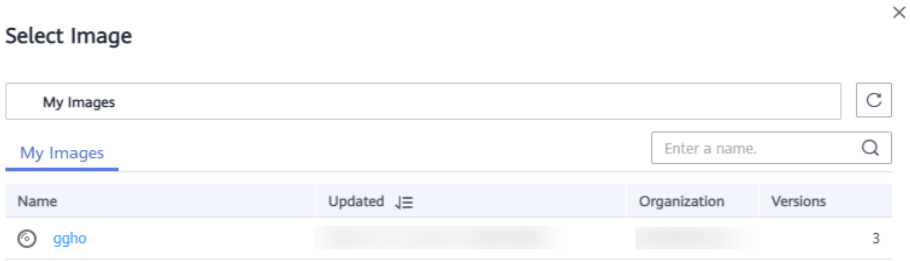


Using the Custom Image to Create an AI Application

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

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

    Figure 5-5 Created private image

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

  The configuration file is as follows:

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

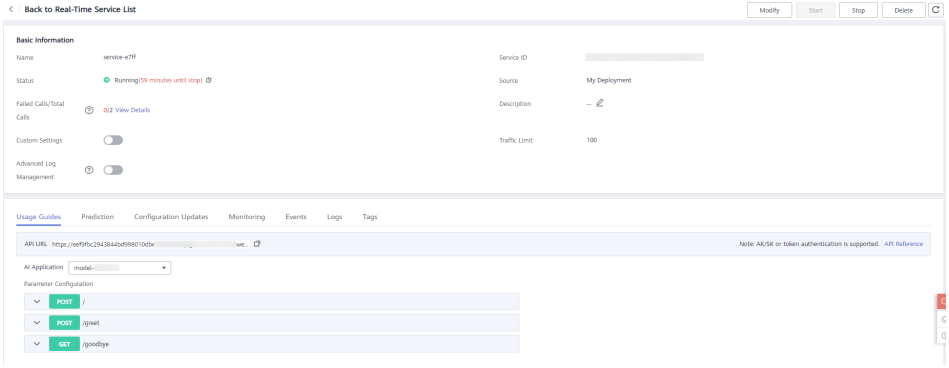


```
        }
    },
{
        "url": "/goodbye",
        "method": "get",
        "request": {
            "Content-type": "application/json"
        },
        "response": {
            "Content-type": "application/json"
        }
    }
]
```

Deploying the AI Application as a Real-Time Service

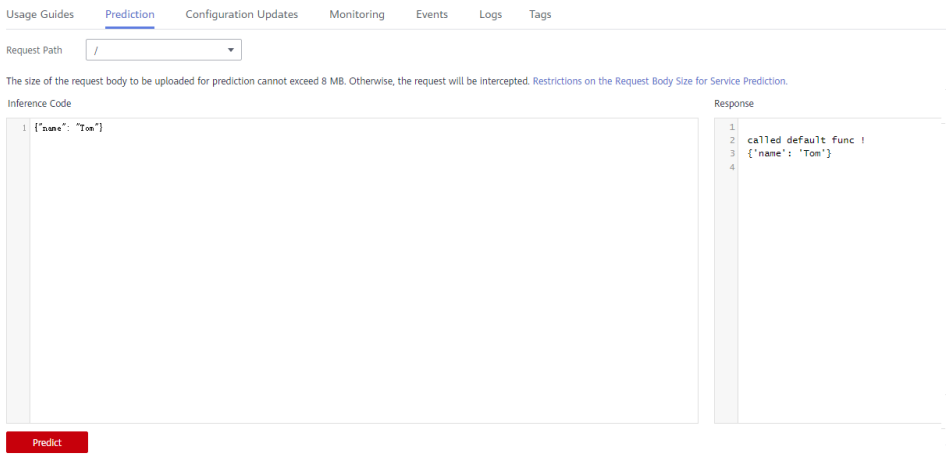
1. Deploy the AI application as a real-time service. For details, see **Deploying as a Real-Time Service**.
2. View the details about the real-time service.

Figure 5-6 Usage Guides



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

Figure 5-7 Accessing a real-time service



5.2 Enabling an Inference Service to Access the Internet

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

Application Scenarios

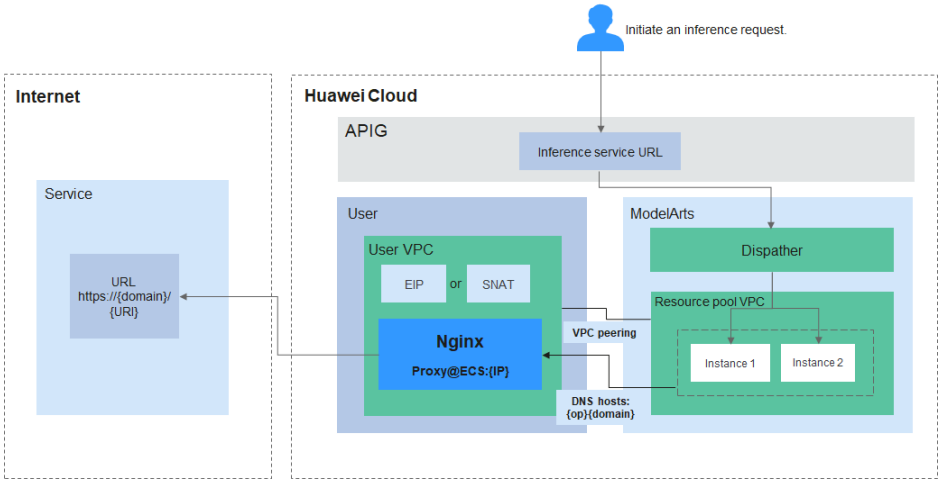
An inference service accesses the Internet in the following scenarios:

- 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 5-8 Networking for an inference service to access the Internet



Procedure

- **Configure the network for the ModelArts resource pool.**
- **Install and configure a forward proxy for your VPC.**
- **Configure the DNS proxy and Internet access URL in the algorithm image.**

Step 1 Configure the network for the ModelArts resource pool.

When purchasing a dedicated resource pool, you can select inference services in **Job Type**. In this case, the selected network must be accessible to the target VPC.

Figure 5-9 Purchasing a dedicated resource pool

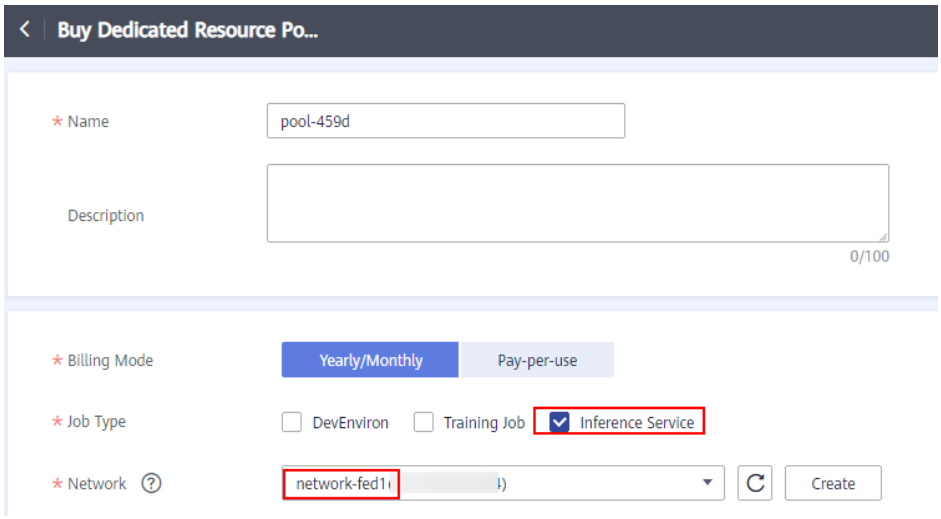
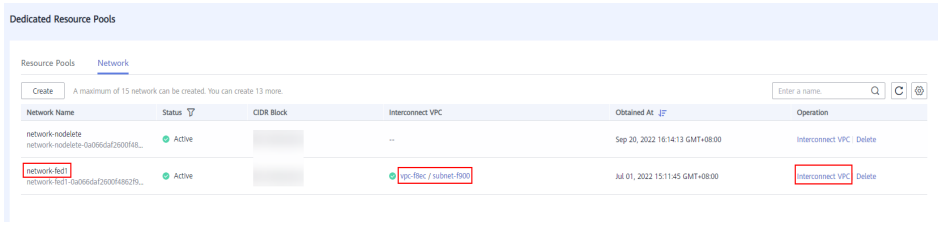


Figure 5-10 Interconnecting the VPC



Interconnecting a VPC enables the ModelArts resource pool to exchange data with your VPC.

Step 2 Install and configure a forward proxy for your VPC.

Before installing a forward proxy, purchase an ECS with the latest Ubuntu image and bind an EIP to the ECS. Then, log in to the ECS, and install and configure a squid forward proxy.

1. If Docker is not installed, run the following command to install it:
   ```
   curl -sSL https://get.daocloud.io/docker | sh
   ```
2. Pull the squid image.
   ```
   docker pull ubuntu/squid
   ```
3. Create a host directory and configure **whitelist.conf** and **squid.conf**.

   Create a host directory:

   ```
   mkdir –p /etc/squid/
   ```

   Add the **whitelist.conf** configuration file. The content is the addresses that can be accessed. For example:

   ```
   .apig.cn-east-3.huaweicloudapis.com
   ```

   Add the **squid.conf** configuration file, which includes the following:

   ```
   # 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
```

Set the permissions on the host directory and configuration files:

```
chmod 640 -R /etc/squid
```

4. Start a squid instance.
   ```
   docker run -d --name squid -e TZ=UTC -v /etc/squid:/etc/squid -p 3128:3128 ubuntu/squid:latest
   ```
5. If **whitelist.conf** or **squid.conf** is updated, go to the container and update the squid.
   ```
   docker exec –it squid bash
   root@{container_id}:/# squid -k reconfigure
   ```

Step 3 Configure the DNS proxy and Internet access URL in the algorithm image.

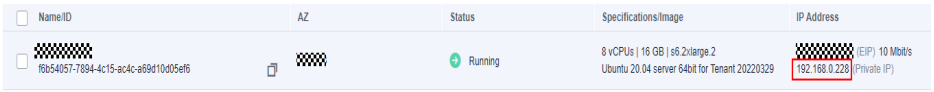
1. Set the proxy.

   In the code, specify the private IP address and port of the proxy server, as shown in the following:

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

   The following figure shows how to obtain the private IP address of a server.

   Figure 5-11 Private IP address

2. Configure the Internet access URL.

   In the inference code, use the service URL to send a service request, for example:

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

----End

5.3 End-to-End O&M of 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.

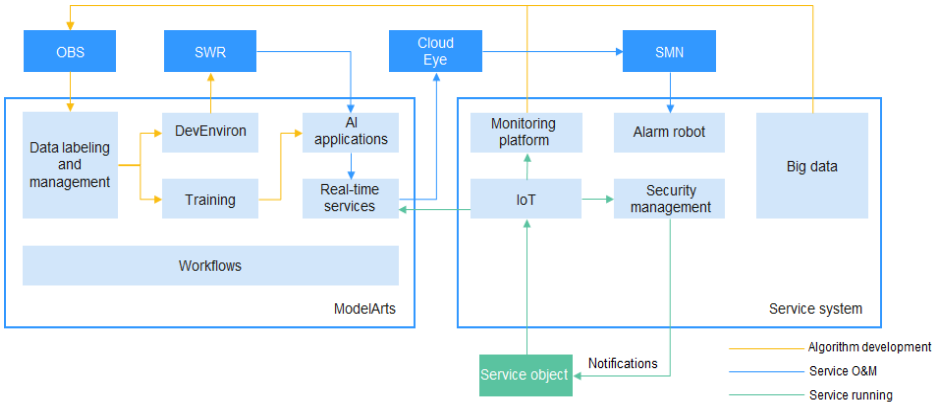
Overview

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 AI application images using a development environment.
- During service O&M, use an image to create an AI application and deploy the AI application 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.

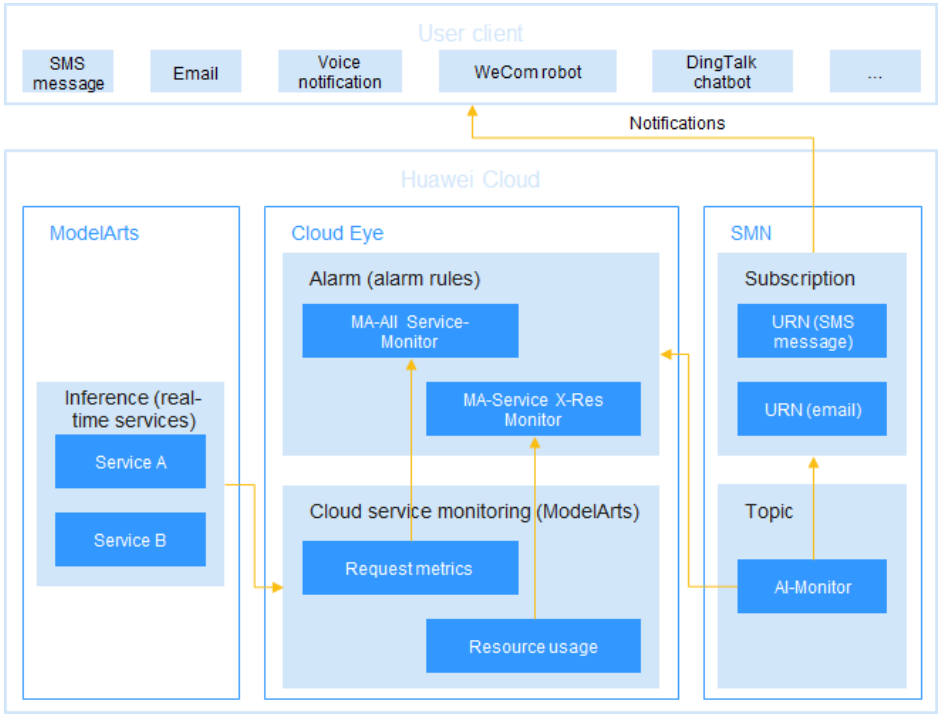
- During service running, access real-time service requests into the service system and then configure service logic and monitoring.

Figure 5-12 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 5-13 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.

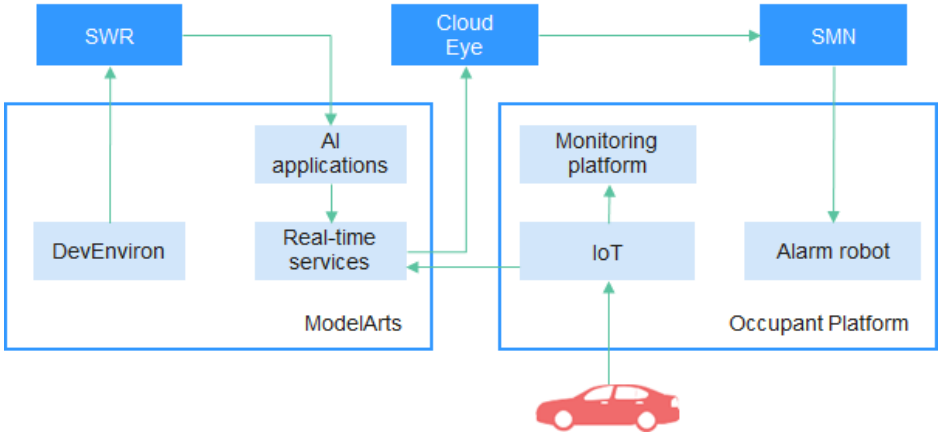
Constraints

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 5-14 Occupant safety algorithm implementation



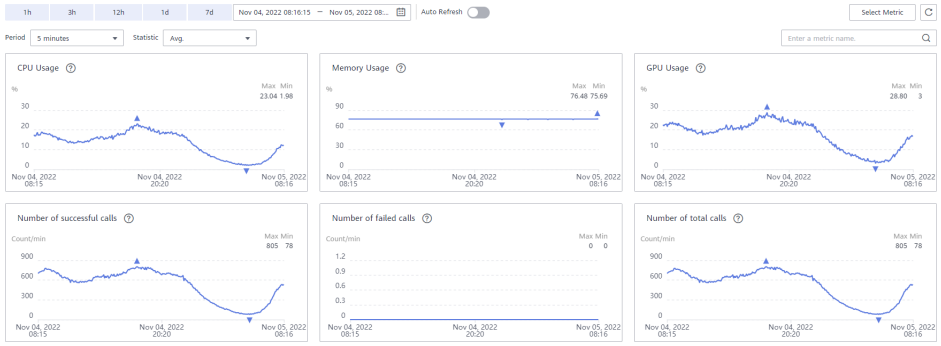
Step 1 Use a locally developed model to create a custom image and use the image to create an AI application on ModelArts. For details, see **Creating a Custom Image and Using It to Create an AI Application**.

Step 2 On the ModelArts management console, deploy the created AI application 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 5-15 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

5.4 Creating an AI Application Using a Custom Engine

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

Before deploying such an AI application as a service, ModelArts downloads the SWR image to the cluster and starts the image as a container as the user whose UID is 1000 and GID is 100. Then, ModelArts downloads the OBS file to the **/home/mind/model** directory in the container and runs the boot command preset in the SWR image. The service available to port 8080 in the container is automatically registered with APIG. You can access the service through the APIG URL.

Specifications for Using a Custom Engine to Create an AI Application

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

- SWR image specifications
  - A common user named **ma-user** in group **ma-group** must be built in the SWR image. Additionally, the UID and GID of the user must be 1000 and 100, respectively. The following is the dockerfile command for the built-in user:
    ```
    groupadd -g 100 ma-group && useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user
    ```
  - Specify a command for starting the image. In the dockerfile, specify **cmd**. The following shows an example:
    ```
    CMD sh /home/mind/run.sh
    ```
    Customize the 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
    ```
  - The service must be HTTPS enabled, and it is available on port 8080. For details, see the **HTTPS example**.
  - (Optional) On port 8080, enable health check with URL **/health**. (The health check URL must be **/health**.)
- OBS model package specifications

  The name of the model package must be **model**. For details about model package specifications, see **Introduction to Model Package Specifications**.
- File size specifications

  When a public resource pool is used, the total size of the downloaded SWR image (not the compressed image displayed on the SWR page) and the OBS model package cannot exceed 30 GB.

HTTPS Example

Use Flask to start HTTPS. The following is an example of the web server code:

```
from flask import Flask, request
import json

app = Flask(__name__)

@app.route('/greet', methods=['POST'])
def say_hello_func():
    print("----------- in hello func ----------")
    data = json.loads(request.get_data(as_text=True))
    print(data)
    username = data['name']
    rsp_msg = 'Hello, {}!'.format(username)
    return json.dumps({"response":rsp_msg}, indent=4)

@app.route('/goodbye', methods=['GET'])
def say_goodbye_func():
    print("----------- in goodbye func ----------")
    return '\nGoodbye!\n'


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

@app.route('/health', methods=['GET'])
def healthy():
    return "{\"status\": \"OK\"}"

# host must be "0.0.0.0", port must be 8080
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, ssl_context='adhoc')
```

Debugging on a Local Computer

Perform the following operations on a local computer with Docker installed to check whether a custom engine complies with specifications:

1. Download the custom image, for example, **custom_engine:v1** to the local computer.
2. Copy the model package folder **model** to the local computer.
3. Run the following command in the same directory as the model package folder to start the service:
   ```
   docker run --user 1000:100 -p 8080:8080 -v model:/home/mind/model    custom_engine:v1
   ```

   NOTE

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

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


Deployment Example

The following section describes how to use a custom engine to create an AI application.

1. Create an AI application and viewing its details.

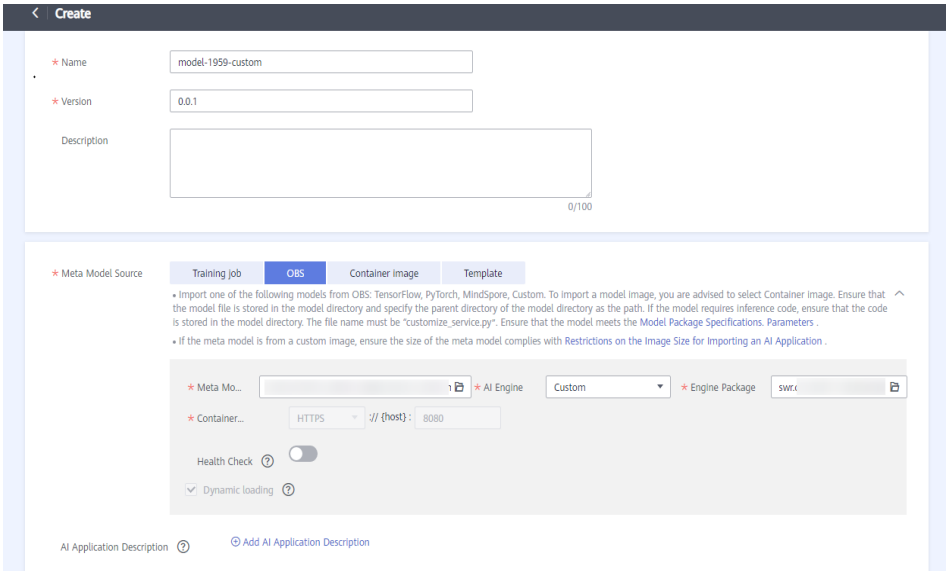
   Log in to the ModelArts console, choose **AI Application Management** > **AI Applications**, and click **Create**. On the page that is displayed, configure the following parameters:

   - **Meta Model Source**: **OBS**
   - **Meta Model**: a model package selected from OBS
   - **AI Engine**: **Custom**
   - **Engine Package**: an SWR image

     Retain the default settings for other parameters.

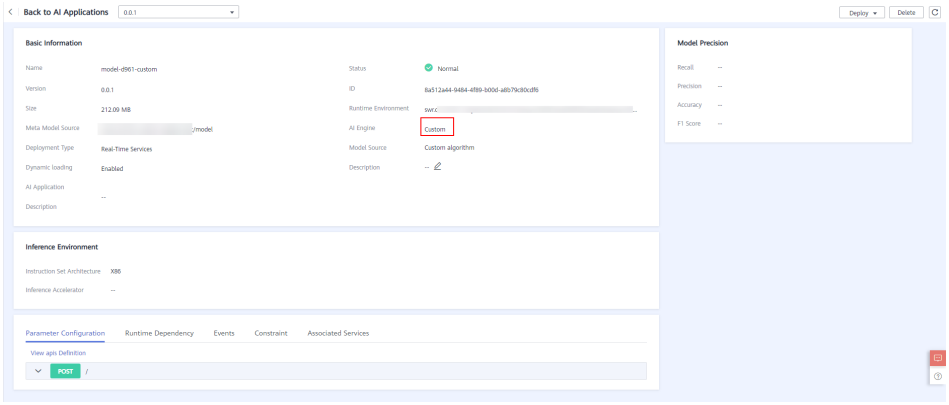
   Click **Create Now**. In the AI application list that is displayed, check the AI application status. When its status changes to **Normal**, the AI application has been created.

   Figure 5-16 Creating an AI application



   Click the AI application name. On the page that is displayed, view details about the AI application.

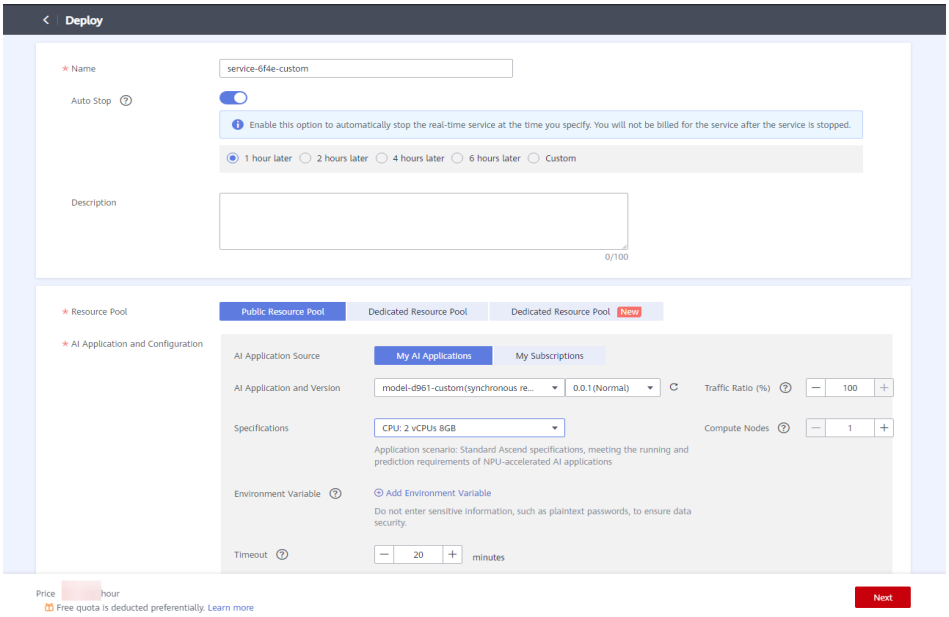
Figure 5-17 Viewing details about an AI application



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

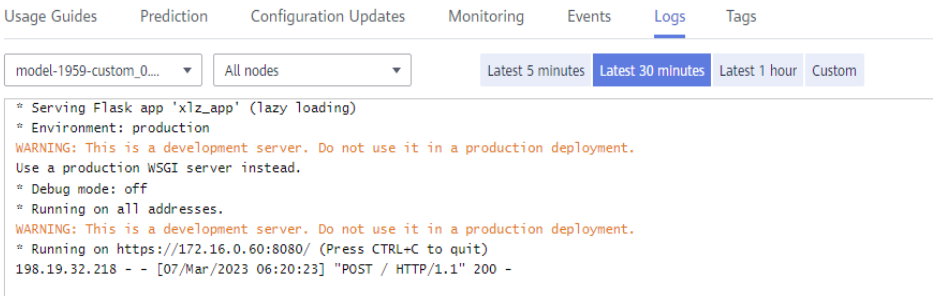
   On the AI application details page, choose **Deploy** > **Real-Time Services** in the upper right corner. On the **Deploy** page, select a proper compute node specification, retain the default settings for other parameters, and click **Next**. When the service status changes to **Running**, the service has been deployed.

Figure 5-18 Deploying a service



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

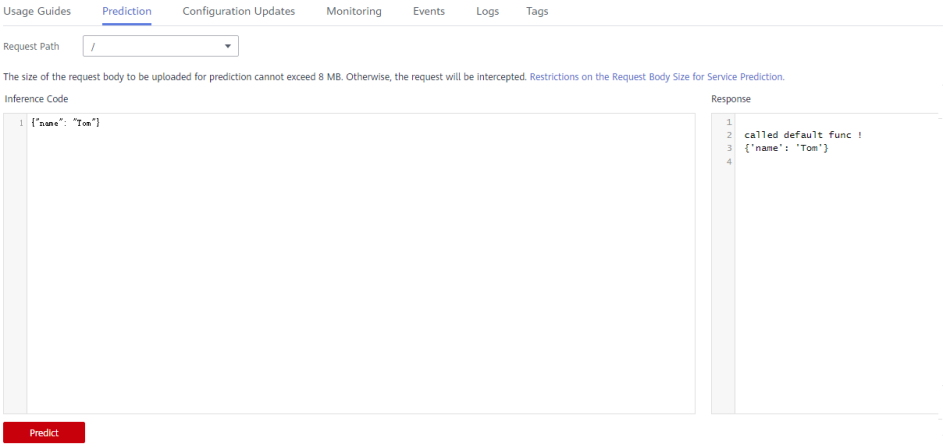
Figure 5-19 Logs



3. Use the service for prediction.

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

Figure 5-20 Prediction



5.5 Using a Large Model to Create an AI Application and Deploying 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 AI applications requires dynamic adjustment of the tenant storage quota. Slow model loading and startup requires 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 AI application management and service deployment in large model application scenarios.

Constraints

- You need to apply for the size quota of an AI application 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 an AI Application 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 an AI Application**
5. **Deploying a Real-Time Service**

Applying for Increasing the Size Quota of an AI Application 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:

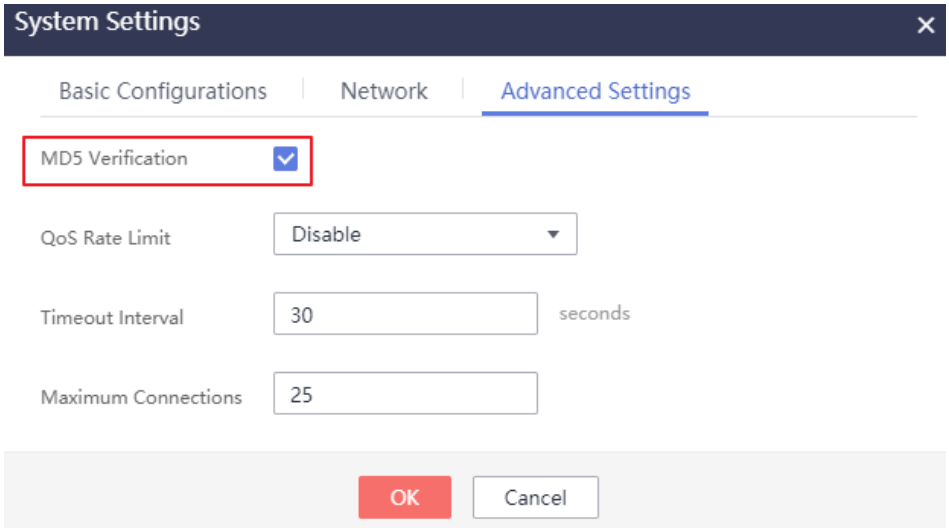
- If the model size exceeds the default quota, submit a service ticket to increase the size quota of a single AI application. The default size quota of an AI application is 20 GB.
- Submit a service ticket to add the whitelist cached using the local storage of the node.

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 5-21**. When dynamic loading is enabled and the local persistent storage of the node is used, OBS Browser+ checks data consistency during data upload.

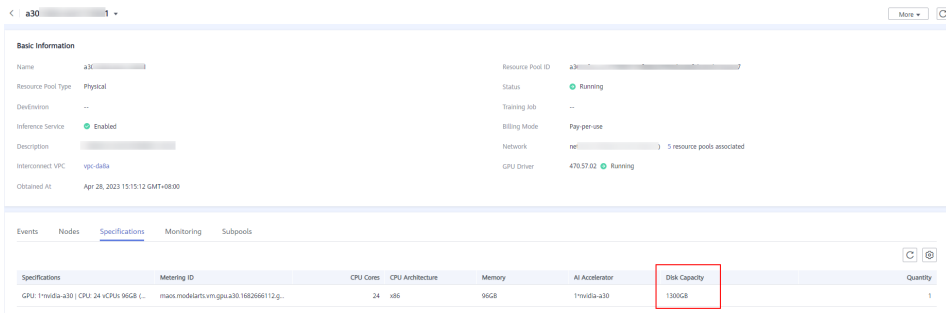
Figure 5-21 Configuring MD5 verification for OBS Browser+



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 5-22 Viewing the disk information of the dedicated resource pool



Creating an AI Application

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

1. Use a custom engine and enable dynamic loading.

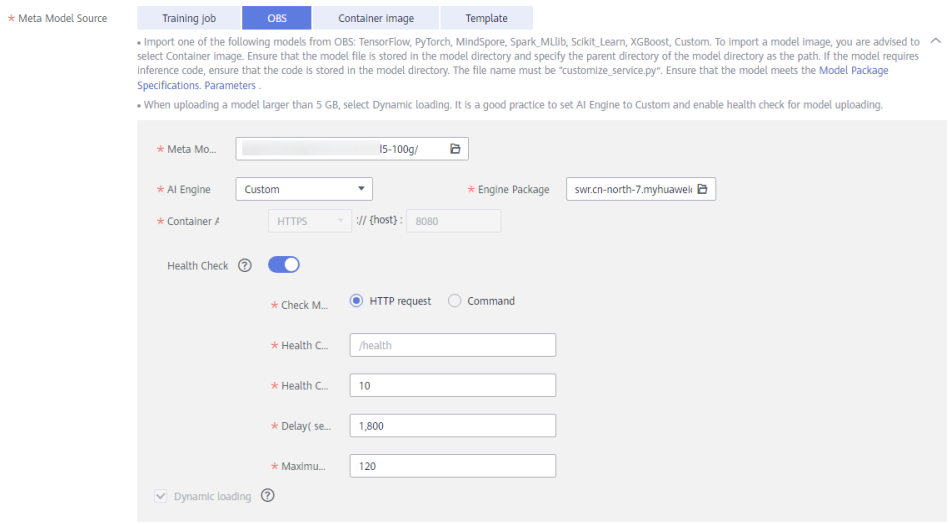
   To use a large model, you need to use a custom engine and enable dynamic loading when importing the model. You can create a custom engine to meet special requirements for image dependency packages and inference frameworks in large model scenarios. For details about how to create a custom engine, see **Creating an AI Application 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 AI applications imported using a large model to identify unavailable services that are displayed as started.

   Figure 5-23 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.

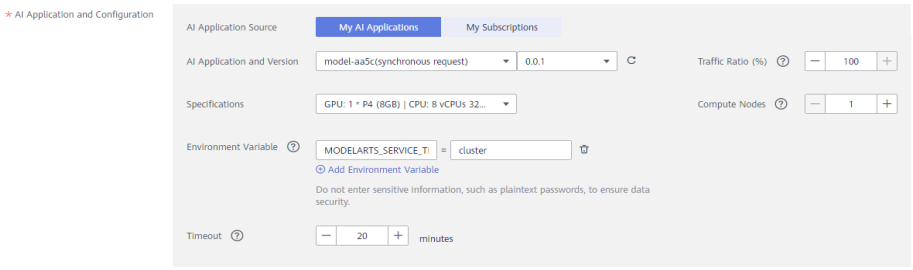
   Generally, the time for loading and starting a large model is longer than that for a common model. Set **Timeout** to a proper value. Otherwise, the timeout may elapse prior to the completion of the model startup, and the deployment may fail.

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 5-24 Customizing the deployment timeout interval and adding an environment variable



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

5.6 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 developed by NVIDIA. 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 an AI application version and deploy and manage services using the AI application.

The following figure shows the reconstruction items.

Figure 5-25 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 && \
```



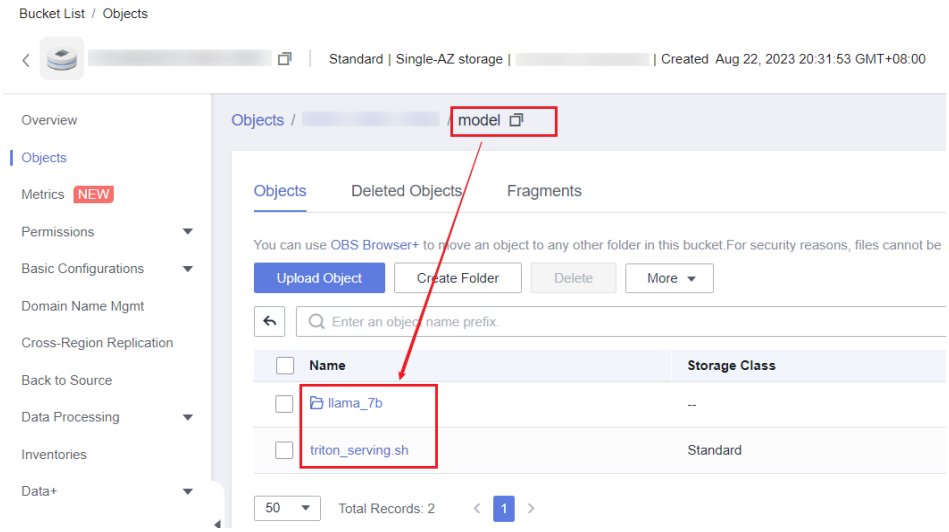
```
    rm -rf /etc/nginx/conf.d/default.conf && \
    chown -R ma-user:100 /etc/nginx/ && \
    chown -R ma-user:100 /var/log/nginx && \
    chown -R ma-user:100 /var/lib/nginx && \
    sed -i "s#/var/run/nginx.pid#/home/ma-user/nginx.pid#g" /etc/init.d/nginx

# 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.

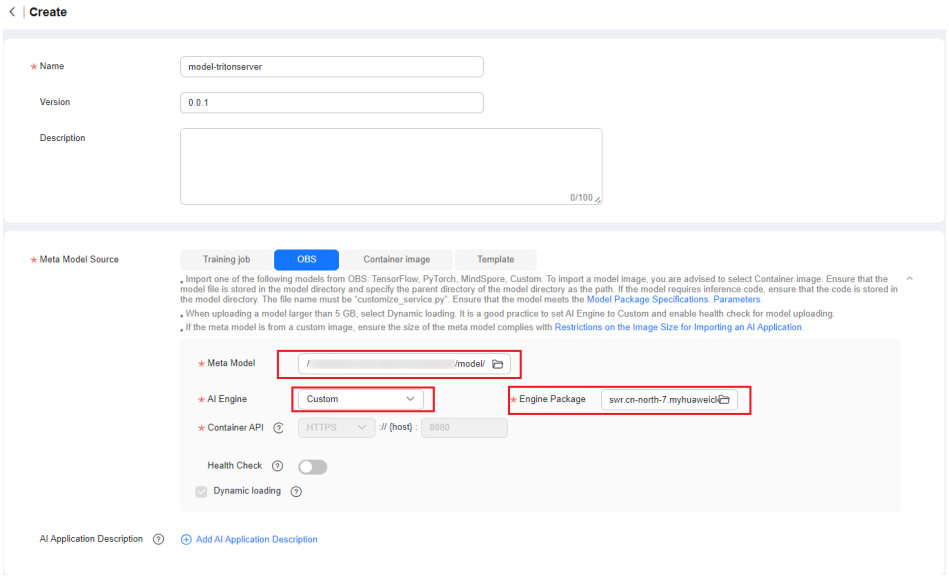
1. Create a **model** directory in OBS and upload the **triton_serving.sh** file and **llama_7b** folder to the **model** directory.

Figure 5-26 Uploading files to the **model** directory



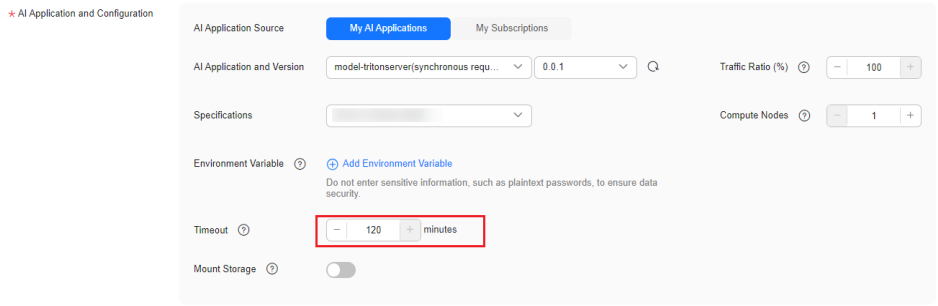
2. Create an AI application. Set **Meta Model Source** to **OBS** and select the meta model from the **model** directory. Set **AI Engine** to **Custom**. Set **Engine Package** to the image created in **Step 3**.

Figure 5-27 Creating an AI application



3. Deploy the created AI application 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 5-28 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]
    },
    {
```

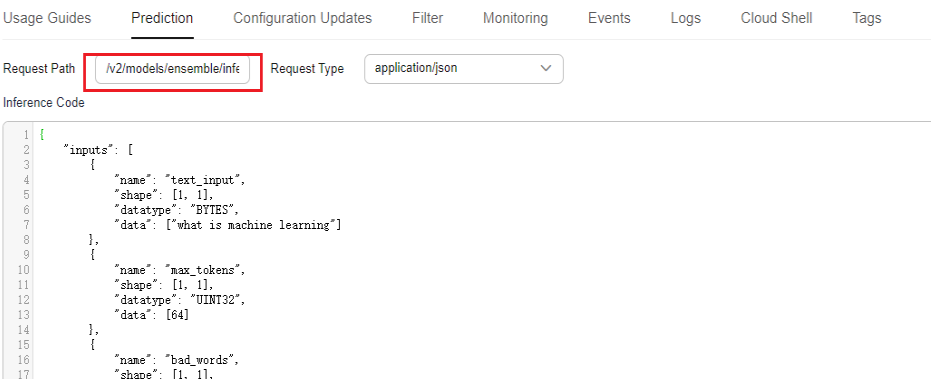


```
            "name": "bad_words",
            "shape": [1, 1],
            "datatype": "BYTES",
            "data": [""]
        },
        {
            "name": "stop_words",
            "shape": [1, 1],
            "datatype": "BYTES",
            "data": [""]
        },
        {
            "name": "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 5-29 Calling a real-time service



----End

5.7 High-Speed Access to Inference Services 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.

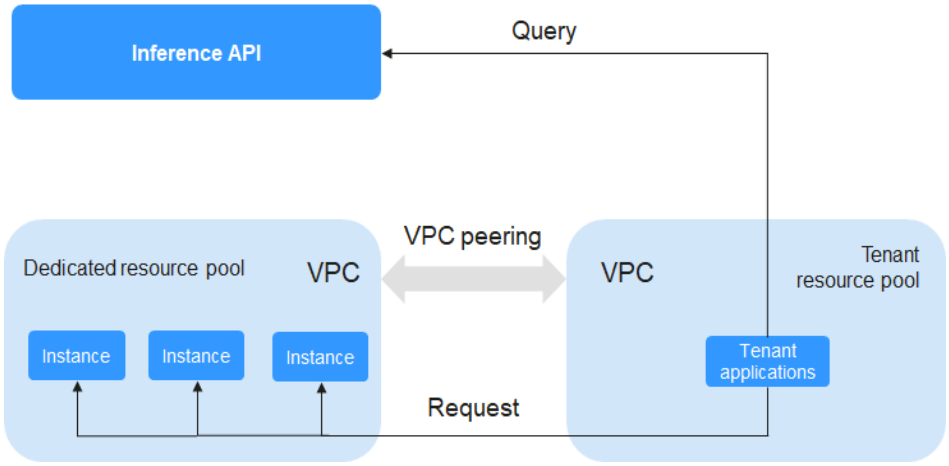
In high-speed access through VPC peering, your service requests are directly sent to instances through VPC peering but not through the inference platform. This accelerates service access.

NOTE

The following functions 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 5-30 High-speed access through VPC peering



Preparations

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

NOTICE

- For details about how to deploy services in new-version dedicated resource pools, see **Comprehensive Upgrades to ModelArts Resource Pool Management Functions**.
- 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 management console, choose **Dedicated Resource Pools** > **Elastic Cluster**, locate the dedicated resource pool used for service deployment, 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 5-31 Locating the target dedicated resource pool

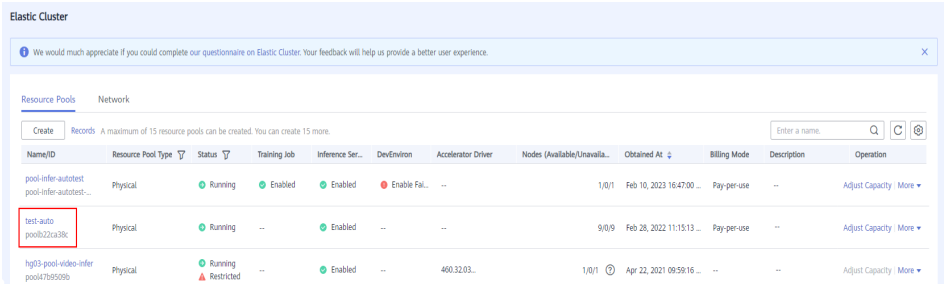


Figure 5-32 Obtaining the network configuration

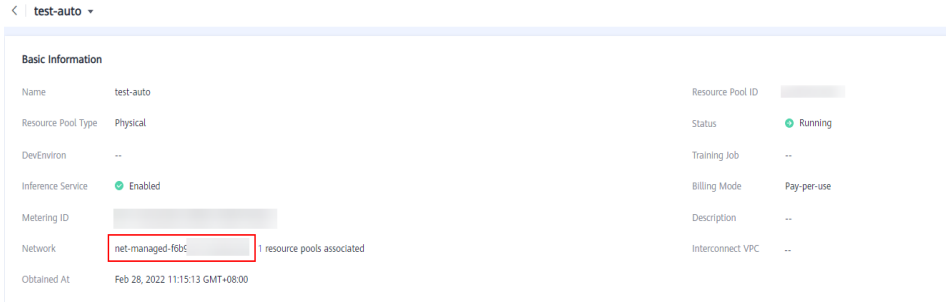
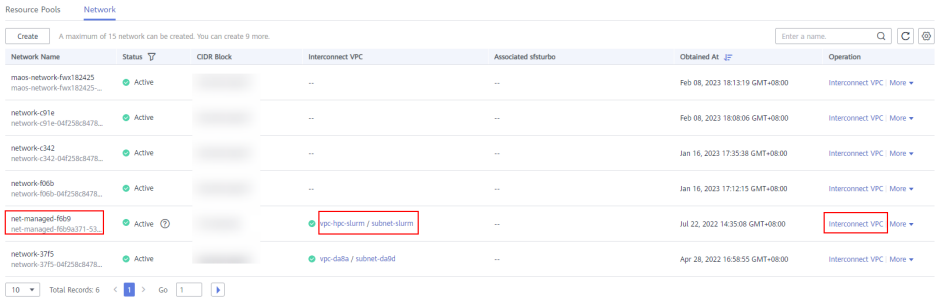


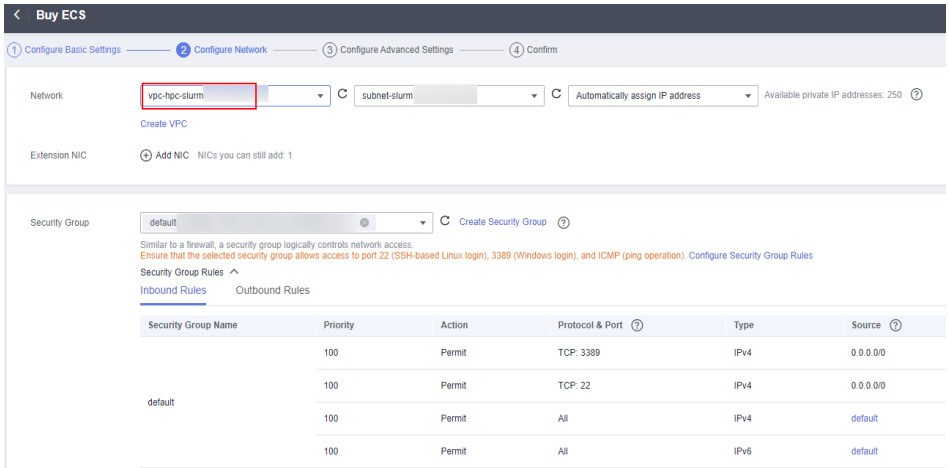
Figure 5-33 Interconnecting the VPC

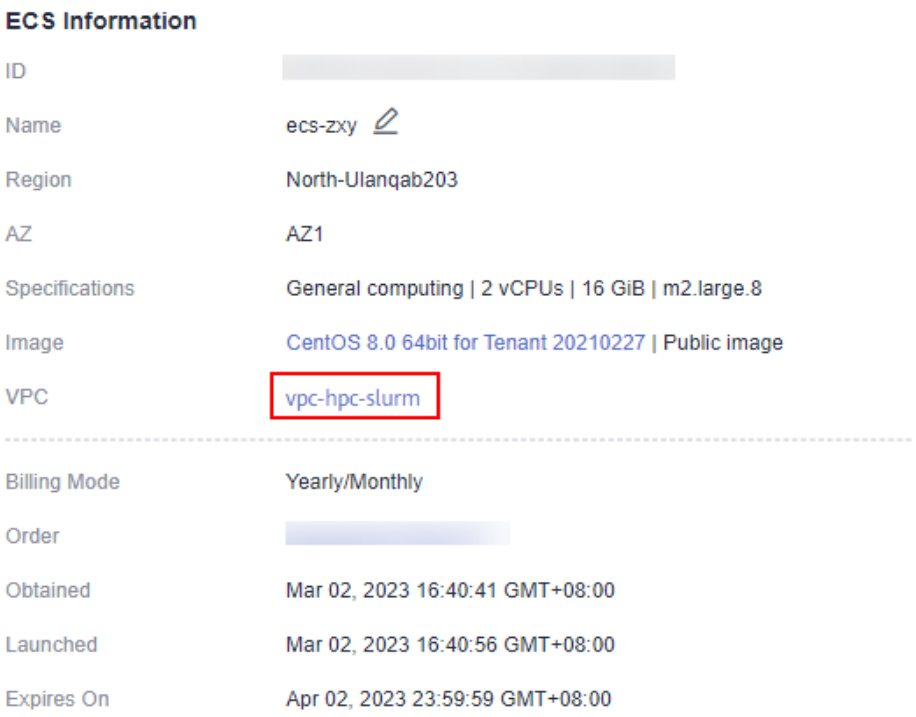


Step 2 Create an ECS in the VPC.

Log in to the ECS management console and click **Buy ECS** in the upper right corner. On the **Buy ECS** page, configure basic settings and click **Next: Configure Network**. On the **Configure Network** page, select the VPC connected in **Step 1**, configure other parameters, confirm the settings, and click **Submit**. When the ECS status changes to **Running**, the ECS has been created. Click its name/ID to go to the server details page and view the VPC configuration.

Figure 5-34 Selecting a VPC when purchasing an ECS





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

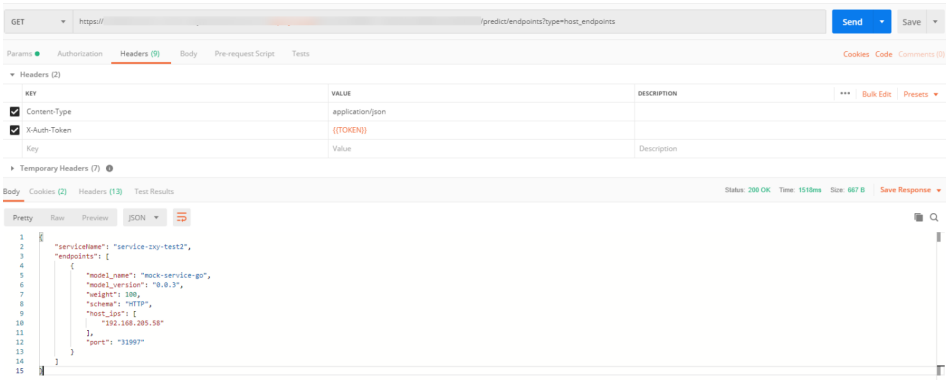
GUI software, for example, Postman can be used to obtain the IP address and port number. Alternatively, log in to the ECS, create a Python environment, and execute code to obtain the service IP address and port number.

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 5-35 Example response

- Method 2: Obtain the IP address and port number using Python.

  The following parameters in the Python code below need to be modified:

  – **project_id**: your project ID. To obtain it, see **Obtaining a Project ID and Name**.

  – **service_id**: service ID, which can be viewed on the service details page.

– **REGION_ENDPOINT**: service endpoint. To obtain it, see **Endpoint**.

```
def get_app_info(project_id, service_id):
    list_host_endpoints_url = "{}/v1/{}/services/{}/predict/endpoints?type=host_endpoints"
    url = list_host_endpoints_url.format(REGION_ENDPOINT, project_id, service_id)
    headers = {'X-Auth-Token': X_Auth_Token}
    response = requests.get(url, headers=headers)
    print(response.content)
```

Step 4 Access the service through the IP address and port number.

Log in to the ECS and access the real-time service either by running Linux commands or by creating a Python environment and executing Python code. Obtain the values of **schema**, **ip**, and **port** from **Step 3**.

- Run the following command to access the real-time service:

```
curl --location --request POST 'http://192.168.205.58:31997' \
--header 'Content-Type:  application/json' \
--data-raw '{"a":"a"}'
```

Figure 5-36 Accessing a real-time service

```
[root@ecs-zxy ~]# curl --location --request POST 'http://192.168.205.58:31997' \
> --header 'Content-Type: application/json' \
> --data-raw '{"a":"a"}'
call Post()[root@ecs-zxy ~]#
```

- Create a Python environment and execute Python code to access the real-time service.

```
def vpc_infer(schema, ip, port, body):
    infer_url = "{}://{}:{}"
    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

5.8 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.
- WebSocket supports only real-time services deployed using AI applications 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 an AI Application Using the Image**
- **Deploying the AI Application as a Real-Time Service**
- **Calling the WebSocket Real-Time Service**

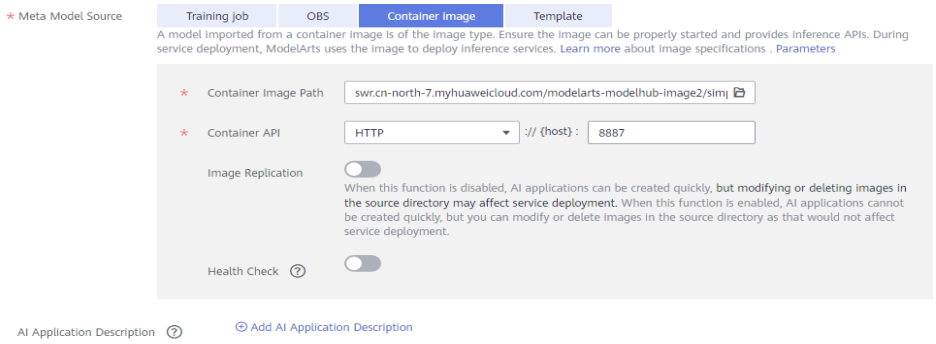
Uploading the Image to SWR

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

Creating an AI Application Using the Image

1. Log in to the ModelArts management console, choose **AI Application Management** > **AI Applications**, and click **Create** under **My AI Applications**. The page for creating an AI application is displayed.
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 5-37 AI application parameters

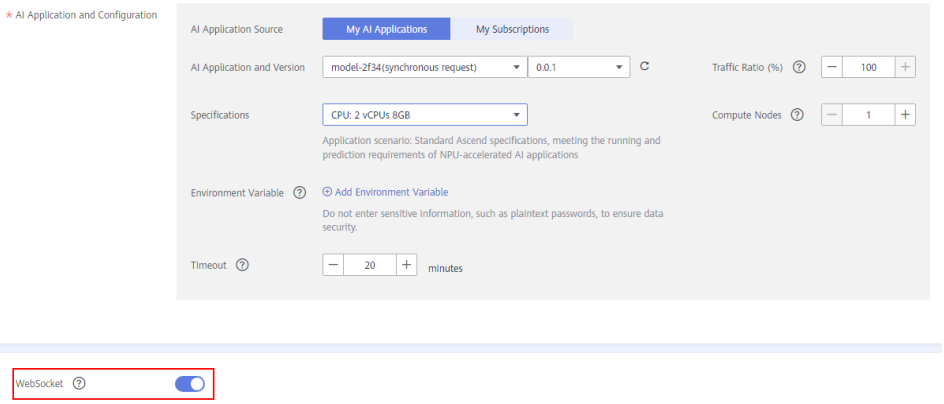


3. Click **Create now**. In the AI application list that is displayed, check the AI application status. When it changes to **Normal**, the AI application has been created.

Deploying the AI Application as a Real-Time Service

1. Log in to the ModelArts management console, choose **Service Deployment** > **Real-Time Services**, and click **Deploy**.
2. Configure the service.
   - **AI Application and Version**: Select the AI application and version created in **Creating an AI Application Using the Image**.
   - **WebSocket**: Enable this function.

Figure 5-38 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:

- **Access Authenticated Using a Token**
- **Access Authenticated Using an AK/SK**
- **Access Authenticated Using an Application**

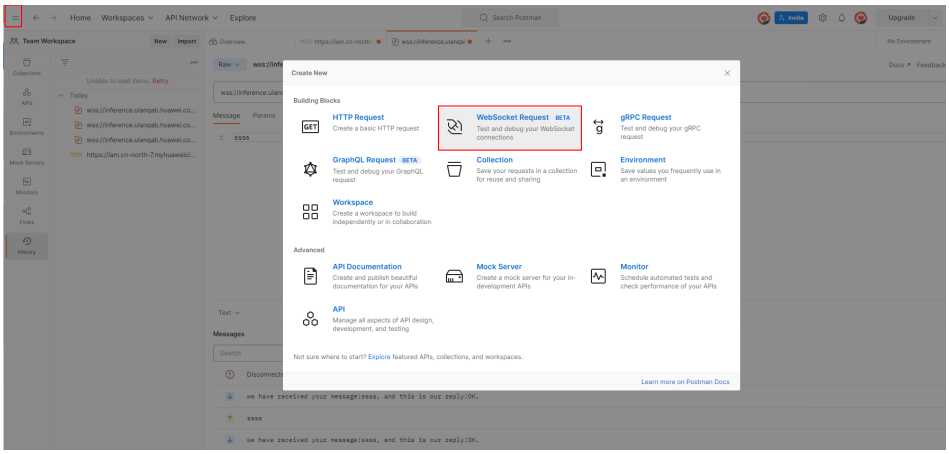
The following section uses GUI software Postman for prediction and token authentication as an example to describe how to call WebSocket.

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).

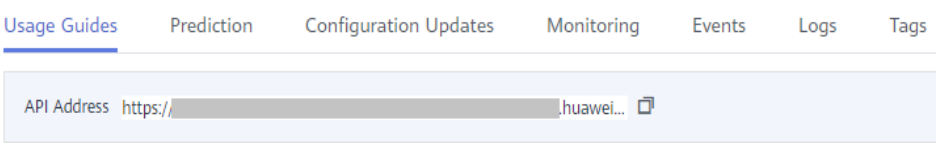
Figure 5-39 WebSocket Request



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 5-40 Obtaining the API address

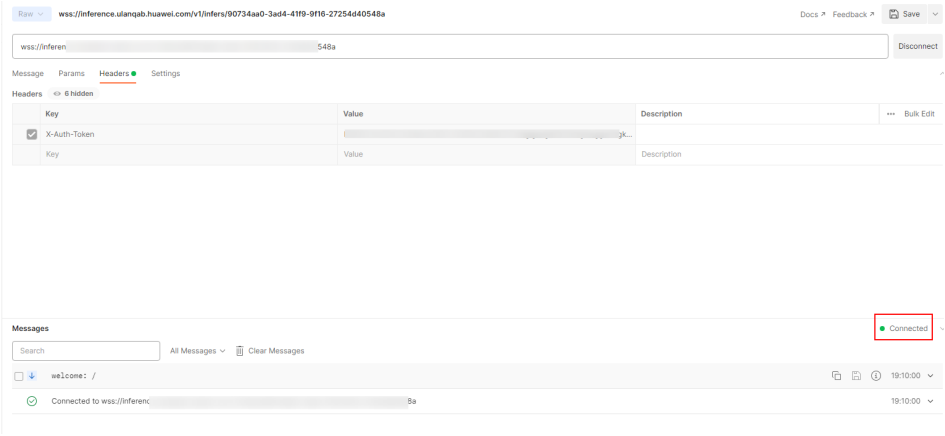


NOTE

- 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 5-41 Connection established



NOTICE

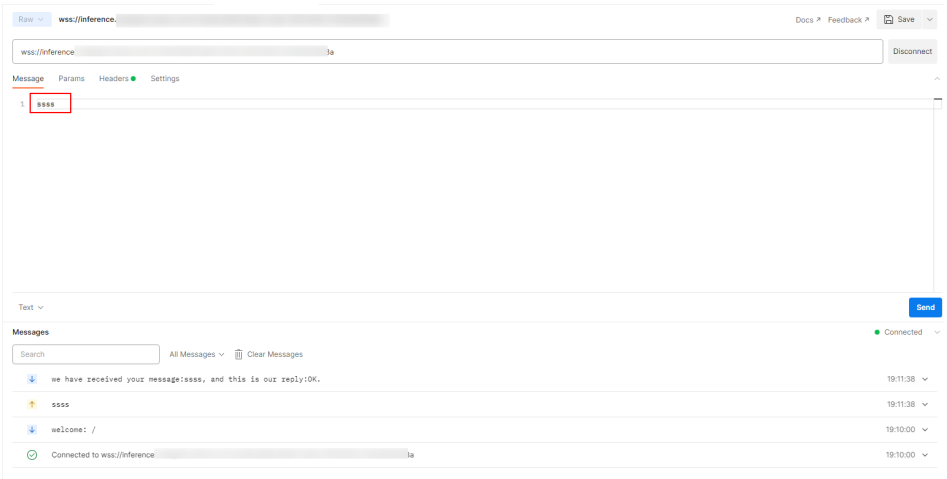
Preferentially check the WebSocket service provided by the custom image. The type of implementing WebSocket varies depending on the tool you used. Possible issues are as follows: A WebSocket connection can be established but cannot be maintained, or the connection is interrupted after one request and 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 5-42 Sending data



----End

