ModelArts

Model Development

Issue	01
Date	2026-06-04

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Using Notebook for AI Development and Debugging

1.1 Notebook User Guide

In traditional software development, reducing costs and enhancing the developer experience are paramount. However, the AI development lifecycle introduces unique challenges, such as complex toolchains and intricate resource management, creating a stark contrast to traditional workflows. To address these hurdles and boost efficiency, ModelArts offers a flexible, open development environment. By leveraging cloud-native resources and unifying the development toolchain, ModelArts delivers a superior on-cloud experience for AI exploration, model training, and education.

WARNING

Notebook instances are designed for coding and debugging. They do not support long-term stable operation and must not be used for production-grade workloads.

Using a Notebook Instance

Create a Notebook Instance

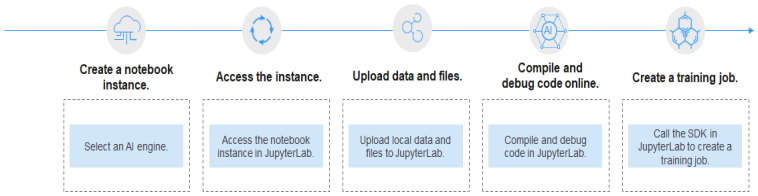
ModelArts provides on-cloud notebook, which is out of the box, relieving you from concerning environment installation or configuration. For details, see **Creating a Notebook Instance (New Page)**.

Open a Notebook Instance

Notebook supports the following methods to develop AI models based on engines such as PyTorch, TensorFlow, and MindSpore:

- JupyterLab: You can access and use notebook instances online via the integrated JupyterLab. For details, see **Using a Notebook Instance for AI Development Through JupyterLab**.

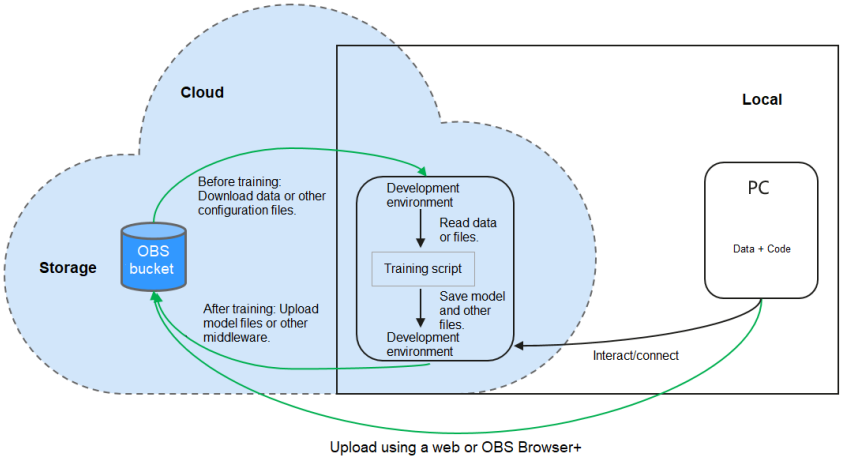
Figure 1-1 Using JupyterLab to develop and debug code online



- Local IDE: ModelArts allows you to develop models using your preferred local IDE. By enabling SSH connection, you can remotely connect your local environment to a ModelArts notebook development environment to debug and run code. This approach maintains your existing coding habits while leveraging the on-cloud notebook development environment.

  A local IDE supports VS Code and SSH. Additionally, VS Code Toolkit is provided for convenient remote access. For details, see **Using Notebook Instances Remotely Through VS Code** and **Using a Notebook Instance Remotely with SSH** .

  Figure 1-2 Remotely accessing a notebook instance from a local IDE



Manage Notebooks

When using notebooks, you can quickly locate instances and switch images within the same instance to flexibly adjust AI engines. You can also modify node specifications to scale resources as needed. For storage, you can start with a smaller capacity and expand the EVS volume later based on your requirements. By using dynamic OBS mounting, you can simulate OBS storage as a local file system. Additionally, if a notebook instance becomes abnormal, you can view instance events for fault locating.

Table 1-1 Notebook operations

Operation	Description	Helpful Links
Searching for a notebook instance	All created instances are displayed on the notebook page. To find a specific instance, use quick search with filter criteria.	**Searching for a Notebook Instance**
Accessing an environment	Access a notebook instance in the **Running** state for coding.	**Using a Notebook Instance**
Saving an image	Create a notebook instance using a preset image, install custom software and dependencies on the base image, and save the running environment instance as a container image.	**Saving a Notebook Instance**
Stopping, starting, or deleting a notebook instance	Stop the notebook instances that are not needed. You can also restart a stopped instance. Delete the notebook instances that are not needed. Deleted notebook instances cannot be recovered. After a notebook instance is deleted, the data stored in the mounted directory will be deleted.	**Starting, Stopping, or Deleting a Notebook Instance**
Configuring auto stop	Once enabled, you can set the auto-stop mode and duration. The instance will automatically stop if its usage surpasses the configured limit, though there might be a 2–5 minute delay before stopping. Billing continues until the instance is stopped.	**Starting, Stopping, or Deleting a Notebook Instance**
Changing an image	ModelArts allows you to change the image of a notebook instance to adjust its AI engine flexibly. The image of a notebook instance can be changed only when the instance is in the **Stopped** state.	**Changing an Image**

Operation	Description	Helpful Links
Changing instance specifications	ModelArts allows you to change the node specifications of a notebook instance, enabling you to flexibly adjust resource specifications. Specifications can be changed only for notebook instances in the **Stopped**, **Running**, or **Start Failed** state.	**Changing the Specifications of a Notebook Instance**
Node affinity scheduling	Node affinity ensures that pods are scheduled to nodes that meet specific conditions, enabling more fine-grained resource management and optimization. You can enable, modify, or disable node affinity scheduling only for notebook instances that are stopped or failed to be started and whose resource pool type is dedicated resource pool.	**Setting Node Affinity Scheduling**
Remote SSH	ModelArts allows you to modify the SSH configuration for notebook instances. The configuration can be modified only when the notebook instance is in the **Stopped** state. Remote SSH cannot be disabled once it is enabled.	**Configuring Remote SSH Connection for a Notebook Instance**
Editing tags	Tags can be used to identify, classify, search for, and manage notebook instances.	**Editing a Tag**
Dynamically expanding EVS disk capacity	During notebook development, select a small EVS disk capacity, for example, 5 GB, when creating a notebook instance because the storage requirements are low at the initial stage. After the development, a large volume of data must be trained. Then, expand the disk capacity to cost-effectively meet your service needs.	**Dynamically Expanding EVS Disk Capacity**

Operation	Description	Helpful Links
Dynamically mounting an OBS parallel file system	In ModelArts running notebook containers, the dynamic mounting feature is used to simulate OBS as a local file system.	**Dynamically Mounting an OBS Parallel File System**
Viewing notebook instance events	Instance statuses and key operations such as creating, starting, and stopping an instance, and changing the instance flavor are recorded in the backend. You can view the events on the notebook instance details page to monitor the instance statuses.	**Viewing Notebook Events**
Reporting notebook cache disk alarms	When creating a notebook instance, you can select CPU, GPU, or Ascend resources based on the service data volume. If you select GPU or Ascend resources, ModelArts mounts hard disks to the cache directory. You can use this directory to store temporary files. Capacity alarms are not generated for the cache directory of the notebook instance by default. Exceeding the capacity limit will restart the notebook instance. After the restart, multiple configurations are reset, discarding your data and losing the environment. This will affect your experience. You are advised to enable the monitoring and alarms for the cache directory usage and report the data to AOM.	**Notebook Cache Directory Alarm Reporting**

Operation	Description	Helpful Links
Upgrading JupyterLab	To enhance the usability of notebook, JupyterLab has been upgraded to version 4.4.10 (stable version in 2025). This major leap from the 2021 legacy version (3.2.3) significantly optimizes performance and startup speed. Key improvements include a rebuilt editor, enhanced debugging and collaboration tools, a more stable ecosystem, and comprehensive refinements to the interactive experience and memory efficiency.	**Upgrading JupyterLab**

Upload a File to a Notebook Instance

Easy and fast file uploading is a common requirement in AI development. ModelArts allows you to upload files in multiple ways. You can view the upload progress and speed during the uploading.

- **Upload local files**.
- **Upload GitHub open-source repository files**.
- **Upload OBS files**.
- **Upload remote files such as open-source datasets**.

ModelArts CLI

The ModelArts CLI is integrated into ModelArts Notebook to connect with ModelArts services and execute management commands on ModelArts resources. ma-cli allows you to interact with cloud services from both ModelArts notebook instances and offline VMs. By using ma-cli commands, you can enable command auto-completion and authentication, build images, submit ModelArts training jobs, submit DLI Spark jobs, and perform OBS data replication. For details, see **ModelArts CLI Command Reference**.

MoXing

ModelArts Notebook has a built-in MoXing Framework module. ModelArts **mox.file** provides a set of APIs for accessing OBS more conveniently, allowing you to operate OBS files by simulating a series of APIs for operating the local file system. For details, see **Using MoXing Commands in a Notebook Instance**.

Helpful Links

- **Billing Item (ModelArts Standard) > Development Environment**
- **Best Practices > Notebook**

- **Troubleshooting > DevEnviron**
- **FAQs > ModelArts Standard Notebook**

1.2 Creating a Notebook Instance (New Page)

NOTE

ModelArts has enhanced the creation page to improve the efficiency of creating notebook instances. The updated page streamlines operations and enhances the GUI display.

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

You can create a notebook instance in either of the following ways:

- Create a notebook instance on the **ModelArts console**. This section provides the operation guide of the new UI. For details about the operation guide of the old-version UI, see **Creating a Notebook Instance (Old Page)**.
- Create a notebook instance via ModelArts APIs. For details, see **Creating a Notebook Instance**.

Constraints

- When a notebook instance is created, auto stop is enabled by default. The notebook instance will automatically stop at the specified time.
- Only running notebook instances can be accessed or stopped.
- By default, each IAM user can create a maximum of 10 notebook instances.
- For single-PU instances powered by Snt9b2*x* (like Snt9b23) or D310P-300 resource pools, EVS disks cannot be used to create notebook instances (when **Storage** is set to **EVS**).
- Notebook does not support open ports for external services.

Precautions

Being a commissioning environment, notebooks allows you to download files via the public network using a public proxy. Therefore, do not download files larger than 10 GB from a notebook instance. Notebook provides limited public network access bandwidth, which ensures only network connectivity but not download speed.

Billing

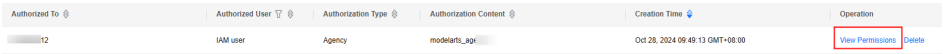
Notebook instance is billed as follows:

- A running notebook instance will be billed based on used resources. The fees vary depending on your selected resources. For details, see **Pricing Details**. When a notebook instance is not used, stop it.
- If you select EVS for storage when creating a notebook instance, the EVS disk will be continuously billed if the instance is not deleted. Stop and delete the notebook instance if it is not required. For details, see **Development Environment**.

Creating a Notebook Instance

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Permission Management** and check whether the access authorization has been configured. If not, configure access authorization. For details, see **Configuring Agency Authorization for ModelArts with One Click**.

   Figure 1-3 Viewing agency configurations

2. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create Notebook** in the upper right corner. On the displayed page, configure the parameters by referring to the following table.

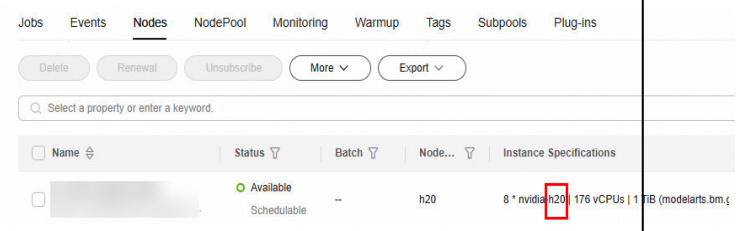
   Table 1-2 Parameters for creating a notebook instance

Parameter		Description
Basic Information	Name	Name of the notebook instance, which is automatically generated by the system. You can rename it as required. The name can contain at most 128 characters and cannot be left empty. Only digits, letters, underscores (_), and hyphens (-) are allowed.
	Add Description	Click **Add Description** to customize the notebook instance description, which cannot exceed 512 characters.
	Tags	To add the same tag to different cloud resources, use Tag Management Service (TMS) to create predefined tags. For details, see **Creating Predefined Tags**. Click **Add Tag** and configure the tag key and tag value as required. You can add at most 20 tags. After adding a tag, you can view, modify, or delete the tag on the **Notebook** page or the notebook instance details page. For details, see **Editing a Tag**. **NOTE** You can select a predefined TMS tag from the tag drop-down list or customize a tag. Predefined tags are available to all service resources that support tags. Customized tags are available only to the service resources of the user who has created the tags.

Parameter		Description
Auto Stop		This function is enabled by default. The notebook instance tracks its runtime once started. It stops automatically if the runtime surpasses the set time limit. **Stop Type**: Select **Stop as scheduled**. In this case, the notebook instance automatically stops when the runtime exceeds the specified time. You can select **1 hour**, **2 hours**, **4 hours**, **6 hours**, or **Custom**. You can select **Custom** to specify any integer from 1 to 72 hours. **CAUTION** To protect in-progress jobs, a notebook instance does not automatically stop immediately at the auto stop time. Instead, there is a 2 to 5 minutes delay for you to renew the auto stop time.
Environment	Select Image	Select **Preset Images** or **Custom images** as required and click  . On the displayed **Image** page, select the target image, and click **OK**. • Preset images are the AI engines built in ModelArts. • You can use a custom image created by yourself. Use either of the following ways to create a custom image: – Save the instance created using the preset image and use it as a custom image. For details, see **Saving a Notebook Instance**. – Create a custom image using a base, enterprise, or third-party image. To create a custom image, you must comply with the image specifications. After the build is complete, you must register the image on the **Image Management** page of ModelArts so that the image can be used in a notebook instance. For details, see**Creating a Custom Image**. An image corresponds to an AI engine. When you select an image during instance creation, the AI engine is specified accordingly. Select an image as required. Enter a keyword of the image name in the search box on the right to quickly search for the image. You can change an image on a stopped notebook instance. For details, see **Updating a Notebook Instance**.

Parameter		Description
	JupyterLab Version	ModelArts notebook supports the following two versions of JupyterLab. The default version is JupyterLab 4. • JupyterLab 4: This version has significantly improved user experience, functions, and performance. For details, see **Upgrading JupyterLab**. • JupyterLab 3: Support for creating new notebook instances using JupyterLab 3 will be discontinued in April 2026. Additionally, technical support, including updates for new features, vulnerability/issue fixes, patch upgrades, service ticket guidance, and online troubleshooting, will no longer be provided. These services will no longer be applicable to the O&M assurance of ModelArts. You are advised to use JupyterLab 4.
Resource	Resource Pool Type	Public and dedicated resource pools are available. A dedicated resource pool allows for mixed deployments using CPU, NPU, and GPU resources. If your node type supports both GPU and CPU, you can choose either GPU or CPU instance specifications. • **Public resource pool**: It is billed based on the runtime of your notebook instances. • **Dedicated resource pool**: The resources provided in a dedicated resource pool are exclusive and more controllable. In the **Resource Pool** area, click **Select Resource Pool**. On the **Select Dedicated Resource Pool** page, select a dedicated resource pool as required and click **OK**. If you do not have a dedicated resource pool, click **Buy Dedicated Resource Pool** in the lower part of the **Select Dedicated Resource Pool** page to purchase a dedicated resource pool. For details about the parameters for purchasing a dedicated resource pool, see **Creating a Standard Dedicated Resource Pool**. **NOTE** If the dedicated resource pool you purchased is a single-node Tnt004 pool whose specification is **GPU: 1*tnt004 | CPU: 8 vCPUs 32GiB** (**modelarts.vm.gpu._tnt004u8**), when you use the cluster to create a notebook instance, the Tnt004 PU is idle but is displayed as sold out or the creation fails due to insufficient resources, contact technical support.
	Instance Specifications	The system will select an instance specification by default. You can click  to change the specification. In the displayed dialog box, select a CPU, NPU, or GPU specification as required and click **OK**.

Parameter		Description
	Specification Type	Preset and custom specifications are supported. Custom specifications are only supported when the **Resource Pool Type** is set to **Dedicated resource pool** and the node pool flavor is GPU, CPU, or a heterogeneous resource pool (e.g., CPU+CPU or GPU+CPU). They are not supported for public resource pools and NPU-based dedicated resource pools. **NOTE** If there are NPU nodes in a heterogeneous resource pool, only the NPU nodes do not support custom specifications. • Preset specifications: Select the preset ModelArts instance specifications from the drop-down list. • Custom specifications: Customize the GPU/CPU and memory specifications based on the nodes in the resource pool. To prevent scheduling failures when using custom specifications, **reserve an additional 0.5 vCPUs | 1 GiB of notebook system resources** in addition to the configured specifications. For example, a notebook instance with custom specifications of 0.5 vCPUs | 512 MiB requires 1 vCPU | 1.5 GiB of available node resources. To view the node details of the resource pool, go to the **ModelArts console**, choose **Resource Management** > **Dedicated Compute Resources** > **Resource Pools** or **Resource Management** > **Dedicated Resource Pool**. The values of custom specifications are as follows: – GPUs To use GPU virtualization, ensure the requirements are met. For details, see **Using GPU Virtualization**. – H20 nodes To view the node type, log in to the **ModelArts console**, choose **Resource Management** > **Dedicated Compute Resources** > **Resource Pools** or **Resource Management** > **Dedicated Resource Pool**, click the name of the target resource pool, and check whether its instance specifications include **h20** in the **Nodes** tab. **h20** indicates that the node type is H20.

Parameter		Description
		Figure 1-4 H20 model  – Use AI Suite (NV GPU) plugin 2.12.0. To view or upgrade the AI Suite (NV GPU) plugin version, see **AI Suite (NV GPU)**. – Volcano Scheduler has been installed. For details about how to install Volcano Scheduler, see **Volcano Scheduler**. Value description: – When the value is less than 1, the step is 0.1 (range: [0.0, 0.9]). – When the value is 1 or greater, the step is 1 (integer values only). – vCPUs The value must be 0.4 or greater. Unit: cores; step: 0.01. – Memory (MiB) The value must be 513 or greater. Unit: MiB; step: 1.

Parameter		Description
	Node affinity scheduling	This parameter is available only when **Resource Pool Type** is set to **Dedicated resource pool**. After you set node affinity, pods can be scheduled to specific node for fine-grained resource management and optimization. ModelArts allows fine-grained control over Pod deployment strategies, including: strict placement (strong affinity), preferred placement (weak affinity), prohibited placement (strong anti-affinity), and avoided placement (weak anti-affinity). After you enable this function, set the affinity type, strength, and nodes in the displayed **Affinity-based Scheduling Policy** window, and click **OK**. ● If **Affinity Type** is set to **Node Affinity** and **Strength** is set to **Weak**, the system will try to schedule pods to the specified node, however, the scheduling may fail. ● If **Affinity Type** is set to **Node Affinity** and **Strength** is set to **Strong**, the system will ensure that the pods are scheduled to the specified node. Otherwise, the pods will not be scheduled. ● If **Affinity Type** is set to **Node Anti-Affinity** and **Strength** is set to **Weak**, the system will avoid scheduling pods to the specified node, however, this operation may fail. ● If **Affinity Type** is set to **Node Anti-Affinity** and **Strength** is set to **Strong**, the system will ensure that the pods will not be scheduled to the specified node. Otherwise, this operation will not be performed at all. You cannot select all nodes. However, you need to reserve at least one available node. Otherwise, the scheduling policy cannot be executed.

Parameter		Description
Storage Settings	Storage Type	Select **Elastic Volume Service**, **Object Storage Service – Parallel File System**, **Object Storage Service – Bucket**, **Scalable File Service**, or **OceanStor Pacific** as required. Storage configuration varies depending on the selected resource type and specifications. For details about storage types, see **Storage Types**. **NOTE** **Object Storage Service – Bucket** and **Object Storage Service – Parallel File System** are under restricted use. Submit a service ticket for a trial if required. • **Elastic Volume Service** Set a disk size based on service requirements. The default value is 5 GB. The maximum disk size is displayed on the GUI. The EVS disk space is charged by GB from the time the notebook instance is created to the time the notebook instance is deleted. • **Scalable File Service** Select this type only for a dedicated resource pool. SFS takes effect only after a dedicated resource pool can communicate with your VPC. For details, see **Step 1: Creating a Network**. **NOTE** For details about how to set permissions to access SFS Turbo folders, see **Permissions Management**. – **Scalable File Service**: Select a created SFS Turbo file system. To create an SFS Turbo file system, log in to the **SFS console**. – **Cloud Mount Path**: Retain the default value **/home/ma-user/work/**. – **Mounted Subdirectory**: Select the storage path on SFS Turbo. – **Mount Method**: This parameter is displayed when the folder control permission is granted for the user. The **read/write** or **read-only** permission is displayed based on the storage path on SFS Turbo. • Select **Object Storage Service – Bucket** or **Object Storage Service – Parallel File System** as the storage location. Click under **Storage Path**. In the displayed dialog box, select the OBS path for storing notebook data and click **OK**. If you want to use existing files or data, upload them to the specified OBS path. **Storage Path** must be set to a specific directory in an OBS bucket rather than the root directory of the OBS bucket. EVS and SFS are all mounted to the **/home/ma-user/work** directory.

Parameter		Description
		You can add a data storage path during the runtime of a notebook instance by referring to **Dynamically Mounting an OBS Parallel File System**. Data stored in the directory is retained even if the notebook instance is stopped or restarted. When a notebook instance is deleted, the EVS storage is released and the stored data is not retained. SFS can be mounted to a new notebook instance and data can be retained.

Parameter	Description
Expansion Storage	You can add different types of extended storage as required. The maximum number of extended storage configurations that can be added varies depending on the storage type. Check the console for details. You can delete unnecessary extended storage configurations. For details about storage types, see **Storage Types**. • If you select **Object Storage Service – Bucket** or **Object Storage Service – Parallel File System** as the storage type, set the following parameters: – Mounted Subdirectory: Click to select a storage address or enter a storage address, for example, **obs://bucketname/path/**. The mounted subdirectory must start with **obs://** or **/**, end with **/**, and must not contain **//** (except for the prefix). – **Cloud Mount Path**: Enter a mount path, for example, **/temp/**. The mount path cannot be empty. It cannot be a blacklisted directory and must start and end with a slash (/). It can contain only letters, digits, underscores (_), and hyphens (-). • If you select **Scalable File Service** (only supported by dedicated resource pools), set the following parameters: – **Scalable File Service**: Select a file system as required. – **Mounted Subdirectory**: The subdirectory mount path must start and end with a slash (/) and can contain only letters, digits, underscores (_), and hyphens (-). – **Cloud Mount Path**: The mount path cannot be empty. It cannot be a blacklisted directory and must start and end with a slash (/). It can contain only letters, digits, underscores (_), and hyphens (-). **NOTE** The directories must be unique and cannot be mounted to a blacklisted directory. Nested mounting is allowed. Blacklisted directories are those with the following prefixes: **/data/**, **/cache/**, **/dev/**, **/etc/**, **/bin/**, **/lib/**, **/sbin/**, **/modelarts/**, **/train-worker1-log/**, **/var/**, **/resource_info/**, **/usr/**, **/sys/**, **/run/**, **/tmp/**, **/infer/**, and **/opt/** After adding extended storage, you can view or edit the extended storage information in the **Storage** tab of the notebook instance details page. If the number of storage devices does not reach the maximum, you can click **Add Extended Storage**. For details, see **Dynamically Mounting an OBS Parallel File System**.

Parameter		Description
Authentication Information	Secret	This parameter is mandatory when the storage type is set to **Object Storage Service – Bucket** or **Object Storage Service – Parallel File System**. Select an existing secret or click **Create Credential** on the right to create one. On the displayed DEW console, **create a secret**, and enter the secret key/value and user AK/SK.
More Settings	Remote SSH	After you enable this function, you can remotely access the development environment of the notebook instance from your local development environment. When a notebook instance is stopped, you can update the SSH configuration on the instance details page. The notebook instances with remote SSH enabled have VS Code plug-ins (such as Python and Jupyter) and the VS Code server package pre-installed, which occupy about 1 GB persistent storage space.
	Key Pair	Set a key pair after remote SSH is enabled. Select an existing key pair or click **Create Key Pair** on the right to create one. On the displayed page, choose **Key Pair Service** from the navigation pane. Then, click the **Account Key Pairs** tab and click **Create Key Pair**. After a notebook instance is created, you can change the key pair on the instance details page. **CAUTION** Download the created key pair and properly keep it. When you use a local IDE to remotely access the notebook development environment, the key pair is required for authentication.

Parameter		Description
	Network	If this function is enabled, you can configure the VPC to connect the notebook instance to the network. After this function is enabled, the instance can be mounted to your VPC to implement access of multiple network planes. Before using this function, configure fine-grained VPC access authorization. For details, see **Creating an IAM User and Granting ModelArts Permissions**. If you have the **VPC Administrator** permission, you do not need to set this parameter. • **VPC**: Select an existing VPC from the drop-down list, or click **Create VPC** as required. In the displayed **Create VPC** dialog box, configure related information, click **OK**, and select the new VPC from the drop-down list. • **Subnet**: After a VPC is selected, the default subnet is displayed. You can also select an existing subnet from the drop-down list, or click **Create Subnet** and configure parameters in the displayed **Create Subnet** dialog box, click **OK**, and select the new subnet from the drop-down list. • **Security Group**: Select an existing security group or click **Create Security Group**. In the displayed dialog box, configure related information and click **OK**. Select the new security group from the drop-down list.

Parameter		Description
	Specify User	When starting a notebook instance, ModelArts supports the following two running user configurations. Supported configurations may vary depending on the resource settings. Only dedicated resource pools in the new network mode support starting notebook instances as the **root** user. For details, see **Starting a Notebook Instance as User root**. For details about how to create a dedicated resource pool in the new network mode, see **Creating a Dedicated Resource Pool**. • **ma-user/ma-group**: The default non-privileged user configuration for ModelArts public images (security mode). To use this mode, the following conditions must be met: – User: **ma-user** (UID: 1000) – User group: **ma-group** (GID: 100) **Note**: If you are using a custom image, you must pre-configure the above user and group in your image; otherwise, the container may fail to start or experience service exceptions due to insufficient permissions. For details about how to add a specific user and user group, see **Dockerfile on a Non-ModelArts Base Image**. • **root/root**: Runs the notebook instance with highest privileges. This is suitable for scenarios requiring access to system-level resources but involves potential security risks. When **root/root** is selected, the system forcibly binds the following user and group: – User: **root** (UID: 0) – User group: **root** (GID: 0) **Note**: Modifying the UID/GID or the associated group for **root** is strictly prohibited, as doing so may cause container permission conflicts or security vulnerabilities.

4. After all parameters are set, the configuration summary is displayed on the right, and the fee is displayed in the lower part. Confirm the information and click **Next**.

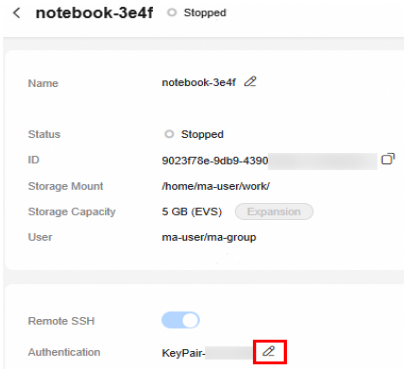
   The final bill may vary slightly. Switch to the notebook instance list. The notebook instance is being created. It will take several minutes before its status changes to **Running**. Then, the notebook instance is created.

   If an error occurs when you create or use a notebook instance, you can rectify the fault by referring to **Troubleshooting** and **FAQs**.

5. In the notebook instance list, click the instance name. On the displayed instance details page, view the instance configuration.

When the SSH remote development feature is enabled and the notebook instance is in the **Stopped** state, you can click to the right of **Authentication** to update the key pair. For details about how to enable SSH remote development, see **Configuring Remote SSH Connection for a Notebook Instance**.

Figure 1-5 Updating a key pair



In the **Storage** tab, click **Mount Storage** to mount a parallel file system to the instance for reading data. For details, see **Dynamically Mounting an OBS Parallel File System**.

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

Accessing a Notebook Instance

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

- Online access using JupyterLab. For details, see **Using a Notebook Instance for AI Development Through JupyterLab**.
- Remotely accessed from a local IDE through VS Code. For details, see **Using Notebook Instances Remotely Through VS Code**.
- Remotely accessed from a local IDE through SSH. For details, see **Using a Notebook Instance Remotely with SSH** .

ModelArts notebook instances are started by **ma-user** by default. After you access the instance, the default working directory is **/home/ma-user/work**.

Figure 1-6 Working directory example

Most notebook instances in a dedicated resource pool are started as user **root**. The details are as follows:

- When you log in to the terminal as user **root**, the system automatically runs the **source /home/ma-user/.bashrc** command to synchronize the environment variables of user **ma-user**. To disable this function, set the environment variable **export DISABLE_MA_USER_BASHRC** to **true** in the custom image to prevent the **/home/ma-user/.bashrc** file from being loaded.
- If the instance is started by user **root**, only user **root** can be used for SSH remote connection. On the notebook instance details page, you can view the SSH remote development address.

Figure 1-7 Using user **root** for remote SSH connection

Mounting Directories of Notebook Containers

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

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

For details about directory mounting of a notebook instance, see **Table 1-3**. The following mounting points are not saved when images are saved.

Table 1-3 Mounting directories

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

FAQ

- **How do I use EVS in a development environment?**

  When creating a notebook instance, select a small-capacity EVS disk. You can scale out the disk as needed. For details, see **Dynamically Expanding EVS Disk Capacity**.

- **How do I use an OBS parallel file system in a development environment?**

  When training data in a notebook instance, you can use the datasets mounted to a notebook container, and use an OBS parallel file system. For details, see **Dynamically Mounting an OBS Parallel File System**.

- **How do I switch back to JupyterLab 3 if an error occurs during the startup of JupyterLab 4?**

  Locate the target instance in the list and click **Start** in the **Operation** column. In the displayed dialog box, select JupyterLab 3 and click **OK**.

- **Can I use both JupyterLab 3 and 4 in a project?**

  Not recommended. Each JupyterLab instance runs independently. Therefore, you need to create an instance for each version. To try different versions, you can start them in different containers or environments. Pay attention to the following:

  – The configuration file and data path may vary according to the version. Ensure the independence of data and configuration.

  – Running multiple versions at the same time may cause port conflicts or other resource competition problems.

- **How do I upgrade a notebook instance from JupyterLab 3 to JupyterLab 4?**

  You can use either of the following methods to upgrade to JupyterLab 4. For details, see **Upgrading JupyterLab**.

  – Method 1: Stop the notebook instance and upgrade JupyterLab.

  – Method 2: Save the notebook instance and upgrade JupyterLab.

- **Can I use GDB in a notebook instance?**

  No. GDB needs Docker with privileged containers. For security purposes, the development environment does not allow privileged containers. Therefore, GDB cannot be used in notebook instances.

1.3 Creating a Notebook Instance (Old Page)

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

Constraints

- When a notebook instance is created, auto stop is enabled by default. The notebook instance will automatically stop at the specified time.
- Only running notebook instances can be accessed or stopped.
- By default, each IAM user can create a maximum of 10 notebook instances.
- For single-PU instances powered by Snt9b2*x* (like Snt9b23) or D310P-300 resource pools, EVS disks cannot be used to create notebook instances (when **Storage** is set to **EVS**).
- Notebook does not support open ports for external services.

Precautions

Being a commissioning environment, notebooks support downloading from the public network, but the public network proxy is used. Therefore, do not download

large files (larger than 10 GB) in a notebook instance. Notebook provides limited public network access bandwidth, which ensures only network connectivity but not download speed.

Billing

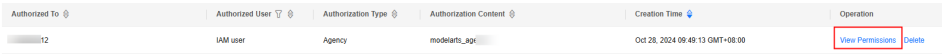
Notebook instance is billed as follows:

- A running notebook instance will be billed based on used resources. The fees vary depending on your selected resources. For details, see **Pricing Details**. When a notebook instance is not used, stop it.
- If you select EVS for storage when creating a notebook instance, the EVS disk will be continuously billed if the instance is not deleted. Stop and delete the notebook instance and delete unused EVS resources to avoid unnecessary fees. For details, see **Development Environment**.

Procedure

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Permission Management** and check whether the access authorization has been configured. If not, configure access authorization. For details, see **Configuring Agency Authorization for ModelArts with One Click**.

   Figure 1-8 Viewing agency configurations

2. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
3. Click **Create Notebook** in the upper right corner. On the displayed page, configure the parameters.
   a. Configure the basic information of the notebook instance, including its name, description, and auto stop status. For details, see **Table 1-4**.

      Figure 1-9 Basic information of a notebook instance

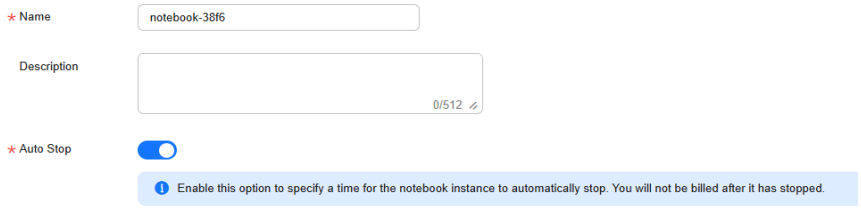
      

Table 1-4 Basic parameters

Parameter	Description
Name	Name of the notebook instance, which is automatically generated by the system. You can rename it based on service requirements. A name consists of a maximum of 128 characters and cannot be empty. It can contain only digits, letters, underscores (_), and hyphens (-).
Description	Brief description of the notebook instance
Auto Stop	Automatically stops the notebook instance at a specified time. This function is enabled by default. The default value is **1 hour**, indicating that the notebook instance automatically stops after running for 1 hour and its resource billing will stop then. The options are **1 hour**, **2 hours**, **4 hours**, **6 hours**, and **Custom**. You can select **Custom** to specify any integer from 1 to 72 hours. • **Stop as scheduled**: If this option is enabled, the notebook instance automatically stops when the running duration exceeds the specified duration. **NOTE** To protect in-progress jobs, a notebook instance does not automatically stop immediately at the auto stop time. Instead, there is a 2 to 5 minutes delay for you to renew the auto stop time.

b. Configure notebook parameters, such as the image and instance flavor. For details, see **Table 1-5**.

Table 1-5 Notebook instance parameters

Parameter	Description
Image	Preset images and custom images are supported. • Preset images are the AI engines built in ModelArts. • You can use a custom image created by yourself. Use either of the following ways to create a custom image: – Save the instance created using the public image and use it as a custom image. For details, see **Saving a Notebook Instance**. – Create a custom image using a base, enterprise, or third-party image. To create a custom image, you must comply with the image specifications. After the build is complete, you must register the image on the **Image Management** page of ModelArts so that the image can be used in a notebook instance. For details, see **Creating a Custom Image**. An image corresponds to an AI engine. When you select an image during instance creation, the AI engine is specified accordingly. Select an image as required. Enter a keyword of the image name in the search box on the right to quickly search for the image. You can change an image on a stopped notebook instance.
JupyterLab Version	ModelArts notebook supports the following two versions of JupyterLab. The default version is JupyterLab 4. • JupyterLab 4: This version has significantly improved user experience, functions, and performance. For details, see **Upgrading JupyterLab**. • JupyterLab 3: Support for creating new notebook instances using JupyterLab 3.2.3 will be discontinued in April 2026. Additionally, technical support, including updates for new features, vulnerability/issue fixes, patch upgrades, service ticket guidance, and online troubleshooting, will no longer be provided. These services will no longer be applicable to the O&M assurance of ModelArts. You are advised to use JupyterLab 4.

Parameter	Description
Resource Type	Public and dedicated resource pools are available. A dedicated resource pool allows for mixed deployments using CPU, NPU, and GPU resources. If your node type supports both GPU and CPU, you can choose either GPU or CPU instance specifications. • **Public resource pool**: It is billed based on the runtime of your notebook instances. • **Dedicated resource pool**: The resources provided in a dedicated resource pool are exclusive and more controllable. In the **Resource Pool** area, click **Select Resource Pool**. On the **Select Dedicated Resource Pool** page, select a dedicated resource pool as required and click **OK**. If you do not have a dedicated resource pool, click **Buy Dedicated Resource Pool** in the lower part of the page to purchase a dedicated resource pool. For details about the parameters for purchasing a dedicated resource pool, see **Creating a Standard Dedicated Resource Pool**. **NOTE** If the dedicated resource pool you purchased is a single-node Tnt004 pool whose specification is GPU: 1*tnt004 | CPU: 8 vCPUs and 32 GiB (modelarts.vm.gpu._tnt004u8), when you use the cluster to create a notebook instance, the Tnt004 card is idle but is displayed as sold out or the creation fails due to insufficient resources, contact technical support.
Type	This parameter is displayed when **Resource Type** is set to **Public resource pool**. Processor type, which can be **CPU** or **GPU**. The chips vary depending on the selected image. GPUs deliver better performance that CPUs but at a higher cost. Select a chip type as needed.
Instance Specifications	The available resource specifications vary among chip types. Select the specifications based on your needs.

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

Parameter	Description
	Figure 1-10 Configuring the secret values  **NOTE** When using DEW secrets to mount storage, you must include 'accessKeyId' and 'secretAccessKey' (corresponding to your AK and SK, respectively). Ensure the information provided is correct, or the function may not work as expected. **EVS** and **SFS** are all mounted to the **/home/ma-user/work** directory. You can add a data storage path during the runtime of a notebook instance by referring to **Dynamically Mounting an OBS Parallel File System**. The data is retained in **/home/ma-user/work**, even if the notebook instance is stopped or restarted. When a notebook instance is deleted, the EVS storage is released and the stored data is not retained. SFS can be mounted to a new notebook instance and data can be retained.
Extended Storage	If you need multiple data storage paths, click **Add Extended Storage** to add more storage mount directories. You can add an OBS, PFS, or SFS directory. Constraints: • For each type, a maximum of five directories can be mounted. • The directories must be unique and cannot be mounted to a blacklisted directory. Nested mounting is allowed. Blacklisted directories are those with the following prefixes: **/data/**, **/cache/**, **/dev/**, **/etc/**, **/bin/**, **/lib/**, **/sbin/**, **/modelarts/**, **/train-worker1-log/**, **/var/**, **/resource_info/**, **/usr/**, **/sys/**, **/run/**, **/tmp/**, **/infer/**, and **/opt/** After this parameter is configured, the notebook instance details page is displayed. Click Storage **Storage** > **Extended Storage** to view or edit the extended storage information. If the number of storage devices does not reach the maximum, you can click **Add Extended Storage**.

Parameter	Description
Remote SSH	• After you enable this function, you can remotely access the development environment of the notebook instance from your local development environment. • When a notebook instance is stopped, you can update the SSH configuration on the instance details page. **NOTE** The notebook instances with remote SSH enabled have VS Code plug-ins (such as Python and Jupyter) and the VS Code server package pre-installed, which occupy about 1 GB persistent storage space.
Key Pair	Set a key pair after remote SSH is enabled. Select an existing key pair. Alternatively, click **Create** on the right of the text box to create one on the **DEW console**. On the displayed **Create Account Key Pair** page, configure the parameters. After a notebook instance is created, you can change the key pair on the instance details page. **CAUTION** Download the created key pair and properly keep it. When you use a local IDE to remotely access the notebook development environment, the key pair is required for authentication.
VPC Access	After this function is enabled, the instance can be mounted to your VPC to implement access of multiple network planes. Before using this function, you need to configure VPC fine-grained access authorization by referring to **Creating an IAM User and Granting ModelArts Permissions**. If you have the **VPC Administrator** permission, you do not need to set this parameter. • **VPC**: Select an existing VPC from the drop-down list on the right or create a VPC as required. • **Subnet**: After you select a VPC, the default subnet is displayed. Select an existing subnet from the drop-down list on the right or create a subnet as required. • **Security Group**: Select an existing security group or create a security group.

c. (Optional) Add tags to the notebook instance. Enter a tag key and value and click **Add**.

Table 1-6 Adding a tag

Parameter	Description
Tags	ModelArts can work with Tag Management Service (TMS). When creating resource-consuming tasks in ModelArts, for example, training jobs, configure tags for these tasks so that ModelArts can use tags to manage resources by group. For details about how to use tags, see **How Does ModelArts Use Tags to Manage Resources by Group?** After adding a tag, you can view, modify, or delete the tag on the notebook instance details page.

NOTE

You can select a predefined TMS tag from the tag drop-down list or customize a tag. Predefined tags are available to all service resources that support tags. Customized tags are available only to the service resources of the user who has created the tags.

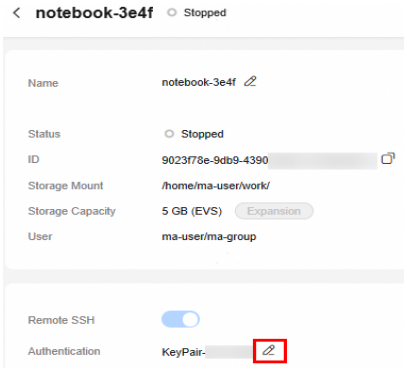
4. Click **Next**.
5. After confirming the parameter settings, click **Submit**.

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

   If an error occurs when you create or use a notebook instance, you can rectify the fault by referring to **Troubleshooting** and **FAQs**.
6. In the notebook instance list, click the instance name. On the instance details page that is displayed, view the instance configuration.

   When the SSH remote development feature is enabled and the notebook instance is in the **Stopped** state, you can click to the right of **Authentication** to update the key pair. For details about how to enable SSH remote development, see **Configuring Remote SSH Connection for a Notebook Instance**.

Figure 1-11 Updating a key pair



In the **Storage** tab, click **Mount Storage** to mount an OBS parallel file system to the instance for reading data. For details, see **Dynamically Mounting an OBS Parallel File System**.

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

Accessing a Notebook Instance

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

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

ModelArts notebook instances are started by ma-user by default. After you access the instance, the default working directory is **/home/ma-user/work**.

Figure 1-12 Working directory example

Most notebook instances in a dedicated resource pool are started as user **root**. The details are as follows:

- When you log in to the terminal as user **root**, the system automatically runs the **source /home/ma-user/.bashrc** command to synchronize the environment variables of user **ma-user**. To disable this function, set the

environment variable **export DISABLE_MA_USER_BASHRC** to **true** in the custom image to prevent the **/home/ma-user/.bashrc** file from being loaded.

- If the instance is started by user **root**, only user **root** can be used for SSH remote connection. On the notebook instance details page, you can view the SSH remote development address.

  Figure 1-13 Using user **root** for SSH remote connection

Mounting Directories of Notebook Containers

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

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

For details about directory mounting of a notebook instance, see **Table 1-7**. The following mounting points are not saved when images are saved.

Table 1-7 Mounting directories

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

FAQ

1. How do I use EVS in a development environment?

   When creating a notebook instance, select a small-capacity EVS disk. You can scale out the disk as needed. For details, see **Dynamically Expanding EVS Disk Capacity**.

2. How do I use an OBS parallel file system in a development environment?

   When training data in a notebook instance, you can use the datasets mounted to a notebook container, and use an OBS parallel file system. For details, see **Dynamically Mounting an OBS Parallel File System**.

3. How do I switch back to JupyterLab 3 if an error occurs during the startup of JupyterLab 4?

Locate the target instance in the list and click **Start** in the **Operation** column. In the displayed dialog box, select JupyterLab 3 and click **OK**.

4. Can I use both JupyterLab 3 and 4 in a project?

   Not recommended. Each JupyterLab instance runs independently. Therefore, you need to create an instance for each version. To try different versions, you can start them in different containers or environments. Pay attention to the following:

   - The configuration file and data path may vary according to the version. Ensure the independence of data and configuration.
   - Running multiple versions at the same time may cause port conflicts or other resource competition problems.

5. Can I use GDB in a notebook instance?

   No. GDB needs Docker with privileged containers. For security purposes, the development environment does not allow privileged containers. Therefore, GDB cannot be used in notebook instances.

1.4 Notebook Storage

1.4.1 Storage Types

Storage options vary based on performance, ease of use, and cost. No single storage type fits every need. Explore cloud storage scenarios to improve your usage.

Table 1-8 On-cloud storage applications

Storage Type	Application	Advantage	Disadvantage
EVS	It is suitable for data and algorithm exploration in the development environment and offers good performance.	Block storage SSDs feature better overall I/O performance than NFS. The storage capacity can be dynamically expanded to up to 4,096 GB. As persistent storage, EVS disks are mounted to **/home/ma-user/work**. The data in this directory is retained after the instance is stopped. The storage capacity can be expanded online based on demand.	This type of storage can only be used in a single development environment.

Storage Type	Application	Advantage	Disadvantage
Object Storage Service – Parallel File System	**NOTE** • PFS is currently in the restricted use phase. To use this feature, contact Huawei technical support. • Only parallel file systems in the same region can be mounted. Parallel file systems mounted as persistent storage for AI development and exploration. 1. Dataset storage. Datasets stored in PFS buckets are directly mounted to notebook instances for browsing and data processing and can be directly used during training. Select PFS when creating a notebook instance. After the instance is running, the parallel file system that carries the datasets is dynamically mounted to the notebook instances. For details, see **Dynamically Mounting an OBS Parallel File System**. 2. Code storage. After debugging on a notebook instance, specify the OBS path as the code path for	PFS is an optimized high-performance object storage file system with low storage costs and large throughput. It can quickly process high-performance computing (HPC) workloads. PFS mounting is recommended if OBS is used. **NOTE** Package or split the data to be uploaded by 128 MB or 64 MB. Download and decompress the data in local storage for better I/O and throughput performance.	The performance of frequent read and write operations on small files is poor, which may cause notebook instance freezing. Exercise caution when using this storage type for heavy model training and large file decompression. **NOTE** Before mounting PFS storage to a notebook instance, grant ModelArts with full read and write permissions on the PFS bucket. The policy will be retained even after the notebook instance is deleted.

Storage Type	Application	Advantage	Disadvantage
	starting training, facilitating temporary modification. 3. Training observation. Mount storage to the training output path such as the path to training logs. In this way, view and check training on the notebook instance in real time. This is especially suitable for analyzing the output of jobs trained using **Using TensorBoard Visualization Jobs in JupyterLab**.		

Storage Type	Application	Advantage	Disadvantage
Object Storage Service – Bucket	**NOTE** • OBS bucket is currently in the restricted use phase. To use this feature, contact Huawei technical support. • Only OBS buckets in the same region can be mounted. When uploading or downloading a large amount of data in the development environment, you can use OBS buckets to transfer data.	Low storage cost and high throughput, but average performance in reading and writing small files. It is good practice to package or split the file by 128 MB or 64 MB. In this way, you can download the packages, decompress them, and use them locally.	The object storage semantics is different from the Posix semantics and needs to be further understood. The performance of frequent read and write operations on small files is poor, which may cause notebook instance freezing. Exercise caution when using this storage type in scenarios such as model training and large file decompression.

Storage Type	Application	Advantage	Disadvantage
Scalable File Service	Available only in dedicated resource pools. Use SFS storage in informal production scenarios such as exploration and experiments. Development and training environments can mount the same SFS storage simultaneously, eliminating the need to download data for every training job. Generally, this setup is not suitable for large-scale training exceeding 32 PUs or for models with heavy I/O read/write requirements.	SFS uses NFS and can be shared among multiple development and training environments. It is ideal for light-duty distributed training jobs that do not need extra data downloads at the start.	The performance is lower than that of EVS block storage.
Local storage	First choice for heavy-duty training jobs.	High-performance SSDs for the used VM or BMS, featuring high file I/O throughput. For heavy-duty training jobs, store data in the target directory and then start training. By default, the storage is mounted to the **/cache** directory. For details about the available space of the **/cache** directory, see **What Are Sizes of the /cache Directories for Different Notebook Specifications in DevEnviron?**	The storage lifecycle is associated with the container lifecycle. Data needs to be downloaded each time the training job starts.

Helpful Links

- ModelArts supports dynamic storage and extended storage. For details, see **Dynamically Mounting an OBS Parallel File System**.
- When starting your notebook-based development, use a small EVS disk, like 5 GB, if your initial storage needs are low. Once development ends and you need to train with large datasets, increase the storage size as needed to save costs. For details, see **Dynamically Expanding EVS Disk Capacity**.

1.4.2 Dynamically Mounting an OBS Parallel File System

Overview

A parallel file system is an optimized high-performance file system provided by Object Storage Service (OBS). For details, see **About Parallel File System**.

In ModelArts running notebook containers, the dynamic mounting feature is used to simulate OBS as a local file system. Dynamic mounting uses a mounting tool to convert the object storage protocol into the POSIX file protocol. After the mounting, you can perform application operations on the OBS objects in the notebook container.

Application Scenarios

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

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

Restrictions

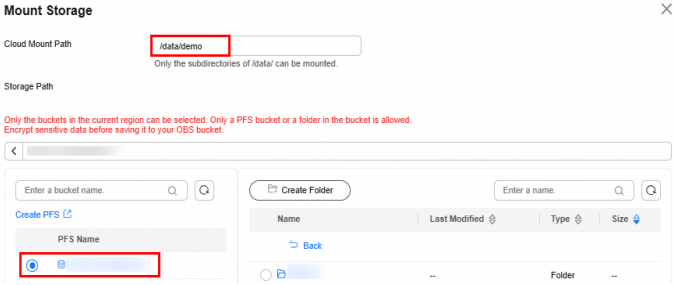
OBS provides object buckets and parallel file systems for storage. The ModelArts notebook supports only the mounting of an OBS parallel file system to **/data/** of a notebook container.

Procedure

Method 1: Through the ModelArts management console

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
1. Select a running notebook instance and click its name. On the notebook instance details page, click the **Storage** tab. From there, click **Mount Storage** and configure mounting parameters.
   a. Set a local mounting directory. Enter a folder name in **/data/**, for example, **demo**. The system will automatically create the folder in **/data/** of the notebook container to mount the OBS file system.
   b. Select the folder for storing the OBS parallel file system and click **OK**.

Figure 1-14 Dynamically mounting an OBS parallel file system



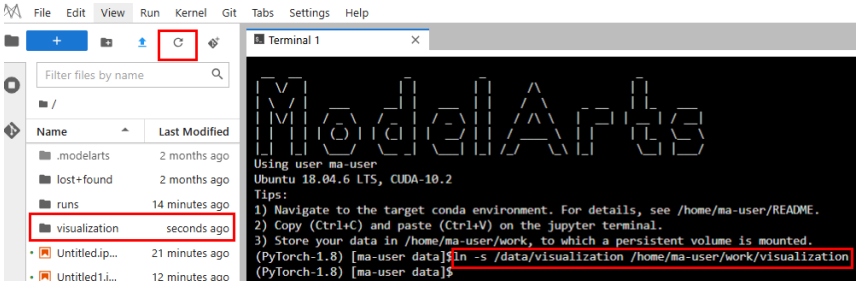
2. View the mounting result on the notebook instance details page.
3. (Optional) Create a soft link in the terminal. The mounted parallel file system directory is displayed in the file directory on the left of the notebook instance.

   For example, run the command below to link the mount point **/data/visualization** to the **/home/ma-user/work/visualization** directory. Replace **visualization** with the actual mount path.

   ```
   ln -s /data/visualization /home/ma-user/work/visualization
   ```

   By running this command, a soft link named **visualization** is created in **/home/ma-user/work** and is directed to **/data/visualization**. Click on the left of the notebook instance to view the new soft link.

Figure 1-15 Creating a soft link



1.4.3 Dynamically Expanding EVS Disk Capacity

Overview

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

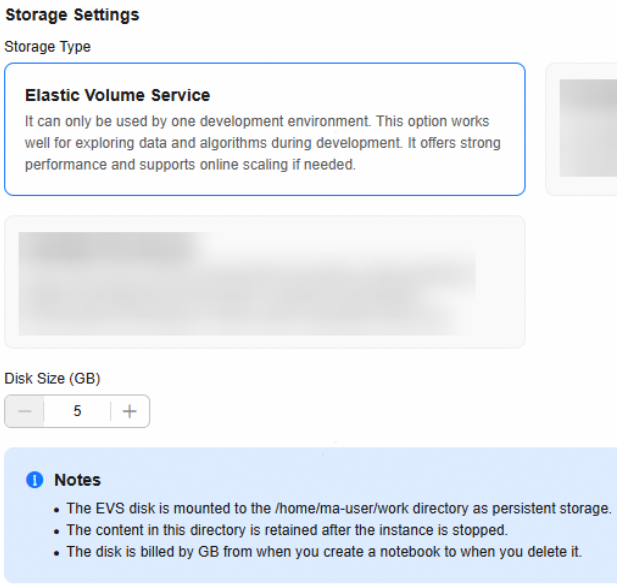
Application Scenarios

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

Restrictions

- The target notebook instance must use EVS for storage.

Figure 1-16 Selecting EVS when creating a notebook instance

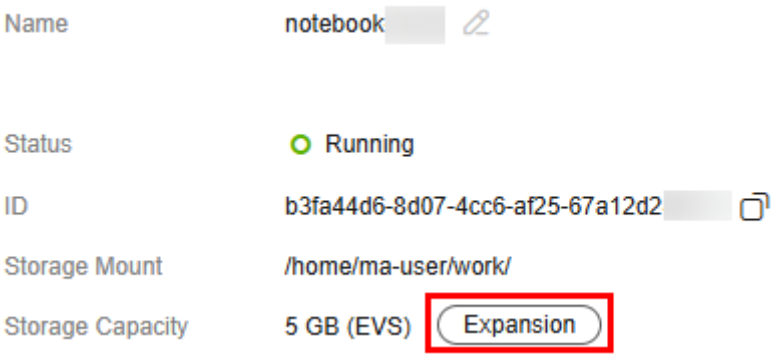


- Up to 100 GB can be expanded at a time. Additionally, the total capacity after expansion cannot exceed 4,096 GB.
- If the original capacity of an EVS disk is 4,096 GB, the disk capacity cannot be expanded.
- After the instance is stopped, the expanded capacity still takes effect. The billing is based on the expanded EVS disk capacity.
- An EVS disk is billed as long as it is used. To stop billing an EVS disk, delete data from the EVS disk and release the disk.

Procedure

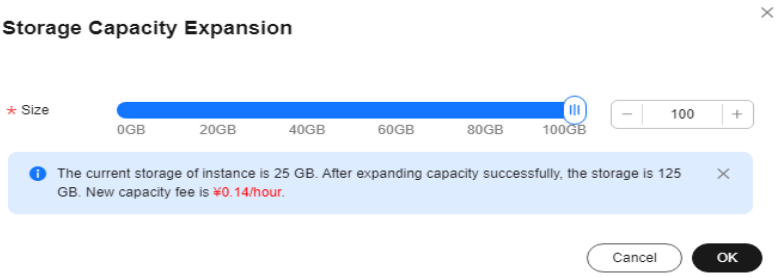
1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Click the name of a running notebook instance. On the instance details page, click **Expansion**.

Figure 1-17 Instance details page



3. Set the capacity to be expanded and click **OK**. **Expanding** shows that the capacity expansion is in progress. After the expansion, the displayed storage capacity is the expanded capacity.

Figure 1-18 Capacity expansion



1.5 Managing Notebook Instances

1.5.1 Searching for a Notebook Instance

Searching for an Instance

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

- **Grant the permission to the IAM user for viewing all notebook instances**. Go to the **Notebook** page. On the displayed page, click **All**.

Figure 1-19 Viewing all

- Filtering notebook instances:
  - In the upper part of the page, click **Being billed** or **Running** to filter notebook instances. Hover over **Being billed** to view the number of notebook instances currently consuming billing resources (compute and storage).
  - The search box defaults to **Name**. You can also filter by **ID**, **Resource Pool ID**, **Status**, **Billing Type**, or **Resource Tag** as needed.

Assigning the Required Permissions

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

1. Log in to the **ModelArts console** 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 console.
2. On the IAM console, choose **Permissions** > **Policies/Roles** from the navigation pane, click **Create Custom Policy** in the upper right corner, and create two policies.

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

   - **Policy Name**: Enter a custom policy name, for example, **Viewing all notebook instances**.
   - **Policy View**: Select **Visual editor**.
   - **Policy Content**: Select **Allow**, **ModelArts Service**, **modelarts:notebook:listAllNotebooks**, and default resources.

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

   - **Policy Name**: Enter a custom policy name, for example, **Viewing all users of the current IAM project**.
   - **Policy View**: Select **Visual editor**.
   - **Policy Content**: Select **Allow**, **Identity and Access Management**, **iam:users:listUsers**, and default resources.
3. In the navigation pane, choose **User Groups**. Then, click **Authorize** in the **Operation** column of the target user group. On the **Authorize User Group** page, select the custom policies created in 2, and click **Next**. Then, select the scope and click **OK**.

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

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

Starting Notebook Instances of Other IAM Users

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

1.5.2 Updating a Notebook Instance

Changing an Image

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

Notes:

- After the image is changed, the notebook instance may fail to be started, as the notebook instance specifications corresponding to the image do not

match. The charging rule may also change. Exercise caution when performing this operation.

- When you change the image, data in mounted directories like **/home/ma-user/work** or **/data** will not be lost. However, data in non-mounted directories such as **/home/ma-user** will be erased.

Procedure:

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Locate the target notebook instance in the list and choose **More** > **Change Image** in the **Operation** column.
3. On the **Change Image** page, select a preset image or custom image as required, select the notices, and click **OK**.

   The images will be matched based on the hardware model. If the images do not match, they cannot be selected.

   After the image is changed, you can view it in the **Image** column on the notebook page.

Changing the Specifications of a Notebook Instance

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

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Locate the target notebook instance in the list and choose **More** > **Modify Instance Specifications** in the **Operation** column.
3. On the **Modify Instance Specifications** page, select the target specifications, select **I acknowledge the potential risks and agree to proceed with the change**, and click **OK**.

   After the specifications are changed, you can view it in the **Instance Specifications** column on the notebook page.

   NOTE

   The instance specifications can be changed only when there are other options in the cluster. If no other specifications are available, the change cannot be performed.

Configuring Remote SSH Connection for a Notebook Instance

ModelArts allows you to modify the SSH configuration only when the notebook instance is stopped. Remote SSH cannot be disabled once it is enabled.

If remote SSH connection is not configured when you create the notebook instance and you want to enable it after the instance is created, go to the notebook instance details page, and enable SSH configuration. If an IAM user needs to start a notebook instance of another user's, change the key pair.

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Click the target notebook instance to access its details page. Enable **Remote SSH**, update the key pair, and click **OK**.

Setting Node Affinity Scheduling

You can enable, modify, or disable node affinity scheduling only for notebook instances that fail to be stopped or started and whose resource pool type is dedicated resource pool. For details about how to stop a notebook instance, see **Starting, Stopping, or Deleting a Notebook Instance**.

Enabling node affinity scheduling

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Enable node affinity scheduling in either of the following ways:
   - Method 1:
     i. On the **Notebook** page, locate the target notebook instance (stopped or fails to be started), and choose **More** > **Node Affinity Scheduling** in the **Operation** column.
     ii. In the displayed dialog box, set the affinity type, strength, and node, and click **OK**.
     - If **Affinity Type** is set to **Node Affinity** and **Strength** is set to **Weak**, the system will try to schedule pods to the specified node, however, the scheduling may fail.
     - If **Affinity Type** is set to **Node Affinity** and **Strength** is set to **Strong**, the system will ensure that the pods are scheduled to the specified node. Otherwise, the pods will not be scheduled.
     - If **Affinity Type** is set to **Node Anti-Affinity** and **Strength** is set to **Weak**, the system will avoid scheduling pods to the specified node, however, this operation may fail.
     - If **Affinity Type** is set to **Node Anti-Affinity** and **Strength** is set to **Strong**, the system will ensure that the pods will not be scheduled to the specified node. Otherwise, this operation will not be performed at all. You cannot select all nodes. However, you need to reserve at least one available node. Otherwise, the scheduling policy cannot be executed.

     iii. On the **Notebook** page, click the target notebook instance (stopped or fails to be started) to access its details page. If the node affinity scheduling configuration is displayed, node affinity scheduling has been enabled.
   - Method 2:
     i. On the **Notebook** page, click the target notebook instance (stopped or fails to be started).
     ii. On the displayed notebook instance details page, click **Enable** next to **Node affinity scheduling**. In the displayed dialog box, set the affinity type, strength, and node as required, and click **OK**.
     iii. If the node affinity scheduling configuration is displayed, node affinity scheduling has been enabled.

Modifying node affinity scheduling

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

2. Modify node affinity scheduling in either of the following ways:
   - Method 1:
     i. On the **Notebook** page, locate the target notebook instance that is stopped or fails to be started, and choose **More** > **Node Affinity Scheduling** in the **Operation** column.
     ii. In the displayed dialog box, click **Modify Policy**, and click **OK**.
     iii. In the **Affinity-based Scheduling Policy** dialog box, modify the affinity type, strength, or node as required, and click **OK**.
     iv. On the **Notebook** page, click the target notebook instance (stopped or fails to be started) to access its details page. If the node affinity scheduling configuration has been updated, the latest information will be displayed.
   - Method 2:
     i. On the **Notebook** page, click the target notebook instance (stopped or fails to be started).
     ii. On the displayed notebook instance details page, click  next to **Node affinity scheduling**. In the displayed dialog box, modify the affinity type, strength, and node as required, and click **OK**.
     iii. If the updated node affinity scheduling configuration is displayed, the modification is successful.

Disabling node affinity scheduling

After node affinity scheduling is disabled, intelligent scheduling is performed according to system scheduling rules.

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Disable node affinity scheduling in either of the following ways:
   a. Method 1:
      i. On the **Notebook** page, locate the target notebook instance that is stopped or fails to be started, and choose **More** > **Node Affinity Scheduling** in the **Operation** column.
      ii. In the displayed dialog box, disable node affinity scheduling and click **OK**. Then, click **OK** again in the displayed dialog box.
      iii. On the **Notebook** page, locate the target notebook instance (stopped or fails to be started), and choose **More** > **Node Affinity Scheduling** in the **Operation** column. If a message is displayed, indicating that node affinity scheduling is not enabled, disabling node affinity scheduling is successful.
   b. Method 2:
      i. On the **Notebook** page, click the target notebook instance (stopped or fails to be started).
      ii. On the notebook instance details page, click  on the right of **Node affinity scheduling**. In the displayed dialog box, disable the affinity policy and click **OK**.
      iii. In the displayed dialog box, click **OK**.

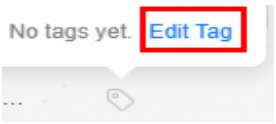
iv. If **Enable** is available on the right of **Node affinity scheduling**, disabling node affinity scheduling is successful.

Editing a Tag

You can modify, add, or delete tags of a notebook instance.

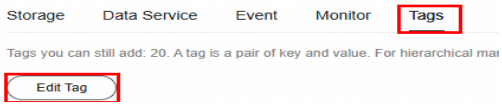
1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Edit the tags in either of the following methods:
   a. Method 1: On the **Notebook** page, locate the notebook instance, hover over in the **Tags** column, and click **Edit Tag**.

      Figure 1-20 Editing a tag

   b. Method 2: On the **Notebook** page, click the notebook instance name to access its details page. In the **Tags** tab, click **Edit Tag**.

      Figure 1-21 Editing a tag

3. On the **Edit Tag** page, you can modify, add, or delete tags.
   - To modify a tag, change the tag key and value for an existing tag as required, and click **OK**.
   - To add a tag, click **Add Tag**, enter or select a tag key, enter a tag value, and click **OK**. A maximum of 20 tags can be added to a notebook instance.
   - To delete a tag, click **Delete** on the right of an existing tag and click **OK**.

   To add the same tag to different cloud resources, use Tag Management Service (TMS) to create predefined tags. For details, see **Creating Predefined Tags**.

   NOTE

   Predefined tags are available to all service resources that support tags. Customized tags are available only to the service resources of the user who has created the tags.

   You can view the modified tags in the **Tags** column on the **Notebook** page or in the **Tags** tab of the notebook instance details page.

1.5.3 Starting, Stopping, or Deleting a Notebook Instance

Starting or Stopping a Notebook Instance

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

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Start or stop the target notebook instance.
   - **Starting a notebook instance**

     Only stopped notebook instances can be started.

     i. Click **Start** in the **Operation** column.

     ii. In the displayed dialog box, select a JupyterLab version and set auto stop as required, and click **OK**.
   - **Stopping a notebook instance**

     Only running notebook instances can be stopped.

     i. Click **Stop** in the **Operation** column.

     ii. In the displayed dialog box, read the message and click **OK**.

     CAUTION

     After a notebook instance is stopped:

     - Data in the **/home/ma-user/work** directory and directories dynamically mounted to **/data** is saved. Data in other directories will be deleted. For example, the external dependency packages installed in other directories in the development environment will be deleted. Save your development environment settings as an image. For details, see **Saving a Notebook Instance**.
     - The notebook instance will no longer be billed. If the instance has an EVS disk attached, the storage space will still be billed.

Deleting a Notebook Instance

Delete the notebook instances that are not needed. **Deleted notebook instances cannot be recovered. Exercise caution. After a notebook instance is deleted, the data stored in the mounted directory will be deleted.**

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

1.5.4 Saving a Notebook Instance

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

image, and save the running instance as a container image. After the image is saved, the default working directory is the **/** path in the root directory.

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

The image saving function implements version iterations by stacking a new layer onto the original image. Due to the immutability of image layers, the size of the newly generated image will always exceed that of the original, regardless of whether addition or deletion is performed.

NOTE

- If the image fails to be saved, view the event on the notebook instance details page. For details, see **Viewing Notebook Events**.
- The image to be saved should not be larger than 35 GB and there should be no more than 125 layers. Otherwise, the image may fail to be saved. For details, see **Space Allocation for Container Engines**.
  - If a dedicated resource pool is used, log in to the ModelArts console. On the dedicated resource pool scaling page, configure the container engine size as needed. For details, see **Resizing a Dedicated Resource Pool**.
  - If the fault persists, contact technical support.
- When you use the containerd image saving function, pay attention to the following:
  - Do not download files and save images at the same in an instance to prevent performance deterioration caused by resource competition.
  - If there is a file larger than 10 GB in an instance system folder (for example, **/opt/**), you should not use the image saving function, in case the image size is too large or the saving duration is too long.

Prerequisites

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

Saving an Image

1. Locate the target notebook instance in the list and click **Save Image** in the **Operation** column.
2. On the displayed page, set the parameters, select **I understand the risks caused by the change and agree to save the image**, and click **OK**.

   You can save the image as a new version of an existing image or create a new image.

   - **Save as new version**: Save the image as a new version of a registered image, that is, in an existing image folder in the **Registered Images** tab.
   - **Create new image**: The image will be saved to Software Repository for Container (SWR) and automatically registered with ModelArts upon completion.

Table 1-9 Parameters

Save Options	Parameter	Description
Save as new version	Image	Click the image selection box. In the **Image** panel, select an image from the basic or enterprise edition as needed, and then click **OK**.
	Image Version	Enter a version for the custom image. It can contain letters, digits, underscores (_), hyphens (-), and periods (.). The maximum length is 64 characters.
	Description	Enter a description for the custom image. It cannot contain the characters & < > " ' /. The length must be between 0 and 256 characters.
Create new image	Image Repository	Select **SWR Basic Edition** or **SWR Enterprise Edition** as needed. By default, **SWR Basic Edition** is used. If you want to use an enterprise repository, go to the **SWR Enterprise Edition console** to register one. The repository in the Enterprise Edition is identified by its registry name. For details about the differences between the basic and enterprise editions, see **Differences Between SWR Basic and Enterprise Editions**.
	Domain Name	This parameter is mandatory only if **Image Repository** is set to **SWR Enterprise Edition**. It specifies the access address of the SWR Enterprise Edition instance.
	Namespace	This parameter is mandatory only if **Image Repository** is set to **SWR Enterprise Edition**. Namespaces are used to isolate image repositories in SWR Enterprise Edition.
	Organization	This parameter is mandatory only if **Image Repository** is set to **SWR Basic Edition**. Organizations are used in SWR to isolate images. An organization can represent a company or department, and its images are managed centrally under that organization. Different organizations may have images with identical names. An IAM user can join different organizations. Users in an organization can share all images in the organization. If no organization is available, click **Go to SWR to create one**. For details, see **Organization Management**.

Save Options	Parameter	Description
	Image Name	Enter a name for the custom image. It can only contain lowercase letters, digits, and special characters (_/.-). It must start and end with a lowercase letter or digit and cannot contain more than two consecutive underscores. The maximum length is 128 characters.
	Image Version	Enter a version for the custom image. It can contain letters, digits, underscores (_), hyphens (-), and periods (.). The maximum length is 64 characters.
	Description	Enter a description for the image. It cannot contain the characters & < > " ' /. The length must be between 0 and 256 characters.

3. Wait until the image is saved. The image will be saved as a snapshot. The process takes approximately 3 to 10 minutes, during which the instance status will be **Snapshotting**. During this period of time, do not perform any operations on the instance. After the image is saved, the instance status changes to **Running**.

NOTICE

- The time consumed during the snapshotting process is counted toward the total running time of the instance. If the instance stops because its scheduled running time expires while a snapshot is in progress, the image saving process will fail.
- The connection may temporarily drop during the saving process; it will recover once the operation is complete.
- The saved image does not contain files and data in the mount directory (**/home/ma-user/work**) used for persistent storage.

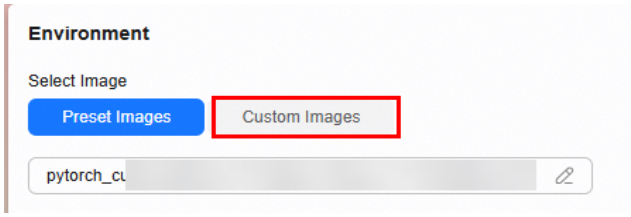
4. View image details.
   a. In the navigation pane on the left, choose **Asset Management** > **Image Management**.
   b. In the **Registered Images** tab, click the name of the image to go to the image details page and view the image details.

Using a Custom Image to Create a Notebook Instance

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

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

Figure 1-22 Selecting a custom image to create a notebook instance



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

Which Data Can Be Saved When I Save an Image?

- Data that can be saved: Files and directories that are statically added to images during container building.

  For example, dependencies and the **/home/ma-user** directory are saved in the image environment.

- Data that cannot be saved: Mounting directories or data volumes that are dynamically connected to the host during container startup. You can run the **df -h** command to view the mounted dynamic directories. Data that is not in the **/** path will not be saved.

  For example, data that is persistently stored in **home/ma-user/work** and data that is dynamically mounted to **/data** is not saved.

1.5.5 Viewing Notebook Events

Instance statuses and key operations such as creating, starting, and stopping an instance, and changing the instance flavor are recorded in the backend. You can view the events on the notebook instance details page to monitor the instance statuses. You can refresh events on the right of the **Event** tab. You can also set the interval for automatically refreshing events to 30 seconds, 1 minute, or 5 minutes.

Viewing Events of a Notebook Instance

To view the event details of a notebook, click the notebook name. On the displayed notebook details page, click the **Event** tab.

Notebook Instance Events

Table 1-10 Events during instance creation

Event	Description	Severity	Solution
Scheduled	The instance has been scheduled.	Warning	Normal event, no action required.
PullingImage	The image is being pulled.	Warning	Normal event, no action required.

Event	Description	Severity	Solution
PulledImage	The image has been pulled.	Warning	Normal event, no action required.
NotebookHealthy	The instance is running and healthy.	Major	Normal event, no action required.
CreateNotebookFailed	Creating an instance failed.	Critical	Internal service error. Submit a service ticket to contact O&M engineers.
PullImageFailed	Pulling the image failed.	Critical	Check whether the image selected during instance creation exists. If the image does not exist, select another image to create an instance. If the image exists, submit a service ticket to contact O&M engineers.
FailedCreate	Failed to create notebook container. Please contact SRE to check node {node_name}	Critical	Internal service error. Submit a service ticket to contact O&M engineers.
CreateContainer-Error	Failed to create container. Please contact SRE to check node {node_name}	Critical	Internal service error. Submit a service ticket to contact O&M engineers.
FailedAttachVolume	Failed to attach volume. Please contact SRE to check node {node_name}	Major	Internal service error. Submit a service ticket to contact O&M engineers.
MountVolumeFailed	Mount volume failed; Check whether the DEW secret is correct if the instance cannot change to running in five minutes	Critical	Wait for 5 to 10 minutes and check whether the instance status changes to **Running**. If yes, no action is required. If the status is not changed, check whether the authentication information selected when OBS is used is correct.

Event	Description	Severity	Solution
	Mount volume failed; Check if vpc of sfs-turbo is interconnected if the instance cannot change to running in five minutes	Critical	Wait for 5 to 10 minutes and check whether the instance status changes to **Running**. If yes, no action is required. If the status is not changed, check whether the VPC of the dedicated resource pool has been connected to SFS. For details, see **Accessing a Real-Time Service (VPC High-Speed Channel)**.
	Mount volume failed; Please contact SRE to check node {node_name} if the instance cannot change to running in five minutes	Critical	Wait for 5 to 10 minutes and check whether the instance status changes to **Running**. If yes, no action is required. If the status is not changed, submit a service ticket to contact O&M engineers.

Table 1-11 Events during instance stopping

Event	Description	Severity	Solution
StopNotebook	The instance has been stopped.	Major	Normal event, no action required.
StopNotebook ResourceIdle	The notebook instance will automatically stop or has automatically stopped because resources are idle.	Major	Normal event, no action required.

Table 1-12 Events during instance update

Event	Description	Severity	Solution
UpdateName	Updating the instance name	Warning	Normal event, no action required.
UpdateDescription	Updating the instance description	Warning	Normal event, no action required.
UpdateFlavor	Updating the instance flavor	Major	Normal event, no action required.
UpdateImage	Updating the instance image	Major	Normal event, no action required.
UpdateStorageSize	The instance storage size is being updated. (User %s is updating storage size from %s GB to %s GB.)	Major	Normal event, no action required.
	The instance storage size has been updated. (User %s updated the storage size.)	Major	Normal event, no action required.
UpdateKeyPair	Configured the instance key pair. (User %s updated the instance key pair to {*%s*}.)	Major	Normal event, no action required.
	Updating the instance key pair (User %s updated the instance key pair from %s to %s.)	Major	Normal event, no action required.
UpdateHook	Updating a custom script	Major	Normal event, no action required.
UpdateStorageSizeFailed	Updating the storage size failed because the resources are sold out. (EVS disks are sold out.)	Critical	Go to the details page of of the instance to be scaled out, click the **Storage** tab, and add dynamic storage or expand the storage capacity.
	Updating the storage size failed due to an internal error. (Updating the EVS disk size failed. The O&M engineers are handling the fault.)	Critical	Internal service error. Submit a service ticket to contact O&M engineers.

Table 1-13 Events during image saving

Event	Description	Severity	Solution
SaveImage	The image has been saved.	Major	Normal event, no action required.
SavedImageFailed	Saving the image failed due to processes in D status. (There are processes in 'D' status. Check process status using 'ps -aux' and kill all the processes in 'D' status.)	Critical	Run **ps -aux** to query all processes in the D state, run **kill -9 <PID>** to stop all processes in the D state, and save the image again.
	Saving the image failed because the image is too large. (The container size (%dG) is greater than the threshold (%dG).)	Critical	Delete unnecessary directories and files except those in the **/home/ma-user/work/** directory of the instance. Reduce the container image size to the threshold specified in the event description and try again.
	Saving the image failed due to the limit on the number of layers. (There are too many layers in your image.)	Critical	The number of image layers used for starting the instance exceeds 125. Create an instance startup image. During the creation, you can reduce the number of image layers by combining commands and building the image by phase.
	Saving the image failed due to task timeout. (Image saving failed due to task timeout)	Critical	The task timed out due to a network or dependent service exception. Submit a service ticket to contact O&M engineers.
	Saving the image failed due to SWR service issues.	Critical	SWR service error. Submit a service ticket to contact O&M engineers.

Event	Description	Severity	Solution
CheckImageSize	The notebook container image size is {image_size}G. **{image_size}** indicates the image size, which is a variable.	Warning	Normal event, no action required.
CheckImageLayer	The number of original notebook image layers is {layer_number}. **{layer_number}** indicates the number of image layers, which is a variable.	Warning	Normal event, no action required.
ContainerCommitStarted	Start to commit notebook container.	Warning	Normal event, no action required.
ContainerCommitSuccess	Notebook container commit successfully.	Warning	Normal event, no action required.
ImagePushStarted	Start to push notebook image.	Warning	Normal event, no action required.
ImagePushSuccess	Notebook image push successfully.	Warning	Normal event, no action required.
ContainerCommitFailed	Failed to commit notebook container. Please contact SRE to check node {node_name}. **{node_name}** indicates the node name, which is a variable and is generally in the format of an IP address, for example, **192.168.225.161**.	Warning	Node error or internal service error. Submit a service ticket to contact O&M engineers.
ImagePushFailed	Failed to push Notebook image. Please contact SRE to check node {node_name}.	Warning	Failed to push the image. Try again. If the fault persists, submit a service ticket to contact O&M engineers.

Table 1-14 Events during instance running

Event Name	Description	Severity	Solution
NotebookUnhealthy	The instance is unhealthy.	Critical	This event may be triggered when a debugging task is started in an instance, for example, the task occupies too many CPU, memory, or I/O resources. It can be automatically cleared after the instance load decreases. Wait for a while and refresh the page. If the **NotebookHealthy** event is added, the instance status is normal and no action is required. If the fault persists for a long time, submit a service ticket to contact O&M engineers for assistance.
OutOfMemory	The instance is evicted because the memory usage exceeds the upper limit.	Critical	When an instance process occupies more memory than the applied specifications, this event is triggered by the Kubernetes mechanism and the instance is restarted. After the restart, the instance status changes to **Normal**. In future use, do not perform tasks with high memory usage.
JupyterProcessKilled	The Jupyter process stops abnormally.	Critical	This event may be triggered if the Jupyter process is stopped by mistake or an unknown error occurs in the instance container. The instance will automatically restart. After the restart, the instance status changes to **Normal**.

Event Name	Description	Severity	Solution
CacheVolumeExceed-Quota	The **/cache** file size has exceeded the upper limit.	Critical	This event is triggered when the **/cache** directory file size exceeds the maximum limit allowed by the instance specifications. The instance will automatically restart. After the restart, the instance status changes to **Normal**. In future use, pay attention to the size of the **/cache** directory. For details about the mapping between the space allocated to the directory and the instance specifications, see **What Are the Sizes of the /cache Directories for Resources with Varying Specifications on ModelArts Notebook Instances?**
NotebookHealthy	The instance recovers from an abnormal state to a normal state.	Major	Normal event, no action required.
EVSSoldOut	EVS disks are sold out.	Critical	This event may be triggered when you create a notebook instance and select EVS as the storage type, but EVS disks are sold out. You are advised to switch the storage type to OBS parallel file system or OBS bucket. If you want still want to use EVS, submit a service ticket to contact O&M engineers for capacity expansion.

Table 1-15 Events for dynamic OBS mounting

Event	Description	Severity	Solution
DynamicMountStorage	The OBS storage is mounted.	Major	Normal event, no action required.

Event	Description	Severity	Solution
DynamicUnmountStorage	The OBS storage is unmounted.	Major	Normal event, no action required.

Table 1-16 Events triggered on the user side

Event	Description	Severity	Solution
RefreshCredentialsFailed	Authentication failed.	Critical	Normal event, no action required.

1.5.6 Notebook Cache Directory Alarm Reporting

When creating a notebook instance, you can select CPU, GPU, or Ascend resources based on the service data volume. If you select GPU or Ascend resources, ModelArts mounts hard disks to the cache directory. You can use this directory to store temporary files.

Capacity alarms are not generated for the cache directory of the notebook instance by default. Exceeding the capacity limit will restart the notebook instance. After the restart, multiple configurations are reset, discarding your data and losing the environment. This will affect your experience. You are advised to enable the monitoring and alarms for the cache directory usage and report the data to AOM.

Configuration Process

1. Enter the basic alarm information.
2. **Set an alarm rule.**
   a. Configure monitoring metrics.
   b. Set alarm triggering conditions.
3. **Configure alarm notifications.**
   a. Create a topic, configure the topic policy, and subscribe to the topic.
   b. Create an alarm action rule.
   c. Select the created action rule.

Configuring Alarm Settings

1. Log in to the **AOM console**.
2. Choose **Alarm Center** > **Alarm Rules**. On the displayed page, click **Create Alarm Rule**.
3. Enter the basic alarm information.

4. Set an alarm rule.

   Rule Type: Select **Threshold alarm**.

   Monotored Object: Select **Select resource objects**. Click **Select Resource Object**. A new dialog box is displayed.

   - **Add By**: Select **Dimension**.
   - **Metric Name**: Click **Custom Metrics** and select the cache metrics to be monitored. Example: **ma_container_notebook_cache_dir_size_bytes** (total size of the cache directory) and **ma_container_notebook_cache_dir_util** (usage of the cache directory)
   - **Dimension**: Select a metric dimension, for example, *service_id:xxx*, and click **Confirm**.

   After setting the monitored object, set **Statistic** and **Statistical Period**.

   Alarm Condition: Set this parameter based on your needs.

Figure 1-23 Selecting monitored objects

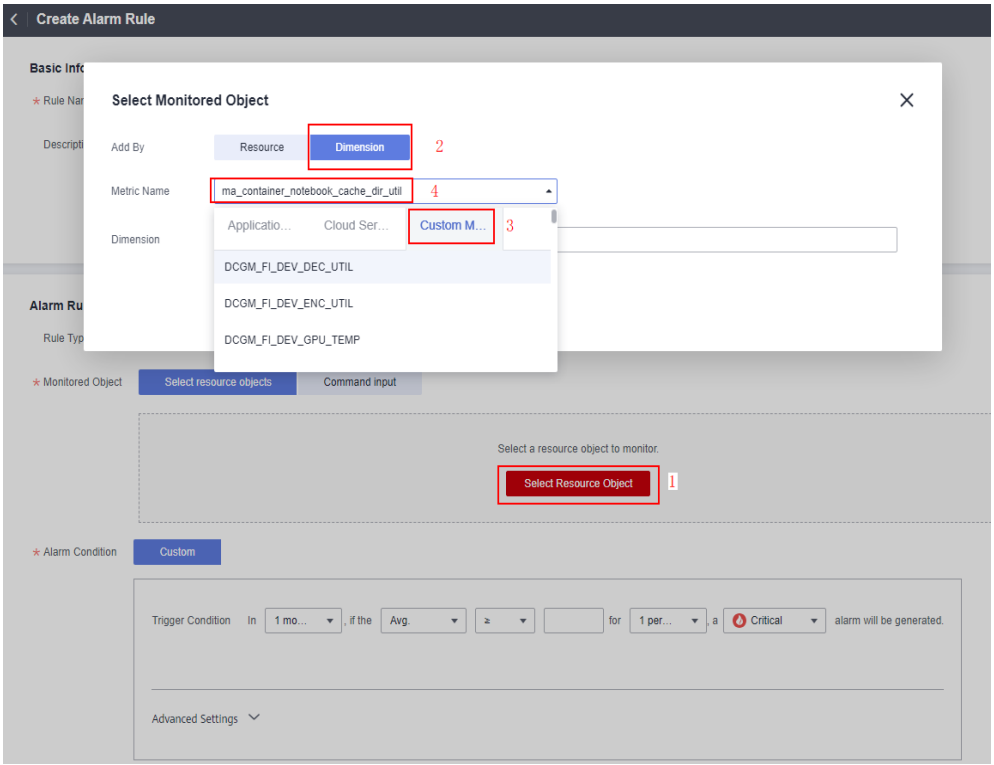


Figure 1-24 Configuring statistics method

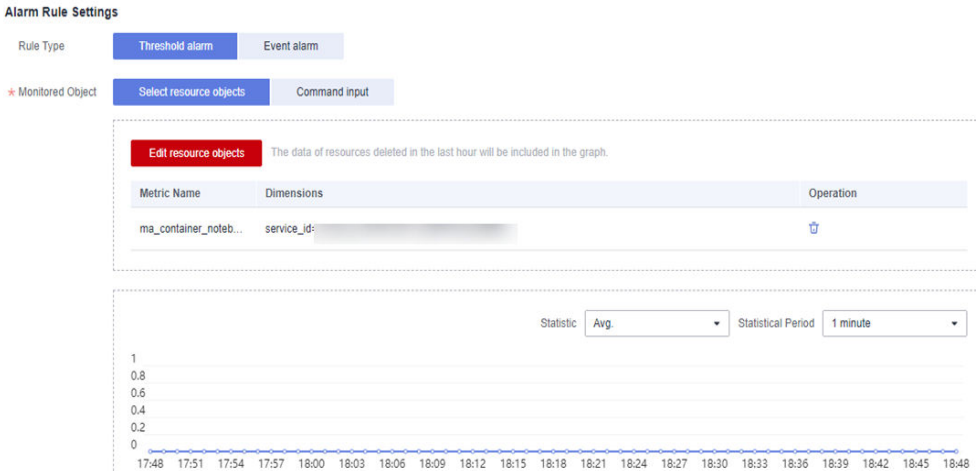
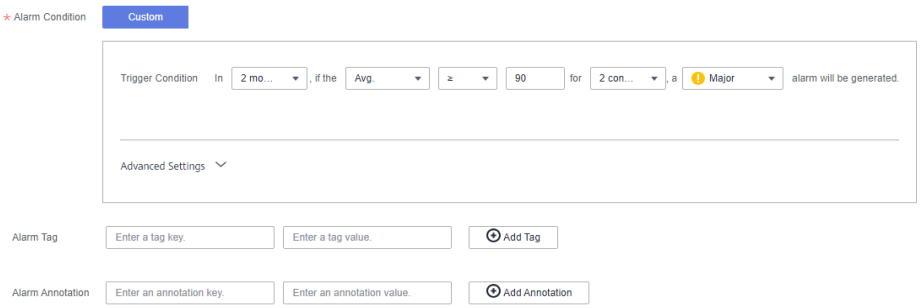


Figure 1-25 Configuring alarm conditions



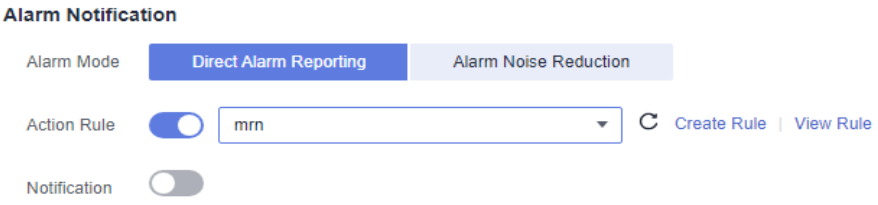
5. Configure alarm notifications and click **Create Now**.

   Alarm Mode: Select **Direct Alarm Reporting**.

   Action Rule: Enable it and select the created action rule. If the existing alarm action rules cannot meet your requirements, click **Create Rule** to create an action rule. For details, see **Creating an Alarm Action Rule**.

   Notification: Enable it.

Figure 1-26 Configuring alarm notifications



Create a topic in SMN to configure alarm notification rules.

- **Creating a topic**

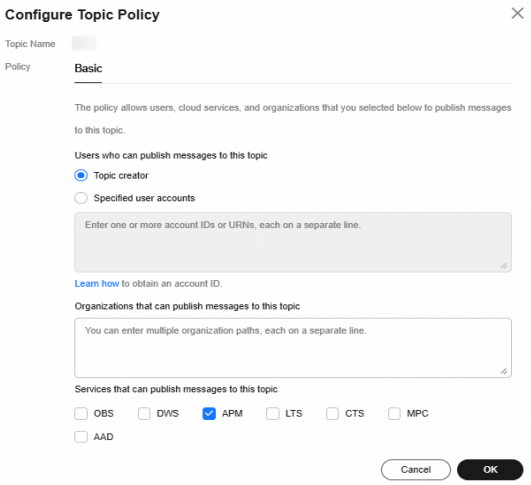
  i. On the displayed **SMN console**, choose **Topic Management** > **Topics** from the navigation pane.

ii. Click **Create Topic**. Enter a topic name, select an enterprise project, and click **OK**.

iii. Locate the target topic and choose **More** > **Configure Topic Policy** in the **Operation** column.

Select **APM** to allow AOM alarms to trigger SMN.

Figure 1-27 Configuring a topic policy

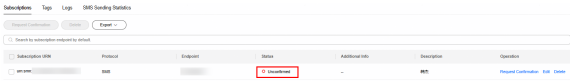


iv. Click **Add Subscription** in the **Operation** column of the topic. After the subscription is successful, a notification is received once the alarm conditions are met.

Select a protocol, such as email or SMS, enter the endpoint, and click **OK**.

A record is displayed in the subscription list, but the record is in the **Unconfirmed** state.

Figure 1-28 Subscriptions



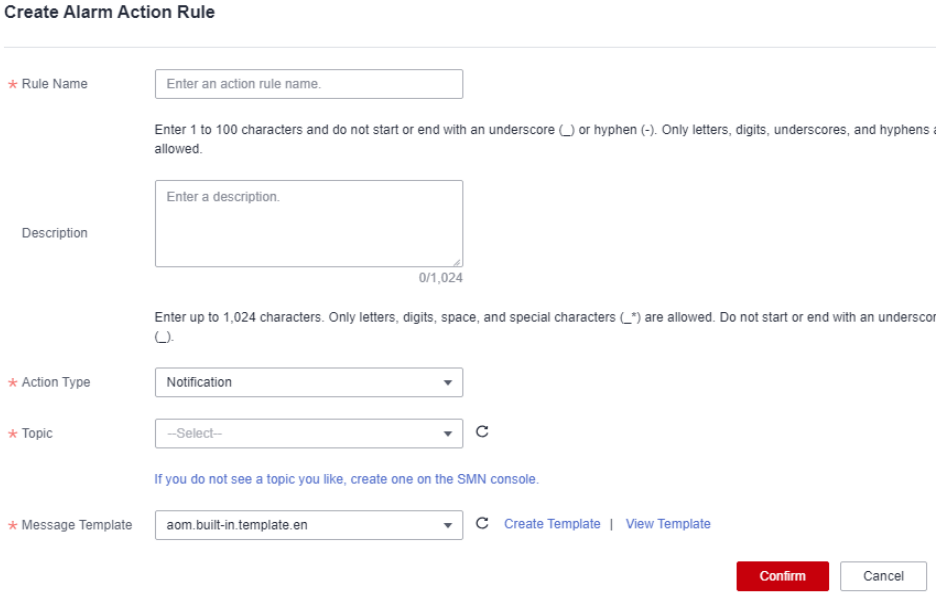
For example, after receiving an email, confirm the subscription. Then, the subscription is in the confirmed state.

- **Creating an alarm action rule**

  An action rule specifies how AOM notifies you when an alarm is triggered. After an alarm action rule is enabled, the system sends notifications based on the associated SMN topic and message template.

  Enter the action rule name, select the action rule type, select the topic created in **the previous step**, select a message template, and click **Confirm**.

Figure 1-29 Creating an alarm action rule



In the **Alarm Notification** area of the **Create Alarm Rule** page, set **Action Rule** to the newly created alarm action rule and click **Create Now**.

After the configuration is complete, you will receive an email or other notifications once the alarm conditions are met.

1.5.7 Starting a Notebook Instance as User root

On the ModelArts platform, the default username is **ma-user** and the default user group is **ma-group**. If you mount an SFS Turbo file system, use a custom image, or directly upload a file, you need to modify the file permissions in advance to ensure that user **ma-user** has the read permission. Otherwise, the **Permission denied** error may occur. The following describes the scenarios and operations for starting a notebook instance as user **root**.

Scenarios

Only dedicated resource pools in the new network mode support starting notebook instances as the **root** user. For details about how to create a dedicated resource pool in the new network mode, see **Creating a Dedicated Resource Pool**.

- Huawei Cloud ModelArts has hardened and optimized the dedicated resource pool network. Only dedicated resource pools in the new network mode support starting notebook instances as the **root** user. Dedicated resource pools in the old network mode do not support this function and will be gradually restricted from October 15, 2025. For details about how to check whether the old network mode is used and the announcement details, see **ModelArts Standard Dedicated Resource Pool Network Adjustment**.
- Public resource pools do not support starting notebook instances as user **root**.

Specifying a User

When creating a notebook instance, you can specify a user in **More Settings**. When starting a notebook instance, ModelArts supports the following two running

user configurations. Supported configurations may vary depending on the resource settings. For details about how to create a notebook instance, see **Creating a Notebook Instance**.

- **ma-user/ma-group**: The default non-privileged user configuration for ModelArts public images (security mode). To use this mode, the following conditions must be met:
  - User: **ma-user** (UID: 1000)
  - User group: **ma-group** (GID: 100)

  Note: If you are using a custom image, you must pre-configure the above user and group in your image; otherwise, the container may fail to start or experience service exceptions due to insufficient permissions. For details about how to add a specific user and user group, see **Dockerfile on a Non-ModelArts Base Image**.
- **root/root**: Runs the notebook instance with the highest privileges. This is suitable for scenarios requiring access to system-level resources but involves potential security risks. When **root/root** is selected, the system forcibly binds the following user and group:
  - User: **root** (UID: 0)
  - User group: **root** (GID: 0)

  Note: Modifying the UID/GID or the associated group for **root** is strictly prohibited, as doing so may cause container permission conflicts or security vulnerabilities.

FAQs

What should I do if the **Specify User** option under **More Settings** cannot be enabled and the message **The selected resource pool does not support specified users** is displayed when I create a notebook instance?

You can resolve this issue by creating a dedicated resource pool based on a new network.

1. Create a network and use it to create a dedicated resource pool. For details, see **Creating a Dedicated Resource Pool**.
2. Use the new dedicated resource pool to create a notebook instance. For details, see **Creating a Notebook Instance**.

1.5.8 Using CTS to Audit Notebook

ModelArts can connect to CTS. With CTS, you can obtain operations associated with ModelArts Notebook for later query, audit, and backtrack operations.

Prerequisites

You have enabled CTS. For details, see **Cloud Trace Service User Guide**.

Key Development Environment Operations Traced by CTS

Table 1-17 Key development environment operations traced by CTS

Operation	Resource Type	Event
Creating a notebook instance	Notebook	createNotebook
Obtaining details about a notebook instance	Notebook	getNotebook
Obtaining all notebook instances	Notebook	listallNotebook
Obtaining notebook instances	Notebook	listNotebook
Opening a CodeLab instance	Notebook	openNotebook
Deleting a notebook instance	Notebook	deleteNotebook
Updating a notebook instance	Notebook	updateNotebook
Starting a notebook instance	Notebook	startNotebook
Stopping a notebook instance	Notebook	stopNotebook
Saving a running instance as a container image	NotebookImage	createNotebookImage
Obtaining supported images	NotebookImage	listNotebookImage
Obtaining user image information	NotebookImageGroups	listNotebookImageGroups
Obtaining image details	NotebookImage	getNotebookImage
Synchronizing image status	NotebookImage	updateNotebookImage
Registering a custom image	NotebookImage	createNotebookImage
Deleting an image	NotebookImage	deleteNotebookImage
Creating a NotebookApp instance	NotebookApp	createNotebookApp
Obtaining NotebookApp instances	NotebookApp	listNotebookApp

Operation	Resource Type	Event
Obtaining details about a NotebookApp instance	NotebookApp	getNotebookApp
Deleting a NotebookApp instance	NotebookApp	deleteNotebookApp
Using a NotebookApp instance	NotebookApp	updateNotebookApp
Obtaining notebook tags in a project	CombineTmsTag	listCombineTmsTag
Obtaining resource tags	NotebookTmsTag	listNotebookTmsTag

Viewing Traces

1. Log in to the **CTS console**.
2. Click  in the upper left corner and select the target region.
3. In the navigation pane on the left, choose **Trace List**.
   Each time you log in to the CTS console, the new edition is displayed by default. Click **Old Edition** in the upper right corner to switch to the trace list of the old edition.
4. Filter traces to view information about the target traces.
   For details, see **Querying Traces in CTS**.

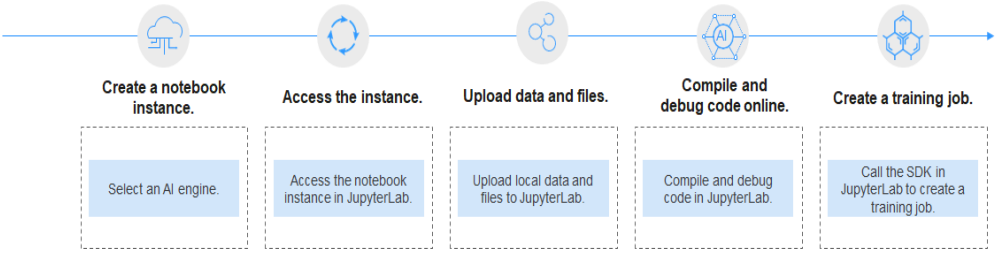
1.6 Using a Notebook Instance for AI Development Through JupyterLab

1.6.1 Using JupyterLab to Develop and Debug Code Online

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

ModelArts allows you to access notebook instances online using JupyterLab and develop AI models based on the PyTorch, TensorFlow, or MindSpore engines. **Figure 1-30** shows the operation process.

Figure 1-30 Using JupyterLab to develop and debug code online



Constraints

- There is no strict limit on the size of files loaded in JupyterLab. However, the actual processing capability is affected by system resources and configurations. If a large text file, for example, larger than 100 MB, is opened in the browser on the left of JupyterLab, the memory may be insufficient, the GUI may respond slowly, or the kernel may be interrupted. To ensure smooth and stable usage, you are advised to open a single file no larger than 100 MB.
- The system can process at most 40 JupyterLab requests (average) per second for a long time or at most 10 JupyterLab requests in a short time. Generally, if more than 10 operations are performed on JupyterLab in the same resource pool within a short period of time, flow control will be triggered.

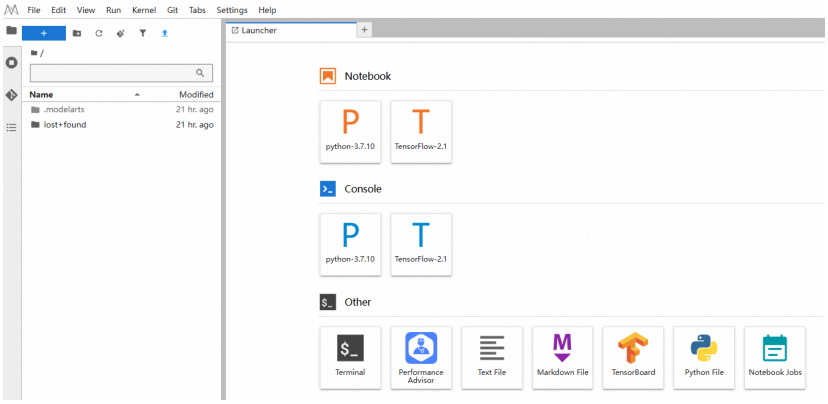
Procedure

1. Create a notebook instance.

   On the ModelArts management console, create a notebook instance with a proper AI engine. For details, see **Creating a Notebook Instance (New Page)**.

2. Locate the created notebook instance, which is in the **Running** state, click **Access Environment** in the **Operation** column. In the displayed dialog box, click **Access** next to **JupyterLab Access**.
3. The **Launcher** page is automatically displayed. Perform required operations. For details, see **JupyterLab Documentation**.

   Figure 1-31 JupyterLab homepage

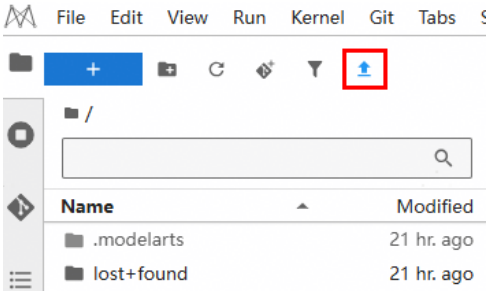
   

   NOTE

   The notebook and console kernels and versions displayed on the **Launcher** page vary depending on the AI engine based on which a notebook instance is created. **Figure 1-31** shows an example only. Obtain the notebook and console kernels and versions on the management console.

4. Upload training data and code files to JupyterLab. For details, see **Uploading Files from a Local Path to JupyterLab**.

Figure 1-32 Button for uploading a file



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

   NOTE

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

6. In JupyterLab, call the ModelArts SDK to create a training job for in-cloud training.

   For details, see **Creating a Training Job**.

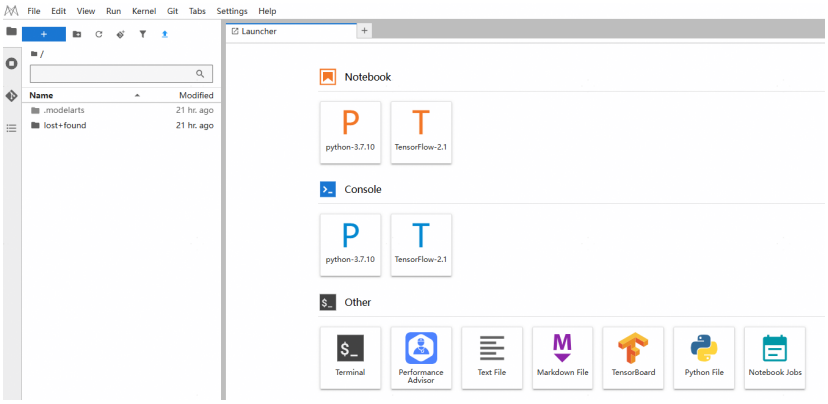
1.6.2 Common Functions of JupyterLab

Introduction

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

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. Locate a running notebook instance and click **Access Environment** in the **Operation** column. In the displayed dialog box, click **Access** next to **JupyterLab Access**.
3. The **Launcher** page is automatically displayed. Perform required operations. For details, see **JupyterLab Documentation**.

Figure 1-33 JupyterLab homepage



NOTE

The notebook and console kernels and versions displayed on the **Launcher** page vary depending on the AI engine based on which a notebook instance is created. **Figure 1-33** shows an example only. Obtain the notebook and console kernels and versions on the management console.

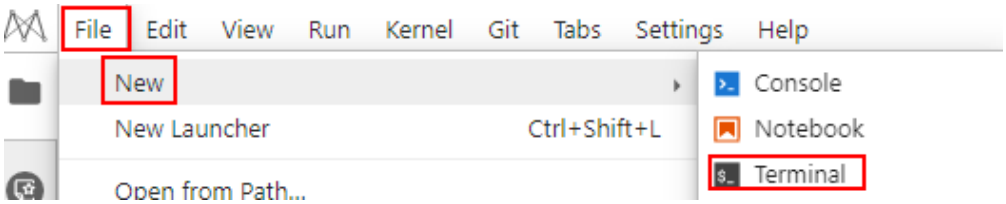
- **Notebook**: Select a kernel for running notebook, for example, TensorFlow or Python.
- **Console**: Call the terminal for command control.
- **Other**: Edit other files.

Creating a Terminal in JupyterLab

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

1. Create a notebook instance. When it is in the **Running** state, locate it in the list and click **Access Environment** in the **Operation** column. In the displayed dialog box, click **Access** next to **JupyterLab Access**. For details, see **Using JupyterLab to Develop and Debug Code Online**.
2. Choose **File** > **New** > **Terminal**. The **Terminal** page is displayed.

Figure 1-34 Terminal



You can view the operation prompt in Terminal. Exercise caution when using plaintext passwords and hard-coded AK/SK. Do not enter sensitive information in commands.

Figure 1-35 Terminal

3. You can use **pip** to install external libraries in the **TensorFlow-1.8** environment on the **Terminal** page. For example, to install Shapely:

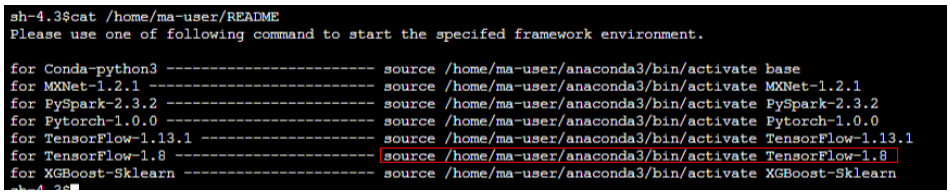
   Enter the following commands in the code input box to obtain the kernel of the current environment and activate the Python environment on which the installation depends:

   cat /home/ma-user/README

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

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

Figure 1-36 Activating the environment



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

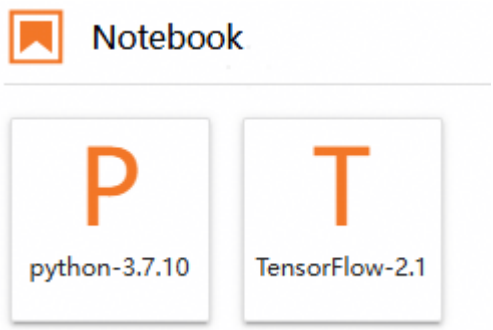
pip install Shapely

Creating an IPYNB File in JupyterLab

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

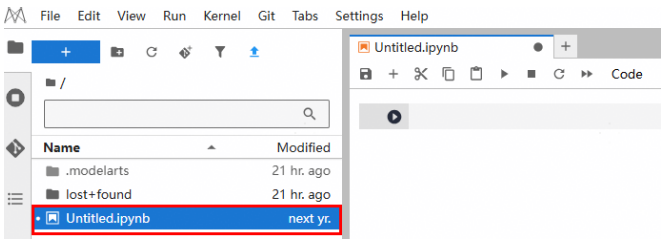
The AI engines supported by each notebook instance vary depending on the runtime environment. The following figure is only an example. Select an AI engine based on site requirements.

Figure 1-37 Selecting an AI engine and creating IPYNB file



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

Figure 1-38 Created IPYNB file



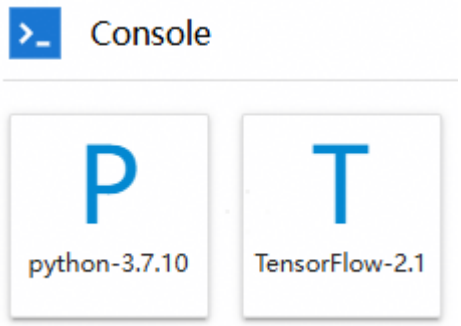
Creating a Notebook File and Accessing the Console

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

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

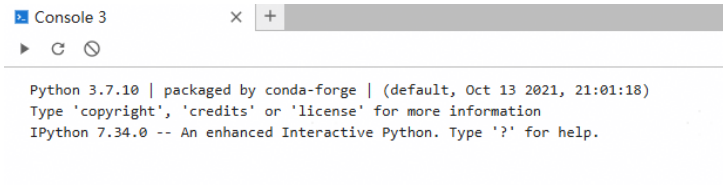
The AI engines supported by each notebook instance vary depending on the runtime environment. The following figure is only an example. Select an AI engine based on site requirements.

Figure 1-39 Selecting an AI engine and creating a console



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

Figure 1-40 Creating a notebook file (console)

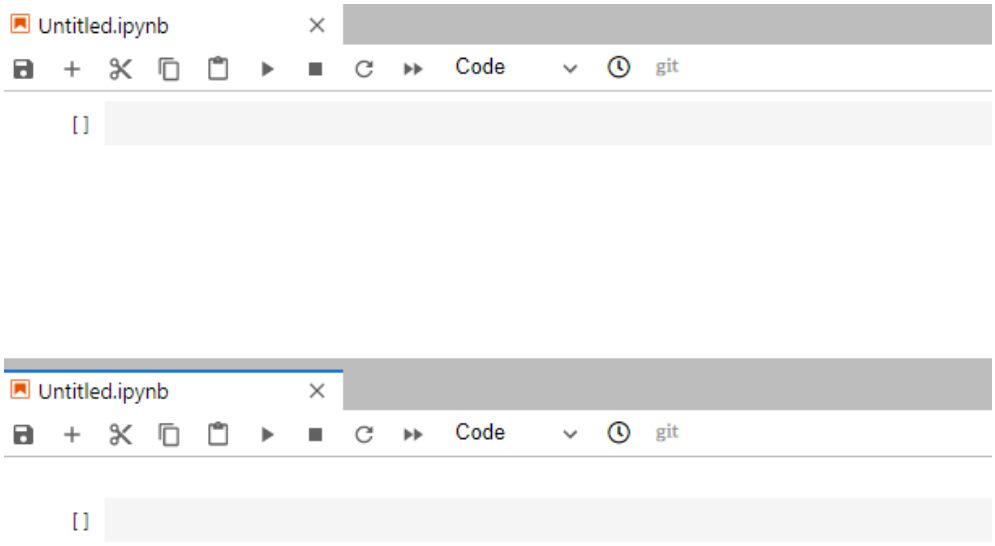


Editing a File in JupyterLab

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

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

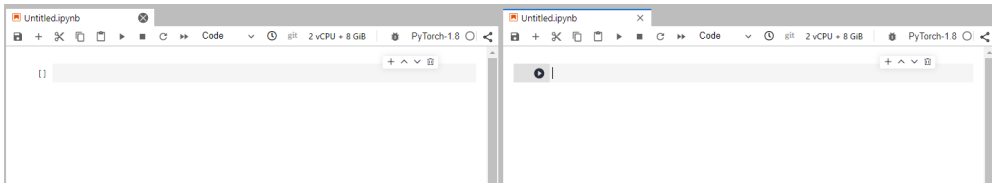
Figure 1-41 Customized display of multiple files



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

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

Figure 1-42 Multiple views of a file



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

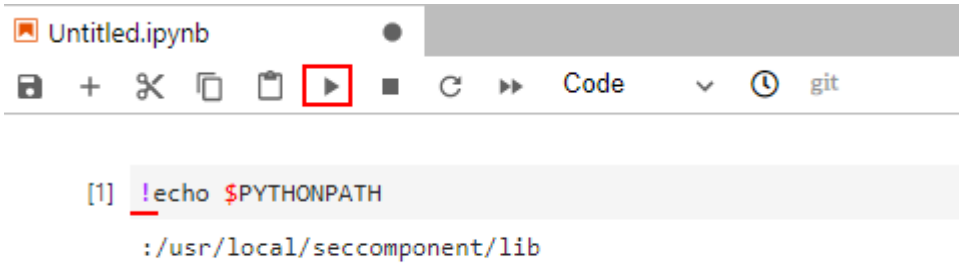
For example, install an external library Shapely.

```
!pip install Shapely
```

For example, obtain PythonPath.

```
!echo $PYTHONPATH
```


Figure 1-43 Running code

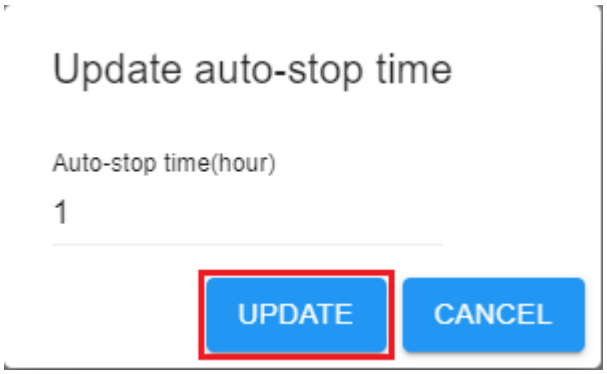


Renewing or Automatically Stopping a Notebook Instance

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

Figure 1-44 Remaining duration

Figure 1-45 Renewing an instance



Common JupyterLab Buttons and Plug-ins

Figure 1-46 Common JupyterLab buttons and plug-ins

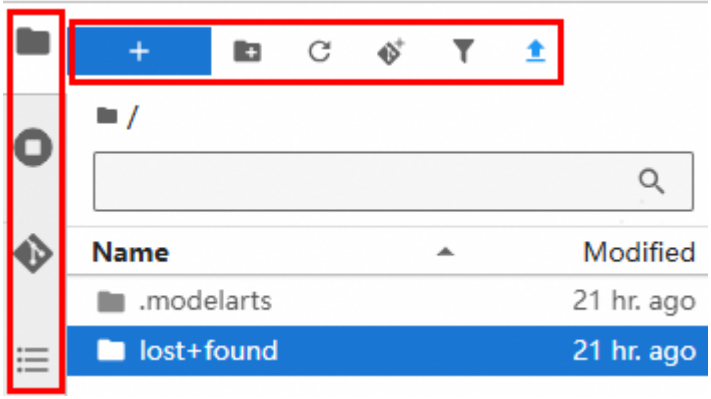


Table 1-18 JupyterLab buttons

Button	Description
+	Quickly open notebook instances and terminals. Open the **Launcher** page, on which you can quickly create notebook instances, consoles, or other files.
	Create a folder.
	Refresh the file directory.
	Git plug-in, which can be used to access the GitHub code library associated with the notebook instance.
	Switch the file filter.
	Upload files.

Table 1-19 JupyterLab plug-ins

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

Plug-in	Description
	Show the document organization.

Figure 1-47 Buttons in the navigation bar

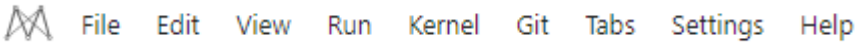


Table 1-20 Buttons in the navigation bar

Button	Description
File	Actions related to files and directories, such as creating, closing, or saving notebooks.
Edit	Actions related to editing documents and other activities in the IPYNB file, such as undoing, redoing, or cutting cells.
View	Actions that alter the appearance of JupyterLab, such as showing the bar or expanding code.
Run	Actions for running code in different activities such as notebooks and code consoles.
Kernel	Actions for managing kernels, such as interrupting, restarting, or shutting down a kernel.
Git	Actions on the Git plug-in, which can be used to quickly access the GitHub code library.
Tabs	A list of the open documents and activities in the dock panel.
Settings	Common settings and an advanced settings editor.
Help	A list of JupyterLab and kernel help links.

Figure 1-48 Buttons in the menu bar of an IPYNB file

Table 1-21 Buttons in the menu bar of an IPYNB file

Button	Description
	Save a file.

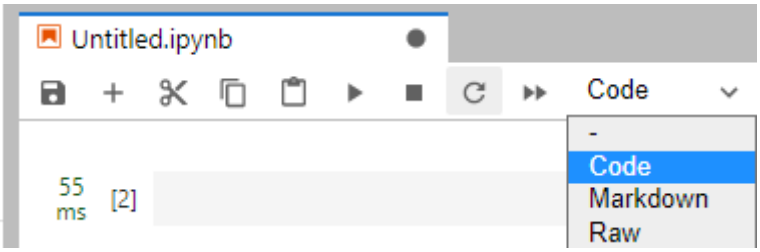
Button	Description
+	Add a new cell.
✂	Cut the selected cell.
⧉	Copy the selected cell.
📋	Paste the selected cell.
▶	Execute the selected cell.
■	Terminate a kernel.
↻	Restart a kernel.
▶▶	Restart a kernel and run all code of the current notebook again.
Code ⌄	There are four options in the drop-down list: **Code** (Python code), **Markdown** (Markdown code, typically used for comments), **Raw** (a conversion tool), and **-** (not modified)
🕒	View historical code versions.
git	Git plug-in. The gray button indicates that the plug-in is unavailable in the current region.
2 vCPU + 4 GiB	Instance flavor.
PyTorch-1.4	Kernel for you to select.
○	Code running status. ● indicates the code is being executed.

Using Code-based Plug-ins

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

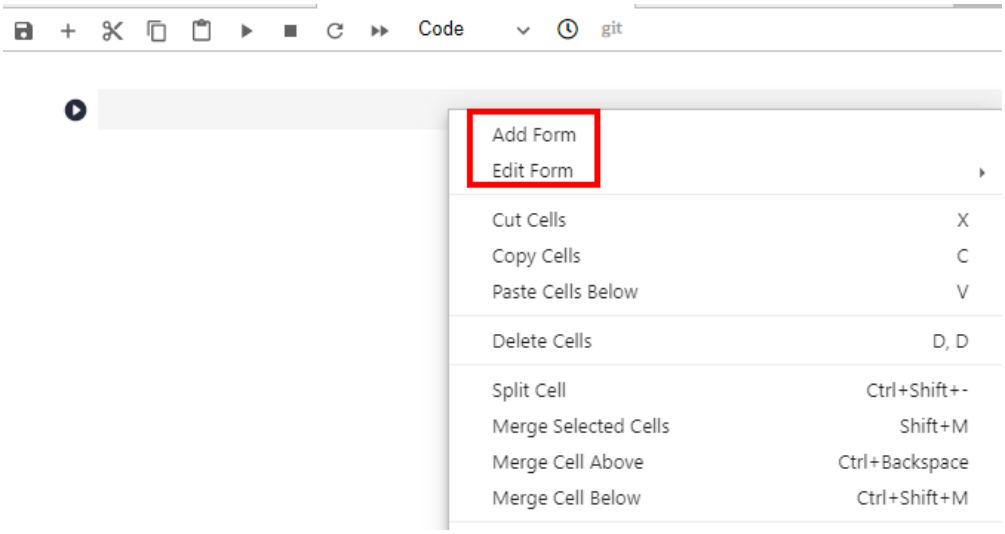
- The **Add Form** and **Edit Form** buttons are available only to the shortcut menu of code cells, as shown below.

Figure 1-49 Viewing a code cell



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

Figure 1-50 Shortcut menu of code cells



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

Figure 1-51 Two edit areas

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

Table 1-22 Edit Form sub-options

Edit-Form Sub-option	Description
Add new form field	• The form field types include **dropdown**, **input**, and **slider**, Each time a field is added, as shown in **Figure 1-52**, the corresponding variable is added to the code and form areas. If a value in the form area is changed, the corresponding variable in the code area is also changed. **NOTE** When creating a dropdown form, click **ADD Item** and add at least two items, as shown in **Figure 1-53**. • If the form field type is set to **dropdown**, the supported variable types are **raw** and **string**. • If the form field type is set to **input**, the supported variable types are **boolean**, **date**, **integer**, **number**, **raw**, and **string**. • If the form field type is set to **slider**, the minimum value, maximum value, and step can be set.
Hide code	Hide the code.
Hide form	Hide the forms.
Show all	Display both code and forms.

Figure 1-52 Form field types

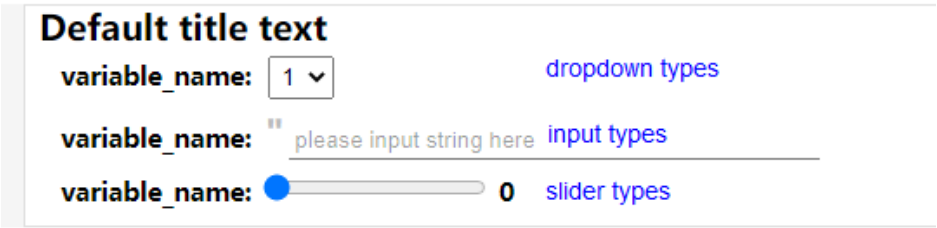


Figure 1-53 Creating a dropdown form

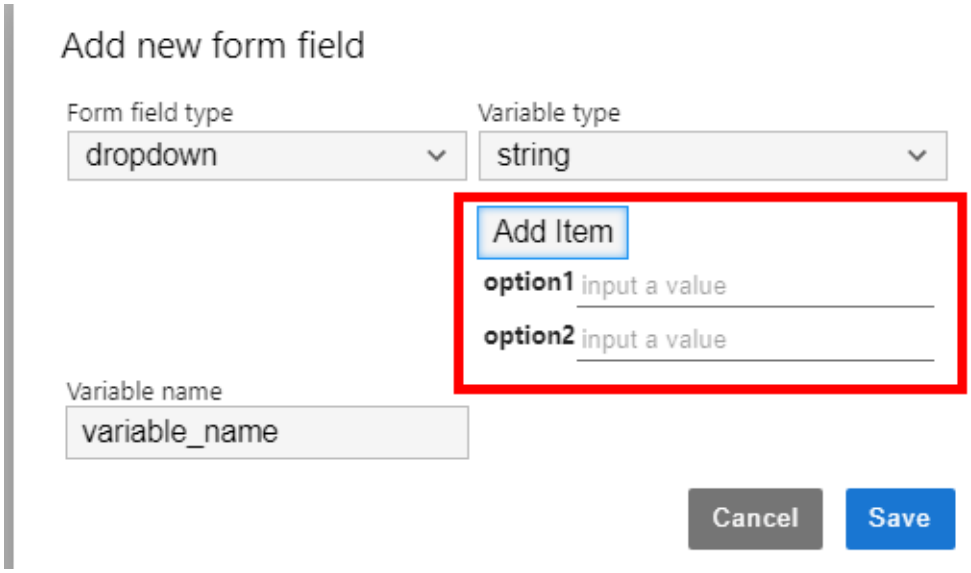
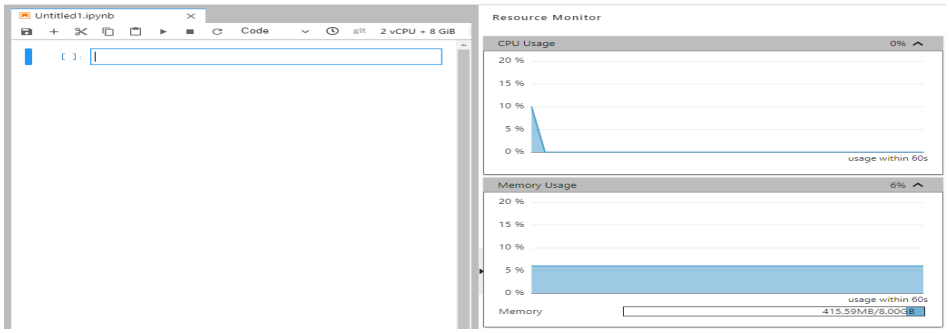


Figure 1-54 Deleting a form

Monitoring Resources

To obtain resource usage, select **Resource Monitor** in the right pane. The CPU usage and memory usage are displayed.

Figure 1-55 Resource usage



1.6.3 Upgrading JupyterLab

To enhance the usability of notebook, JupyterLab has been upgraded to version 4.4.10 (stable version in 2025). This major leap from the 2021 legacy version (3.2.3) significantly optimizes performance and startup speed. Key improvements include a rebuilt editor, enhanced debugging and collaboration tools, a more

stable ecosystem, and comprehensive refinements to the interactive experience and memory efficiency. Hope you enjoy the upgrade.

Notes

Support for creating new notebook instances using JupyterLab 3.2.3 will be discontinued in April 2026. Additionally, technical support, including updates for new features, vulnerability/issue fixes, patch upgrades, service ticket guidance, and online troubleshooting, will no longer be provided. These services will no longer be applicable to the O&M assurance of ModelArts. Therefore, upgrade to JupyterLab 4 to experience the latest features.

JupyterLab 4

This version has significantly improved user experience, functions, and performance. The following table lists the main updates. For more information, see **JupyterLab Documentation**.

Table 1-23 Feature updates

Function Module	Description
Workspaces	You can create multiple workspaces to organize and manage different projects and files, improving project management efficiency.
Launcher creation toolbar	Added a toolbar for function access and convenient operations.
Theme framework optimization	Optimized the theme framework to enhance the theme flexibility and compatibility. You can customize the GUI as required.
Configuration page optimization	Optimized user experience on the configuration page for easier use.
Performance optimization and debugging enhancement	Fully optimized the system performance and enhanced the debugging function to improve the development experience.
Code editor enhancement	Improved the code editor by adding new functions and optimizing performance to enhance the coding experience.
Search function enhancement	Enhanced the search function for quicker and more accurate search.
Performance enhancement	Improved the overall performance, system response speed, and system stability.
Custom CSS style sheets	Custom CSS style sheets can be used to adjust the GUI appearance to meet personalized requirements.

Function Module	Description
Markdown charts	Charts can be used in Markdown to facilitate document writing.
Virtual scroll bar	Introduced the virtual scroll bar to improve the scrolling experience of large files or a large amount of content.
Workspace UI	Improved the workspace UI to provide better visual effect and operation experience and enhance UI friendliness.
File access records	You can view recently opened and closed files to quickly access common files and improve work efficiency.
Keyboard shortcuts improvement	Keyboard shortcuts are improved for higher operation efficiency and convenience.

Method 1: Stopping a Notebook Instance and Upgrading JupyterLab

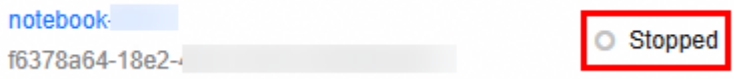
After a notebook is stopped, only the data stored in **/home/ma-user/work** is retained. To save the development environment, save its settings as an image. For details, see **Method 2: Saving a Notebook Instance and Upgrading JupyterLab**.

If your notebook uses EVS for storage, EVS billing per GB continues even after stopping the instance until you delete it. For details about the billing, see **Billing Item > Development Environment**.

1. Stop the notebook instance.
   a. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
   b. On the **Notebook** page, click **Stop** in the **Operation** column of the target instance.
   c. In the displayed dialog box, read the message and click **OK**.

      When the status of the notebook instance is **Stopped**, the notebook instance has been stopped.

      Figure 1-56 Notebook instance status

2. Upgrade JupyterLab.
   a. On the **Notebook** page, click **Start** in the **Operation** column of the stopped instance.
   b. In the displayed dialog box, select JupyterLab 4, configure **Auto Stop**, and click **OK**.
3. Check whether JupyterLab is upgraded.

On the **Notebook** page, click the instance name. If JupyterLab 4 is displayed on the instance details page, JupyterLab has been upgraded.

Method 2: Saving a Notebook Instance and Upgrading JupyterLab

After the image is saved, the default working directory is the root directory **/**. Any installed dependency packages will be retained in the saved image. However, content in persistent storage (the **/home/ma-user/work** directory) will not be included in the final container image. In VS Code remote development scenarios, plugins installed on the server side will also be retained.

1. Save the image. For details, see **Saving a Notebook Instance**.
2. Upgrade JupyterLab.
   a. In the navigation pane on the left, choose **Asset Management** > **Image Management**.
   b. In the **Registered Images** tab, click the name of the image saved in **step 1**.
   c. In the **Operation** column, click **Create Notebook**.
   d. On the displayed page, select JupyterLab 4, modify other parameters as required, and click **Next**.
      When the notebook instance is in the **Running** state, the notebook instance has been created.
3. Check whether JupyterLab is upgraded.
   On the **Notebook** page, click the instance name. If JupyterLab 4 is displayed on the instance details page, JupyterLab has been upgraded.

1.6.4 Using Git to Clone the Code Repository in JupyterLab

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

Prerequisites

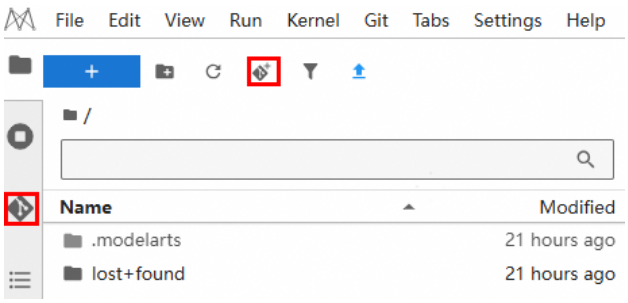
The notebook instance is running.

Starting the Git Plug-in of JupyterLab

Locate a notebook instance in the list and click **Access Environment** in the **Operation** column. In the displayed dialog box, click **Access** next to **JupyterLab Access**.

Figure 1-57 shows the Git plug-in of JupyterLab.

Figure 1-57 Git plug-in

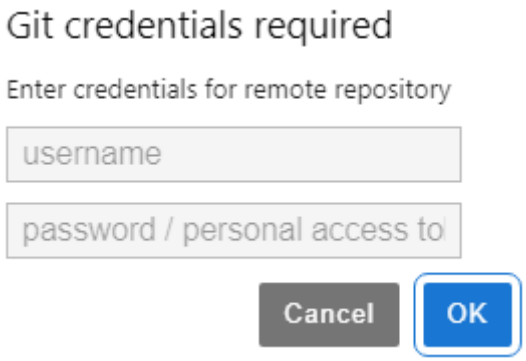


Cloning a GitHub Open-Source Code Repository

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

Cloning a GitHub Private Code Repository

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



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

NOTICE

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

Figure 1-58 Cloning a GitHub private code repository (only authorization using a personal access token is supported)

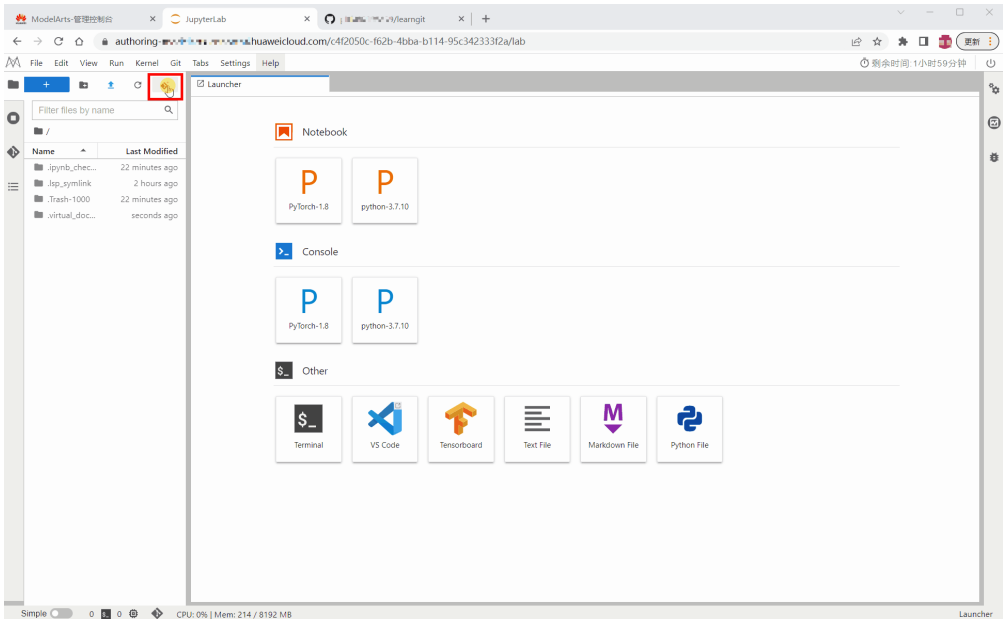
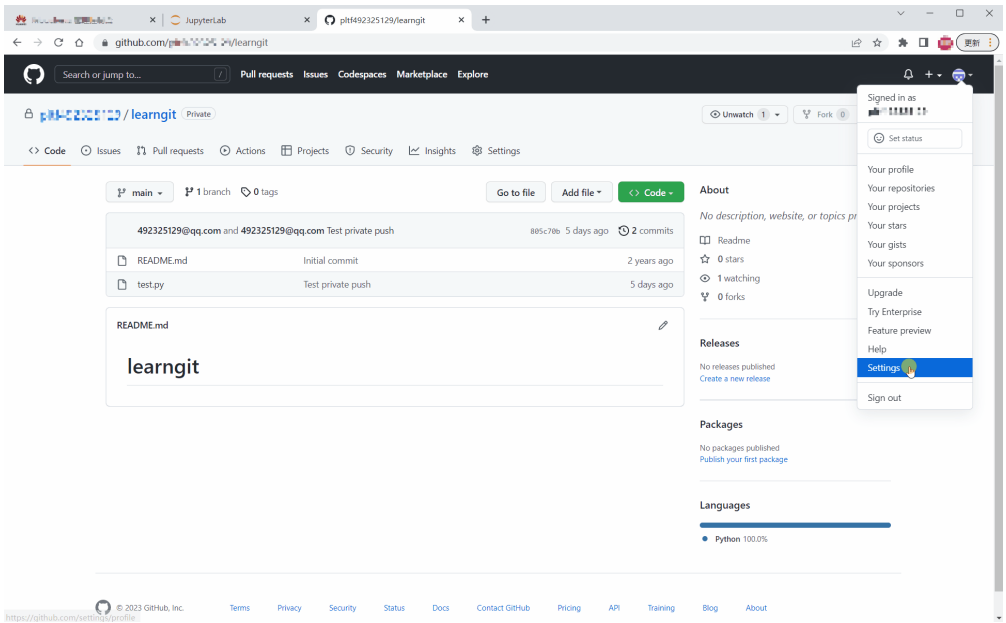


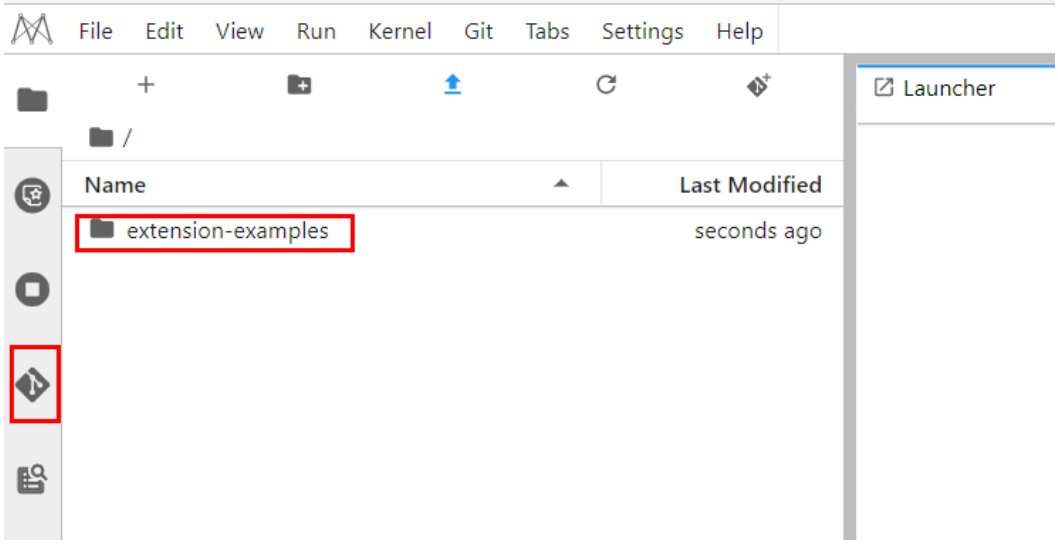
Figure 1-59 Obtaining a personal access token



Viewing a Code Repository

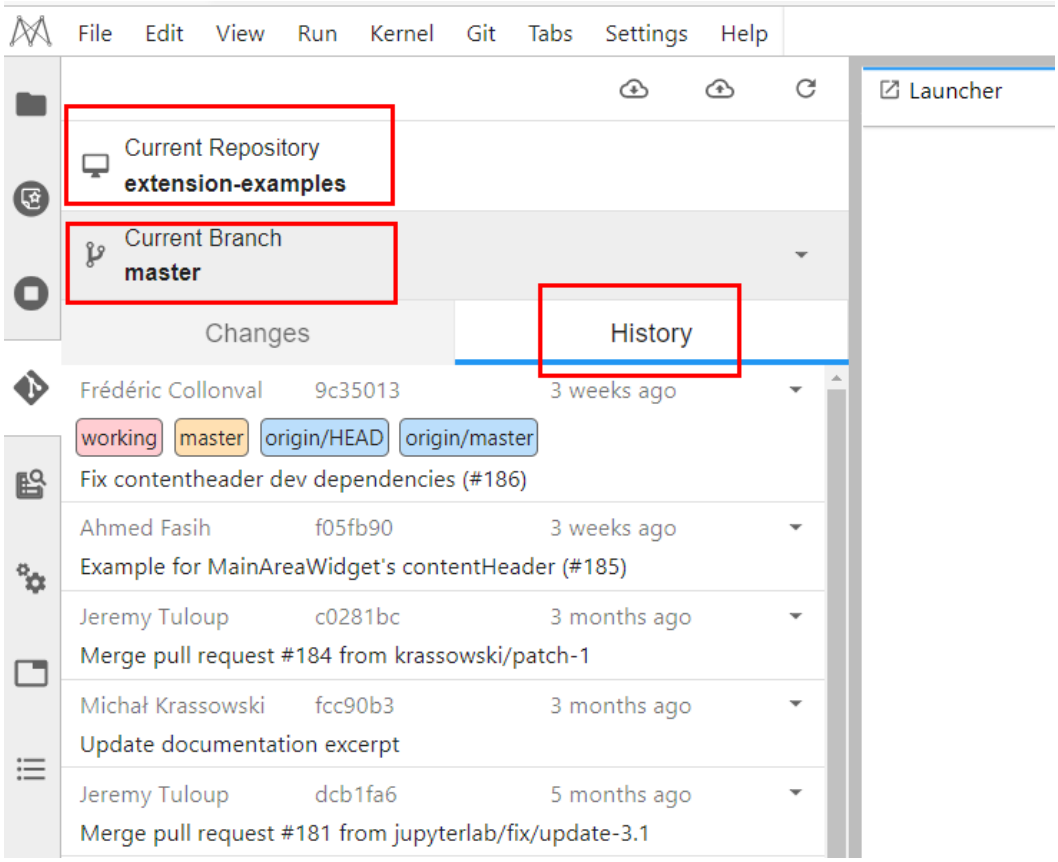
In the list under **Name**, double-click the folder you want to use and click the Git plug-in icon on the left to access the code repository corresponding to the folder.

Figure 1-60 Opening the folder and starting the Git plug-in



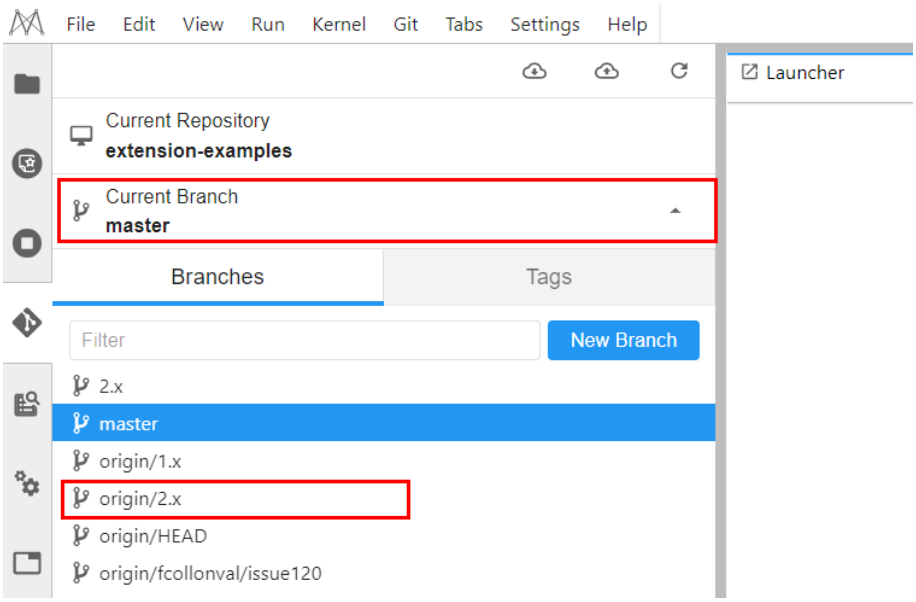
You can view the information current code repository, such as the repository name, branch, and historical submission records.

Figure 1-61 Viewing a code repository



NOTE

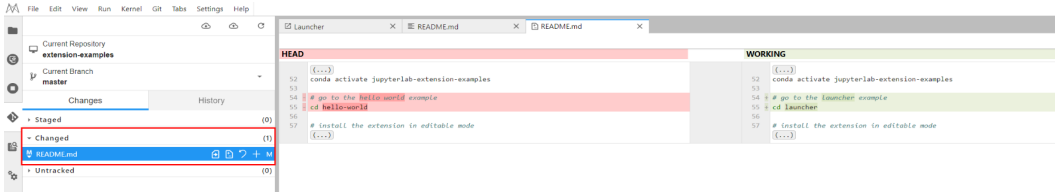
By default, the Git plug-in clones the master branch. To switch another branch, click **Current Branch** to expand all branches and click the target branch name.



Viewing Modifications

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

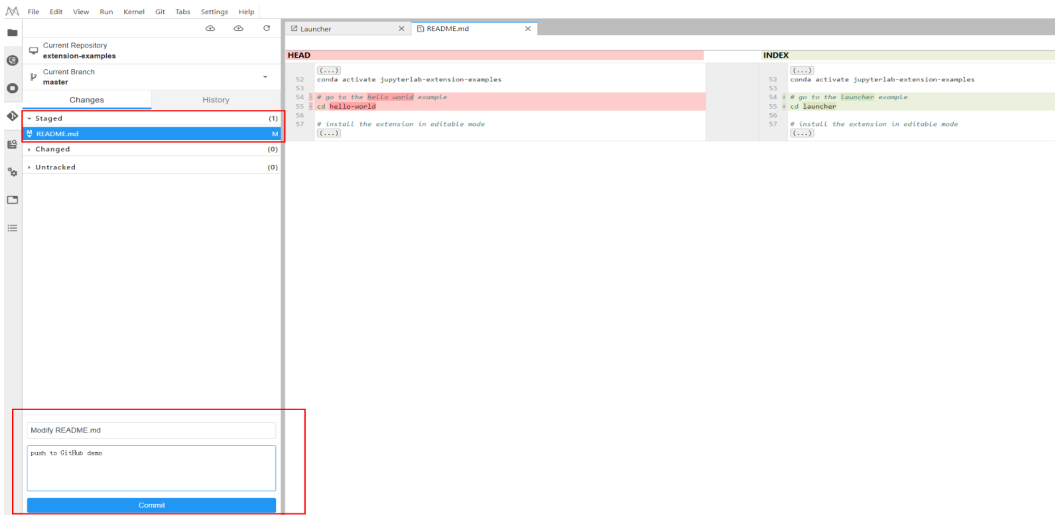
Figure 1-62 Viewing modifications



Committing Modifications

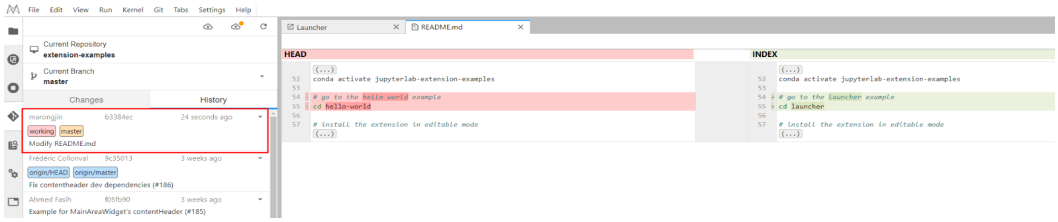
After confirming that the modifications are correct, click **Stage this change** on the right of the file name, which is equivalent to running the **git add** command. The file enters the **Staged** state. Enter the message to be committed in the lower left corner and click **Commit** that is equivalent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

Figure 1-63 Committing modifications



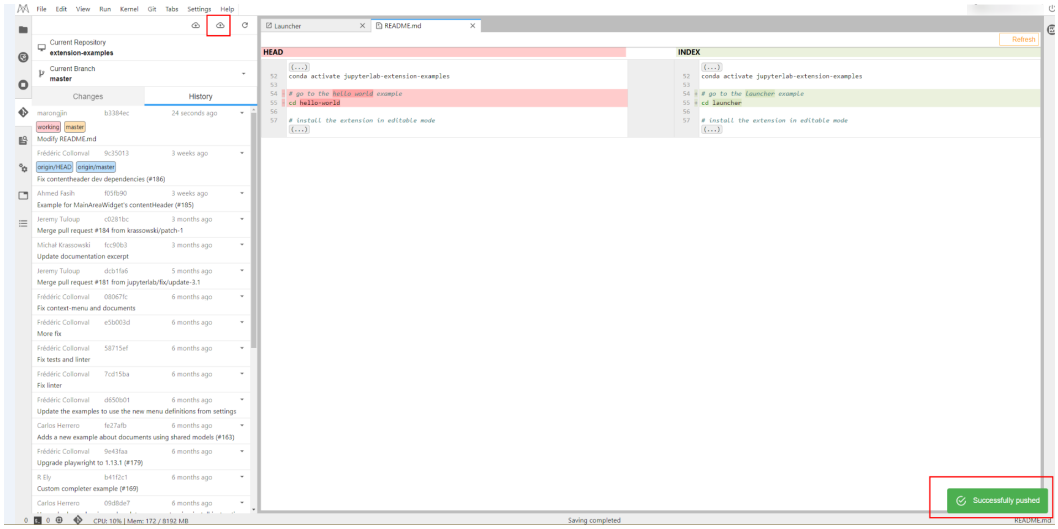
In the **History** tab, view the committing status.

Figure 1-64 Checking whether the committing is successful



Click the **push** icon, which is equivalent to running the **git push** command, to push the code to the GitHub repository. After the pushing is successful, the message "Successfully completed" is displayed. If the token used for OAuth authentication has expired, a dialog box is displayed asking you to enter the user token or account information. Enter the information as prompted. This section describes the authorization using a personal access token. If you use a password for authorization but the password becomes invalid, perform the operations in **What Do I Do If the Git Plug-in Password Is Invalid?**

Figure 1-65 Pushing code to the GitHub repository



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

1.6.5 Creating a Scheduled Job in JupyterLab

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

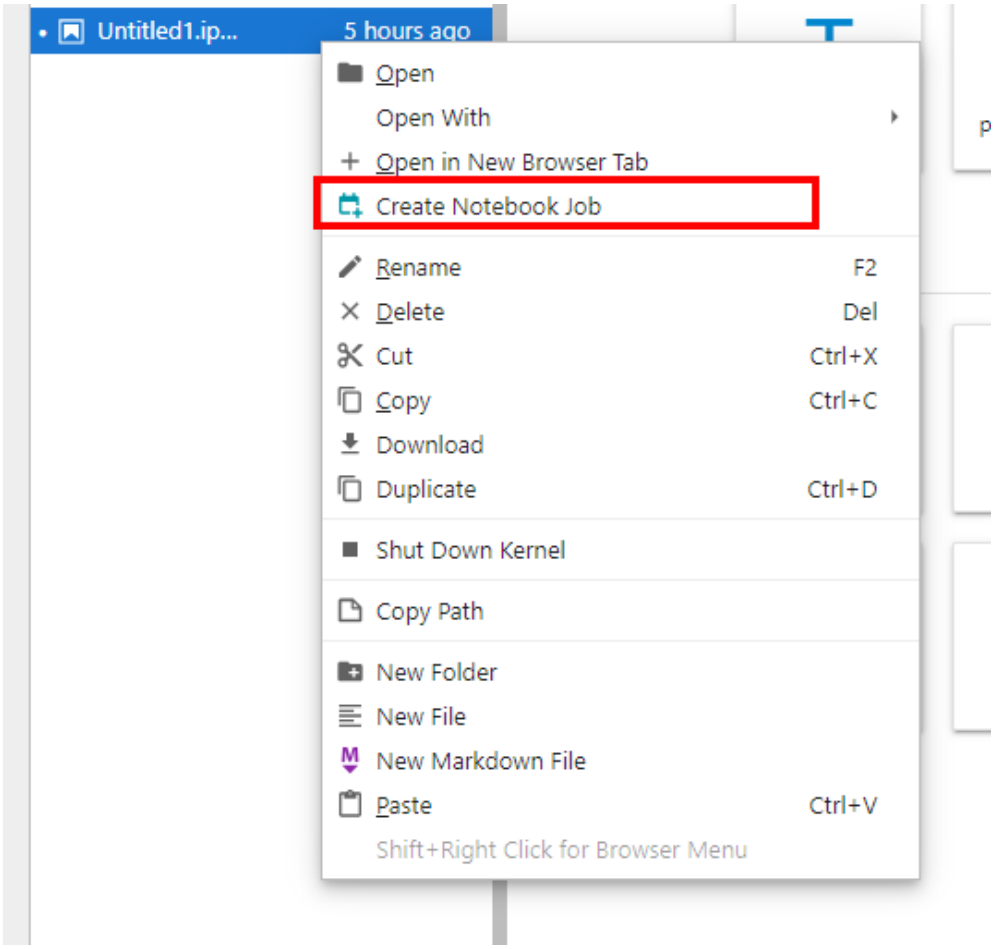
Highlights

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

Procedure

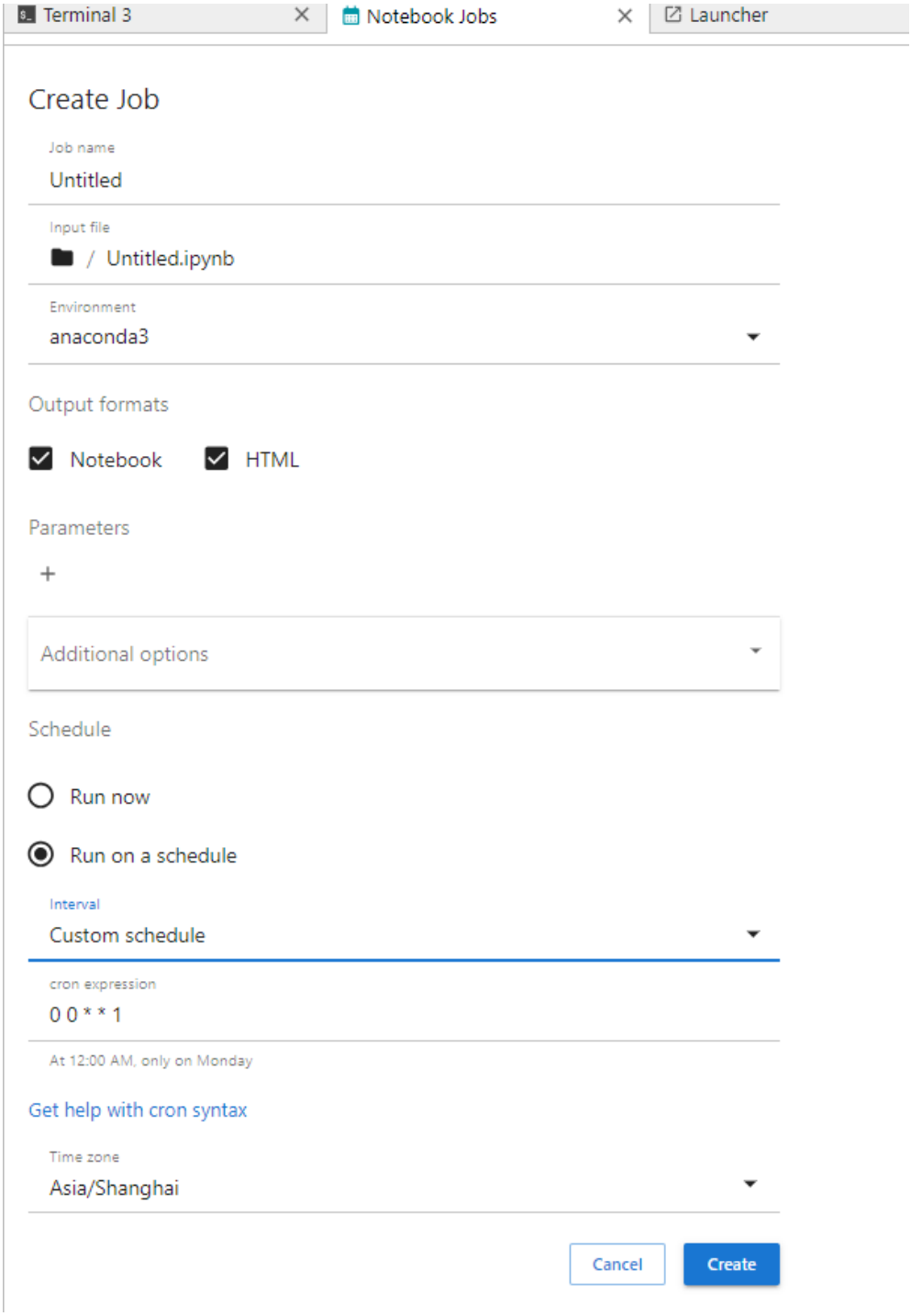
1. Open ModelArts Notebook.
2. Choose a notebook file (IPYNV file) to create a scheduled job.

Figure 1-66 Opening notebook jobs



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

Figure 1-67 Parameters for creating a scheduled job



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

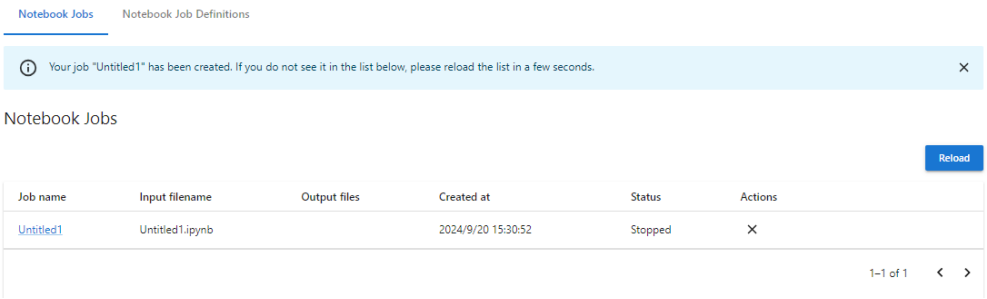
- **Schedule**: Choose **Run now** or **Run on a schedule**. The cron expression is supported.

  NOTE

  - The cron expression must be in the format supported by Linux OS. Otherwise, an error will be reported. If the expression contains a question mark (?), replace it with an asterisk (*).
  - After a scheduled job is configured, modify the file name and content. The created jobs are not affected.

4. Run the job. You can view the job running records in the **Notebook Jobs** tab. Click **Reload** in the upper right corner to get the latest records.

   Figure 1-68 Viewing the running records of scheduled jobs



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

   Figure 1-69 Viewing the execution result of a scheduled job

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

   Figure 1-70 Notebook Job Definitions tab

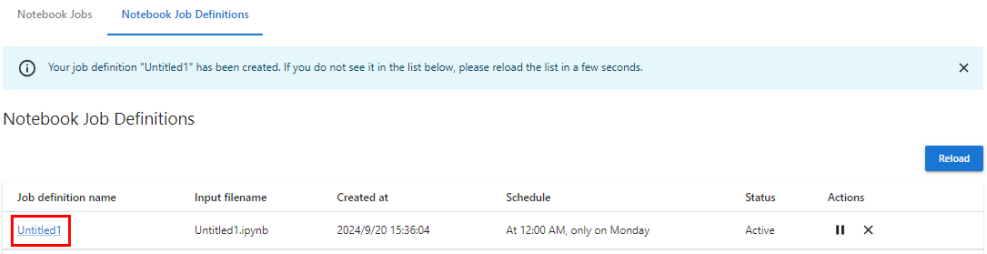
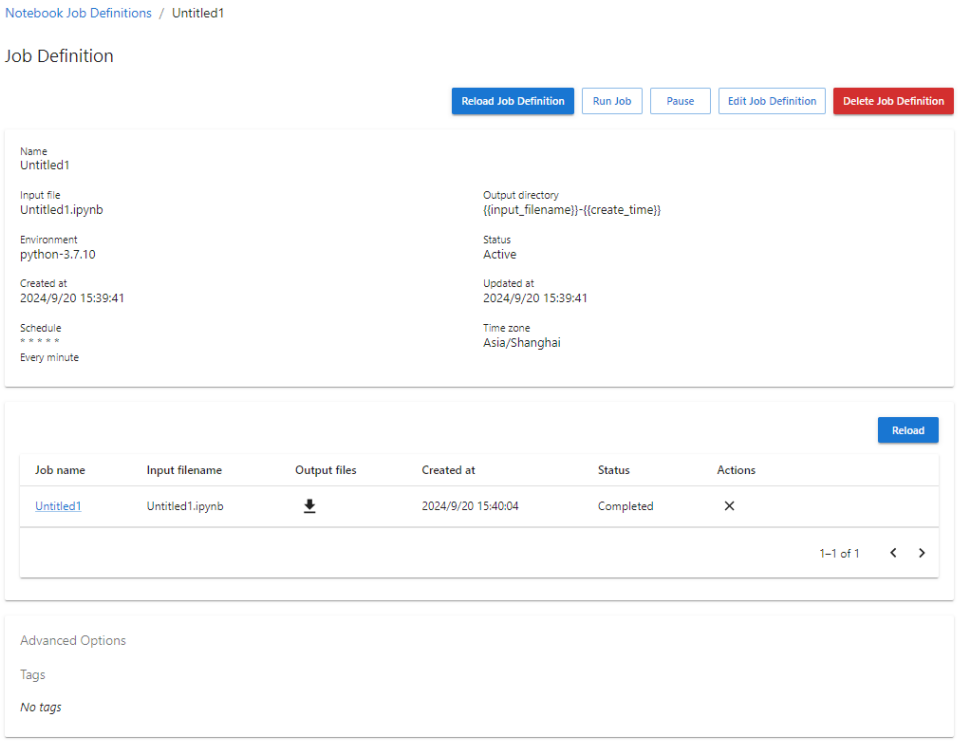


Figure 1-71 Configuring a scheduled job



1.6.6 Uploading Files to JupyterLab

1.6.6.1 Uploading Files from a Local Path to JupyterLab

JupyterLab provides multiple methods for uploading files.

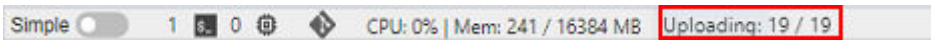
Methods for Uploading a File

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

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

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

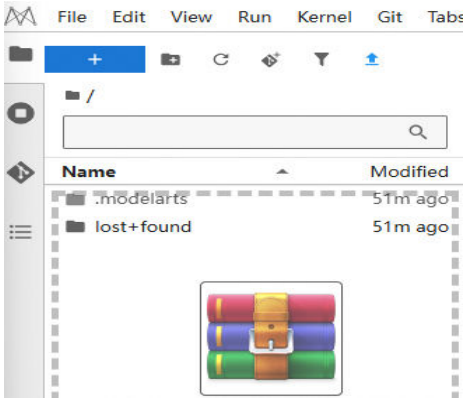
- When multiple files are uploaded in a batch, the total number of files to be uploaded and the number of files that have been uploaded are displayed at the bottom of the JupyterLab window.



Uploading File

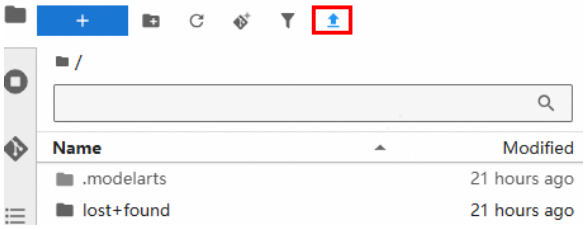
Method 1: Use JupyterLab to open a running notebook environment. Drag the file to JupyterLab.

Figure 1-72 Dragging a file to JupyterLab



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

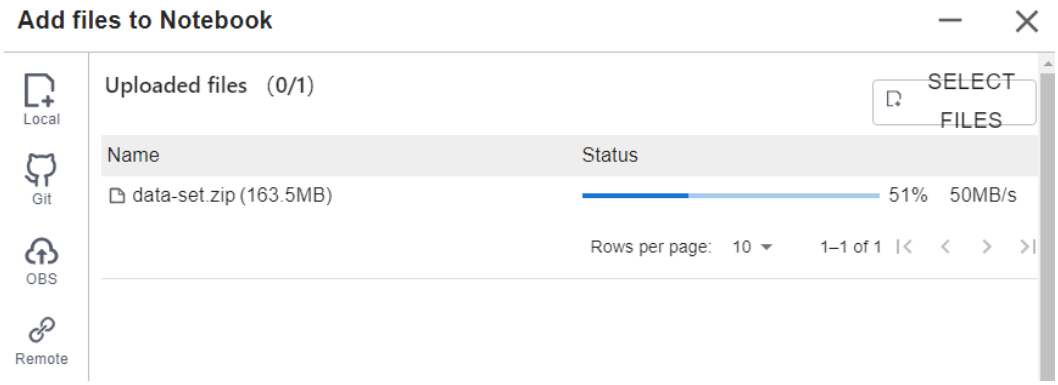
Figure 1-73 Clicking the upload button



Uploading a Local File Smaller Than 100 MB to JupyterLab

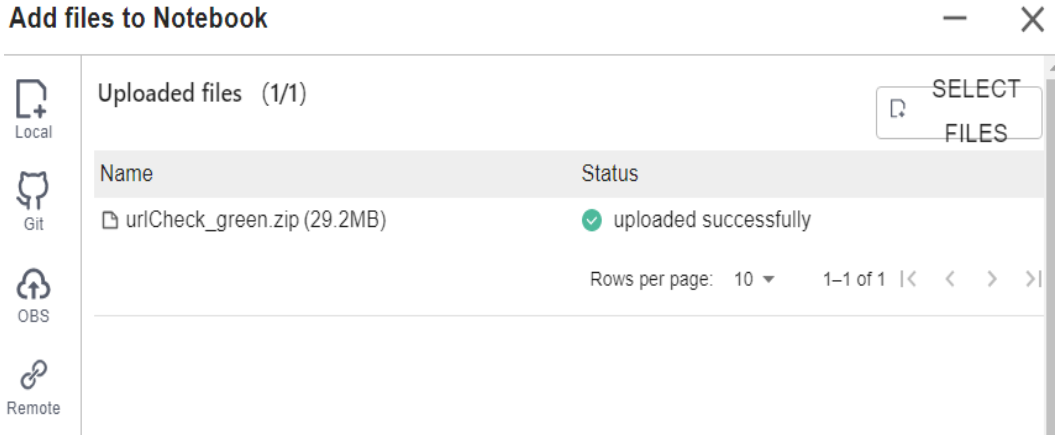
For a file not exceeding 100 MB, directly upload it to the target notebook instance. Detailed information, such as the file size, upload progress, and upload speed are displayed.

Figure 1-74 Uploading a file less than 100 MB



A message is displayed after the file is uploaded.

Figure 1-75 Uploaded

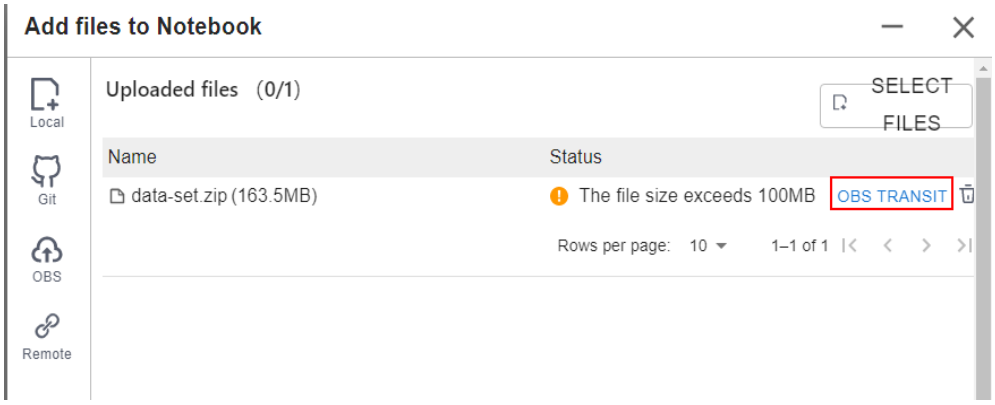


Uploading a Local File that Is 100 MB to 50 GB to JupyterLab

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

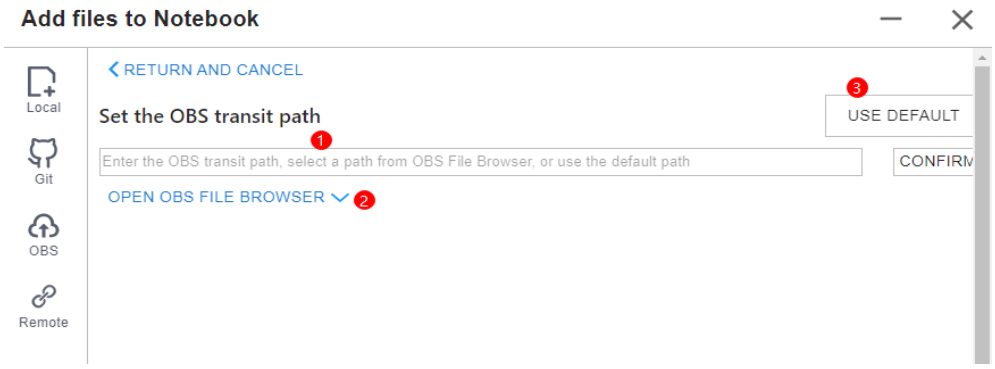
For example, in the scenario shown in the following figure, upload the file through OBS.

Figure 1-76 Uploading a large file through OBS



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

Figure 1-77 Uploading a file through OBS

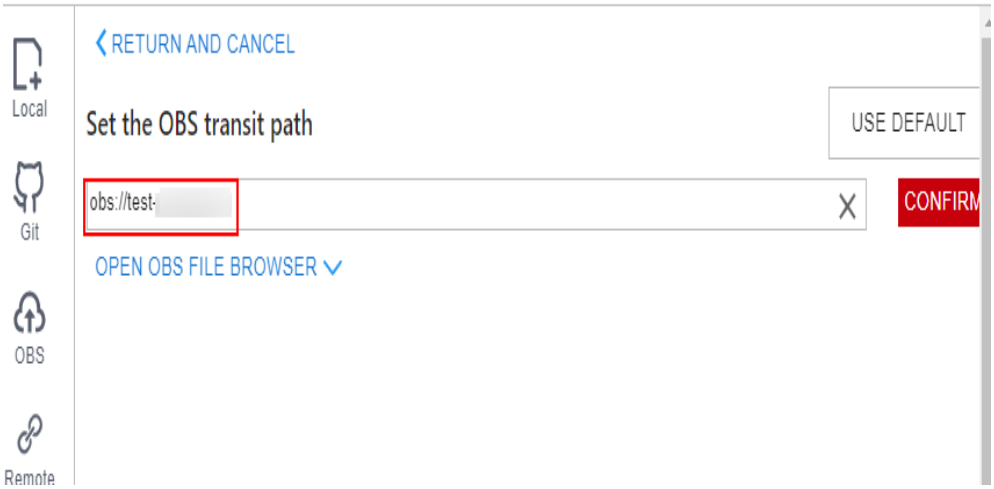


NOTE

Set an OBS path for uploading local files to JupyterLab. After the setting, this path is used by default in follow-up operations. To change the path, click  in the lower left corner of the file upload window, as shown in **Figure 1-81**.

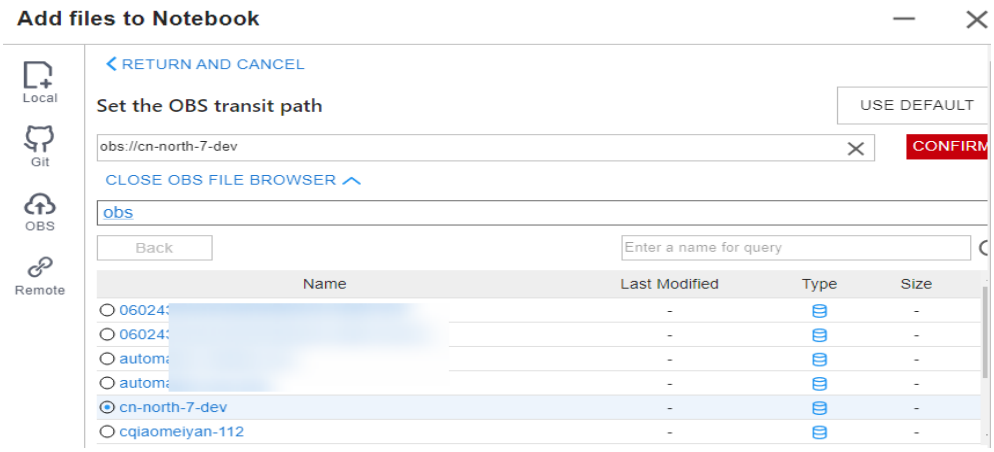
- Method 1: Enter a valid OBS path in the text box and click **OK**.

Figure 1-78 Configuring an OBS path



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

Figure 1-79 OBS File Browser



- Method 3: Use the default path.

Figure 1-80 Using the default path to upload a file

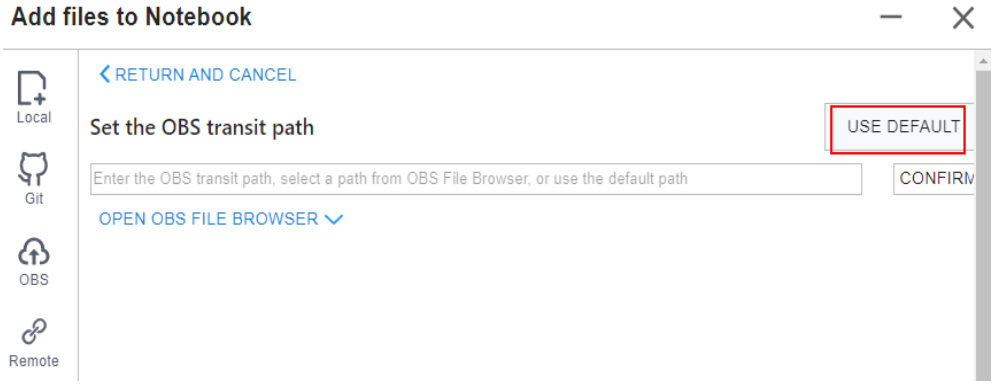
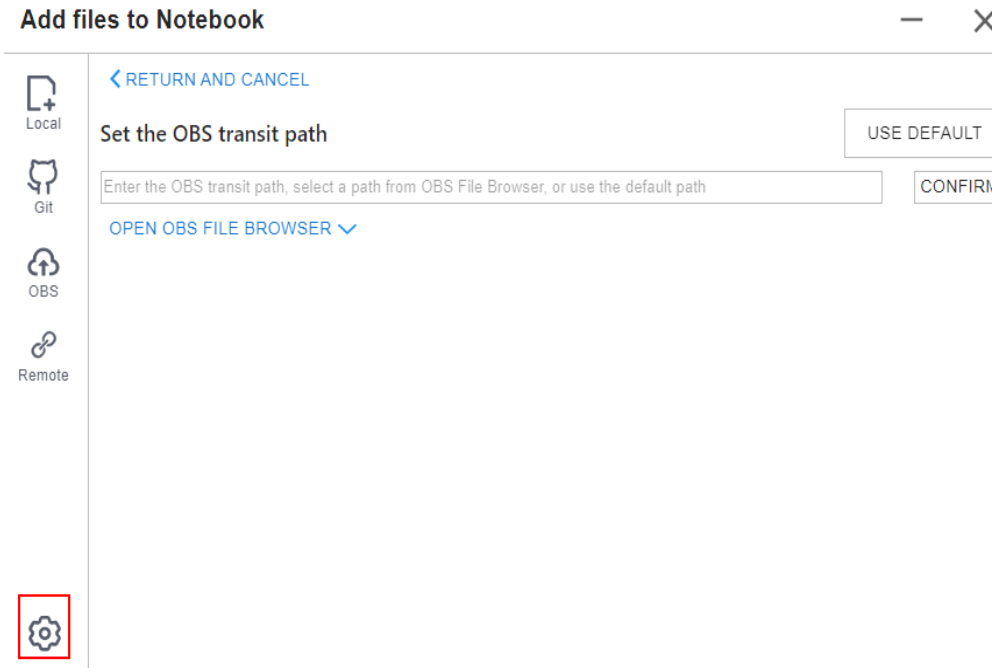
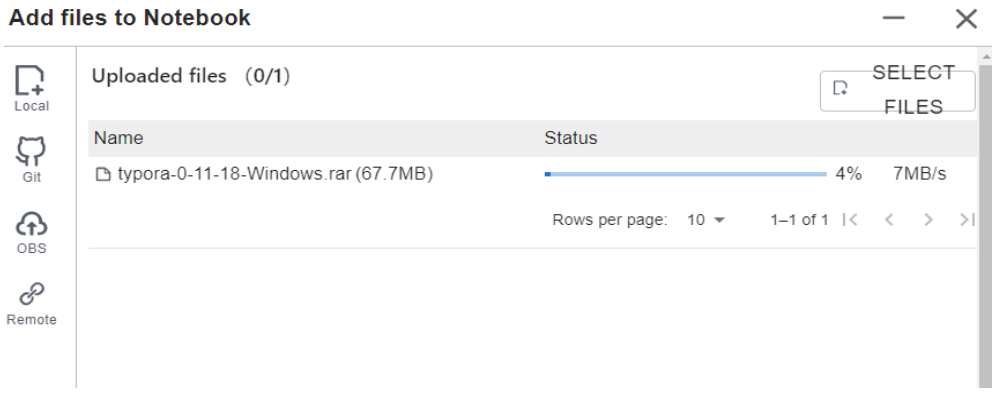


Figure 1-81 Setting an OBS path to upload a local file



After the OBS path is set, upload a file.

Figure 1-82 Uploading a file



Decompressing a package

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

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

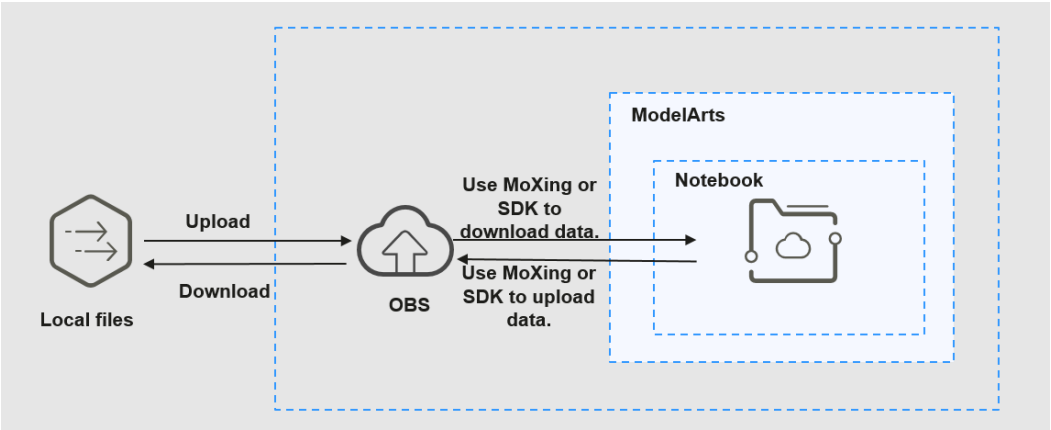
For more details, search for the Linux decompression command in mainstream search engines.

Uploading a Local File Larger Than 50 GB to JupyterLab

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

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

Figure 1-83 Uploading and downloading large files in a notebook instance



The procedure is as follows:

1. Upload the file from a local path to OBS. For details, see **Using OBS Console**.
2. Download the file from OBS to the notebook instance by calling the ModelArts SDK or MoXing API.
   - Method 1: Call the ModelArts SDK to download a file from OBS.

     Example code:

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

   - Method 2: Call the ModelArts MoXing API for reading an OBS file.

     ```
     import moxing as mox

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

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

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

     After the code is executed, open the terminal shown in **Figure 1-84** and run the **ls /home/ma-user/work** command to view the file downloaded to the notebook instance. Alternatively, go to JupyterLab. In the navigation pane on the left, view the downloaded file. If the file is not displayed, refresh the page, as shown in **Figure 1-85**.

     Figure 1-84 Terminal

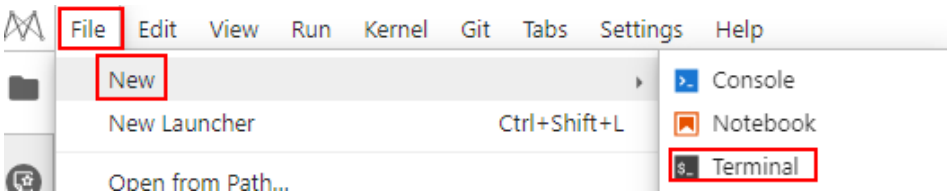
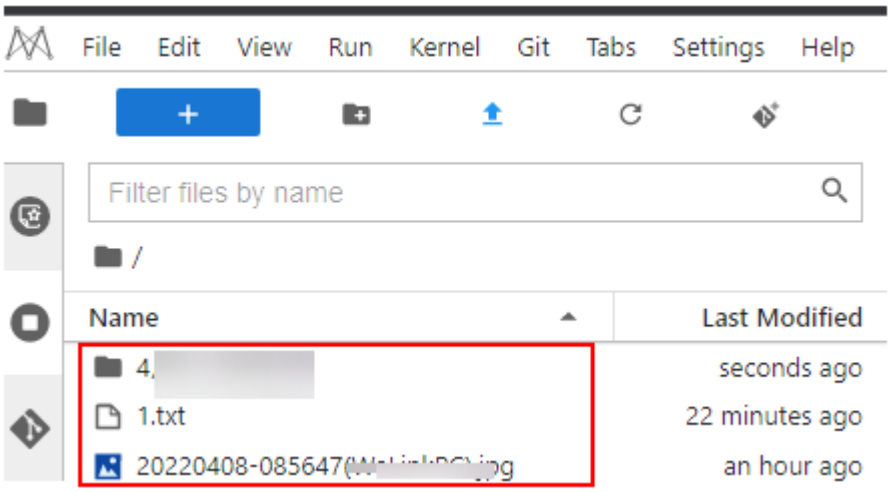
     

Figure 1-85 File downloaded to a notebook instance



Error Handling

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

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

For details, see **Incorrect OBS Path on ModelArts**.

1.6.6.2 Cloning GitHub Open-Source Repository Files to JupyterLab

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

1. Use JupyterLab to open a running notebook instance.
2. On JupyterLab, click from the top menu bar to upload ModelArts files. In the displayed dialog box, click Git on the left to go to the page for cloning files from a GitHub repository.

Figure 1-86 File upload icon

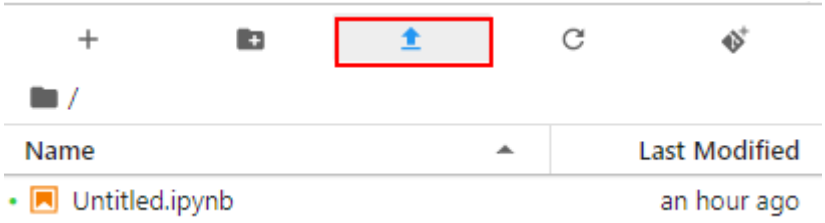
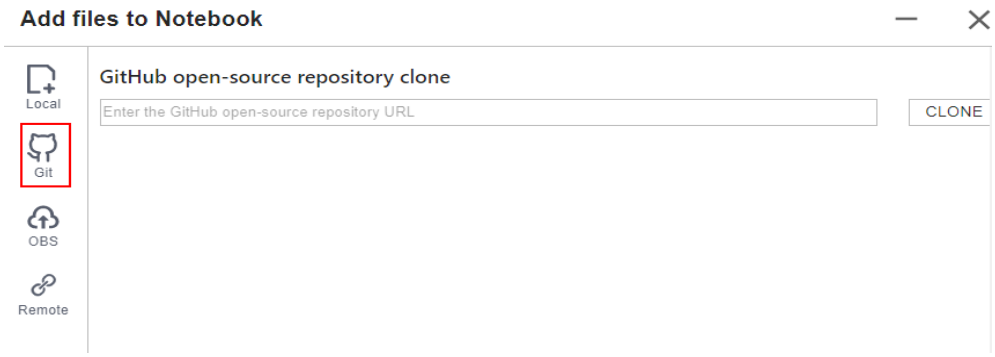


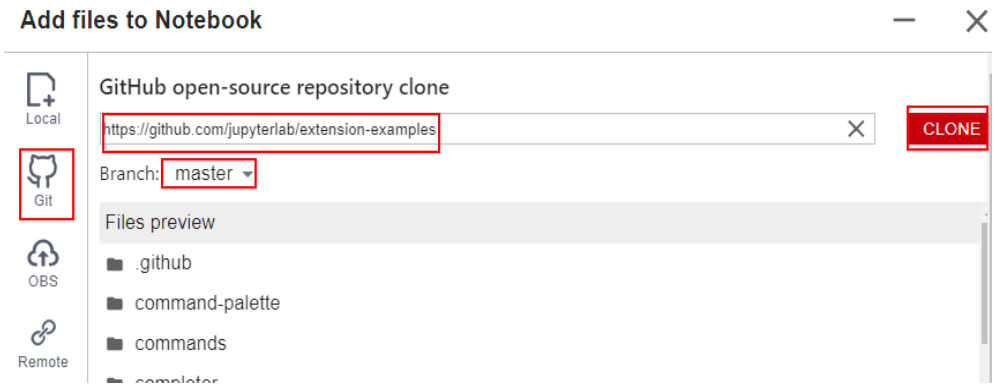
Figure 1-87 Page for cloning files from a GitHub open-source repository



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

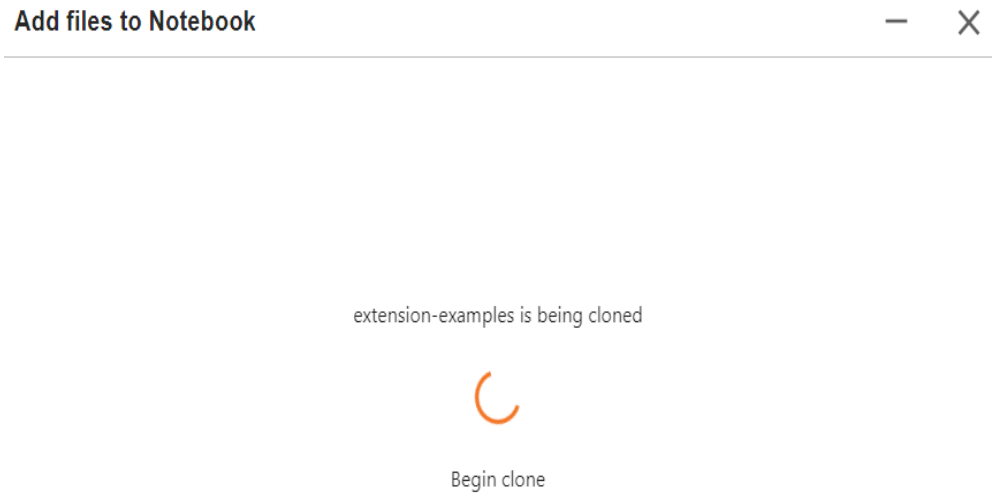
   GitHub open-source repository address: https://github.com/jupyterlab/extension-examples

Figure 1-88 Entering a valid address of a GitHub open-source repository



4. View the clone process.

Figure 1-89 Process of cloning a repository



5. Complete the clone.

Error Handling

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

1.6.6.3 Uploading OBS Files to JupyterLab

In JupyterLab, you can download files from OBS to a notebook instance. Ensure that the file is not larger than 50 GB. Otherwise, the upload will fail.

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

Figure 1-90 File upload icon

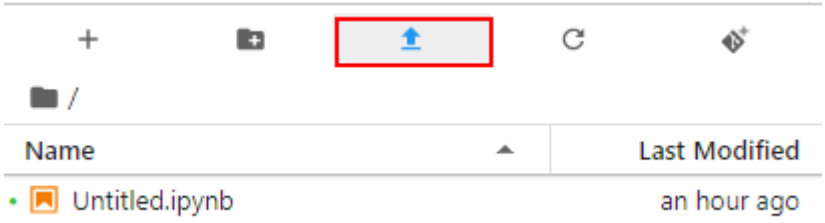
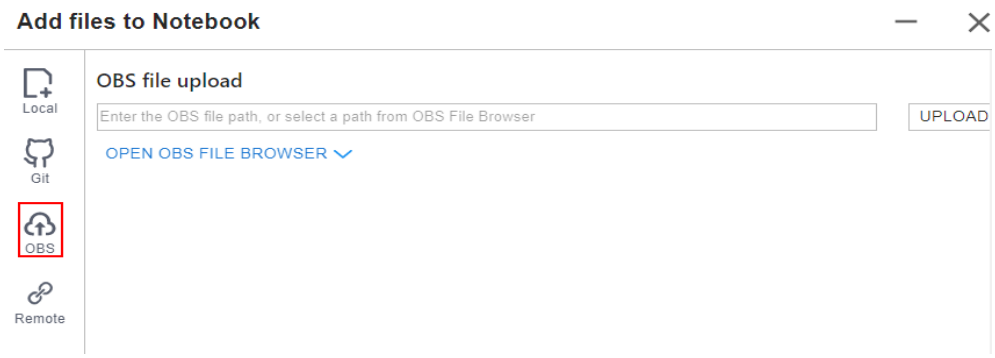
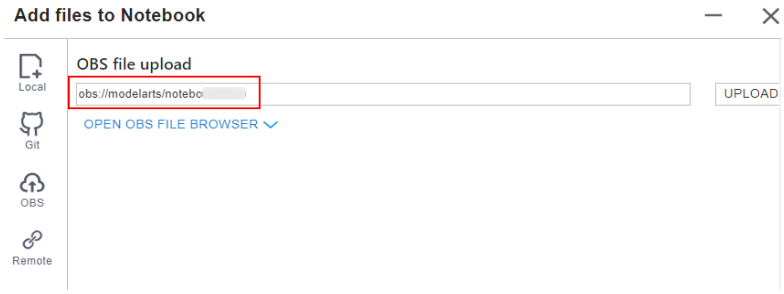


Figure 1-91 OBS file upload



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

Figure 1-92 Entering a valid OBS file path

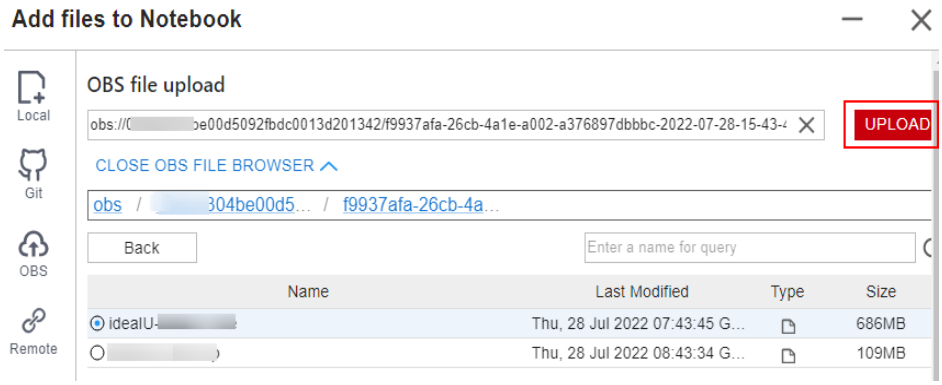


NOTE

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

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

Figure 1-93 Uploading an OBS File



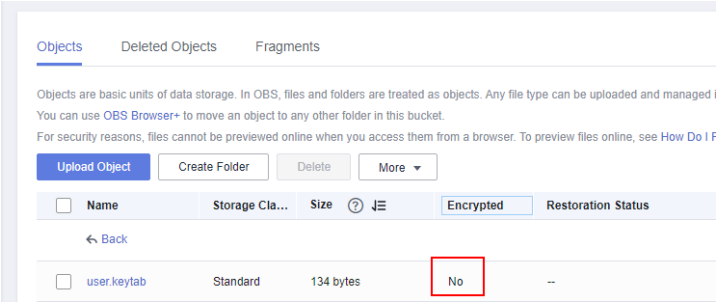
Error Handling

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

- **Scenario 1**

  Possible causes:

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



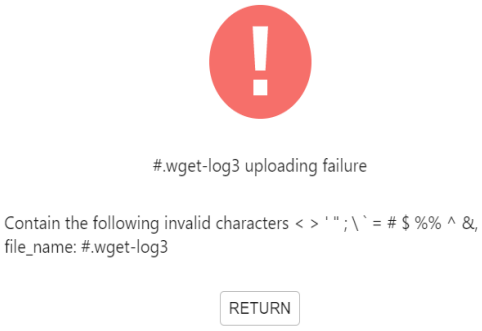
  - The OBS bucket and notebook instance are not in the same region. Ensure that the OBS bucket to be read is in the same region as the notebook instance. You cannot access an OBS bucket in another region.

For details, see **How Do I Check Whether ModelArts and an OBS Bucket Are in the Same Region?**

- The account does not have the permission to access the OBS bucket. In this case, ensure that the notebook account has the permission to read data in the OBS bucket. For details, see **Check Whether You Have Permission to Access the OBS Bucket**.
- The OBS file has been deleted. In this case, make sure that the OBS file to be uploaded is available.

- **Scenario 2**

  Figure 1-94 File uploading failure

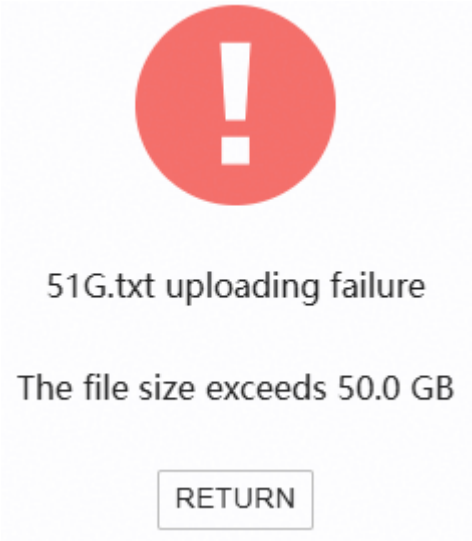
  

  Possible causes:

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

- **Scenario 3**

  Figure 1-95 File uploading failure

  Possible causes:

  The uploaded file exceeded 50 GB.

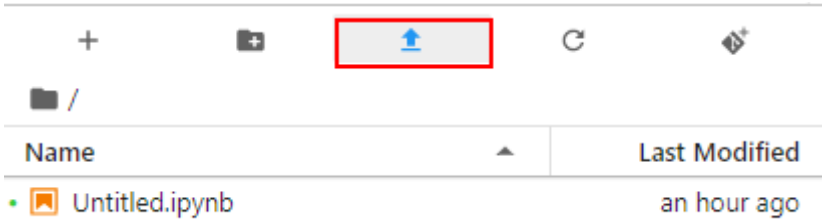
1.6.6.4 Uploading Remote Files to JupyterLab

Files can be downloaded through remote file addresses to JupyterLab.

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

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

   Figure 1-96 File upload icon

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

Error Handling

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

1.6.7 Downloading a File from JupyterLab to a Local PC

Files created in JupyterLab can be downloaded to a local path. For details about how to upload files to JupyterLab, see **Uploading Files to JupyterLab**.

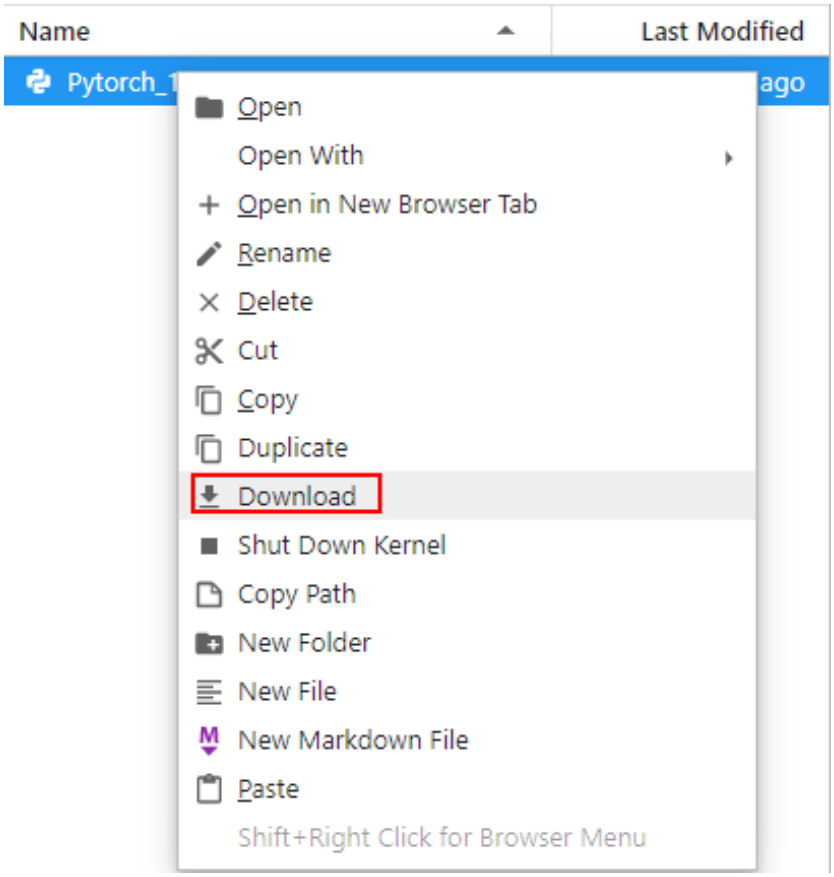
- If a file is less than or equal to 100 MB, directly download it from JupyterLab. For details, see **Downloading a File Less Than or Equal to 100 MB**.
- If a file is larger than 100 MB, use OBS to transfer it to your local path. For details, see **Downloading a File Larger Than 100 MB**.

Downloading a File Less Than or Equal to 100 MB

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

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

Figure 1-97 Downloading a file



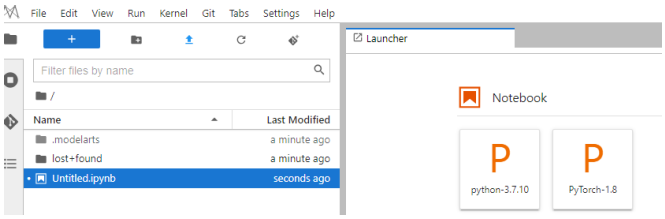
Downloading a File Larger Than 100 MB

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

1. Open the Python runtime environment.

   In the **Launcher** tab, click **python-3.7.10** under **Notebook**, as shown in the following figure.

   Figure 1-98 Opening the Python runtime environment

2. Use MoXing to upload the target file from the notebook instance to OBS.

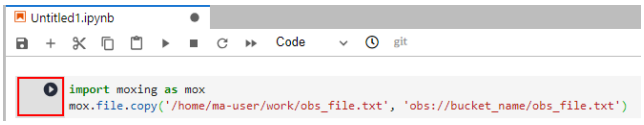
   The following shows the sample Python code for uploading the TXT file and compressed folder. **/home/ma-user/work/xxx** indicates the path for storing the file in the notebook instance. **obs://bucket_name/xxx** indicates the path for storing the file on OBS. Replace them with the actual paths.

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

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

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

Figure 1-99 Running code example



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

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

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

3. Use OBS or ModelArts SDK to download the file from OBS to the local path.
   - Method 1: Use OBS to download the file.

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

     ```
     from modelarts.session import Session

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

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

1.6.8 Using MindInsight Visualization Jobs in JupyterLab

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

MindInsight visualizes information such as scalars, images, computational graphs, and model hyperparameters during training. It also provides functions such as training dashboard, model lineage, data lineage, and performance debugging,

helping you train and debug models efficiently. MindInsight supports MindSpore training jobs. For details about MindInsight, see the **MindSpore official website**.

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

Prerequisites

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

For details, see **Collecting Summary Record**.

Note

- To run a MindInsight training job in a development environment, start MindInsight and then the training process.
- Only single-node single-PU training is supported.
- A running visualization job is not billed separately. When the target notebook instance is stopped, the billing stops.
- If the summary file is stored in OBS, OBS storage will be billed separately. After a job is complete, stop the notebook instance and clear OBS data to stop billing.

Creating a MindInsight Visualization Job in a Development Environment

Step 1 Create a Development Environment and Access It Online

Step 2 Upload the Summary Data

Step 3 Start MindInsight

Step 4 View Visualized Data on the Training Dashboard

Step 1 Create a Development Environment and Access It Online

Log in to ModelArts management console. In the navigation pane on the left, choose **Development Workspace** > **Notebook**, and create a development environment instance using the MindSpore engine. After the development environment is created, click **Access Environment** in the **Operation** column. In the **Access Method** dialog box, click **Access** on the right of **JupyterLab Access** to open the development environment online.

Step 2 Upload the Summary Data

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

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

- For details about how to upload the summary data to the notebook path **/home/ma-user/work/**, see **Uploading Files from a Local Path to JupyterLab**.

- To store the summary data in an OBS parallel file system that is mounted to a notebook instance, upload the summary file generated during model training to the OBS parallel file system and ensure that the OBS parallel file system and ModelArts are in the same region. When MindInsight is started in a notebook instance, the notebook instance automatically reads the summary data from the mounted OBS parallel file system.

Step 3 Start MindInsight

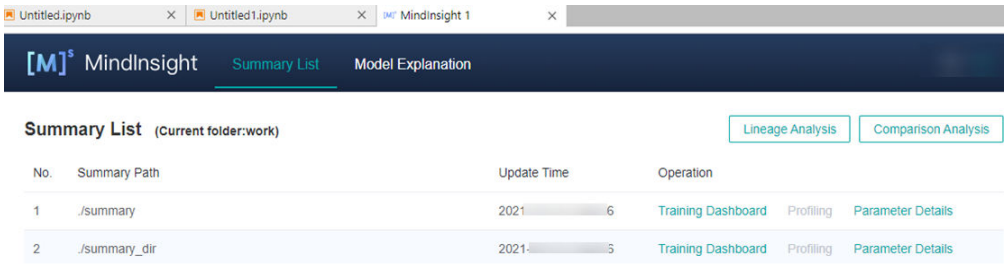
Open MindInsight in JupyterLab.

Click  to go to the MindInsight page.

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

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

Figure 1-100 MindInsight page (2)



Step 4 View Visualized Data on the Training Dashboard

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

For more information, see **Viewing Dashboard** on the MindSpore official website.

Disabling MindInsight

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

Figure 1-101 Stopping an instance

1.6.9 Using TensorBoard Visualization Jobs in JupyterLab

ModelArts supports TensorBoard for visualizing training jobs. TensorBoard is a visualization tool package of TensorFlow. It provides visualization functions and tools required for machine learning experiments. With TensorBoard, computational graph during training, metric trends, and data used during training are effectively displayed. For details about TensorBoard, see the **official website**.

Currently, TensorBoard can be used only in the PyTorch and TensorFlow engines.

Prerequisites

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

For details about how to add the code for collecting the summary record to a TensorFlow-powered training script, see **TensorFlow official website**.

For details about how to add the code for collecting the summary record to a PyTorch-powered training script, see **PyTorch official website**.

Precautions

- A running visualization job is not billed separately. When the target notebook instance is stopped, the billing stops.
- If the summary file is stored in OBS, you will be charged for the storage. After a job is complete, stop the notebook instance and clear OBS data to stop billing.

Process of Creating a TensorBoard Visualization Job in a Development Environment

Step 1 Creating a Development Environment and Accessing It Online

Step 2: Generating Summary Data

Step 3 Uploading Summary Data

Step 4: Starting TensorBoard

Step 5 Viewing Visualized Data on the Dashboard

Step 1 Creating a Development Environment and Accessing It Online

1. Log in to the **ModelArts console**. In the navigation pane on the left, choose **Development Workspace** > **Notebook**.
2. In the upper right corner of the **Notebook** page, click **Create** to create a development environment instance using a TensorFlow or PyTorch image. For details, see **Creating a Notebook Instance (New Page)**.
3. After the development environment is created, click **Access Environment** in the **Operation** column. In the **Access Method** dialog box, click **Access** on the right of **JupyterLab Access** to open the development environment online.

> **NOTE**
>
> If there is a large amount of data, for example, hundreds of MB, in the data directory to be visualized, the CPU or memory of the instance with 2 vCPUs and 8 GB memory may be insufficient, causing the notebook instance fails to work. In this case, user a higher-specification instance, for example, 4 vCPUs and 16 GB memory. Set the specifications based on the site requirements.

Step 2: Generating Summary Data

Summary data is required for using TensorBoard visualization functions in DevEnviron. The following is a simple linear regression training example, which demonstrates how to generate and record summary data. For details about how to generate summary data, see the **PyTorch official document**.

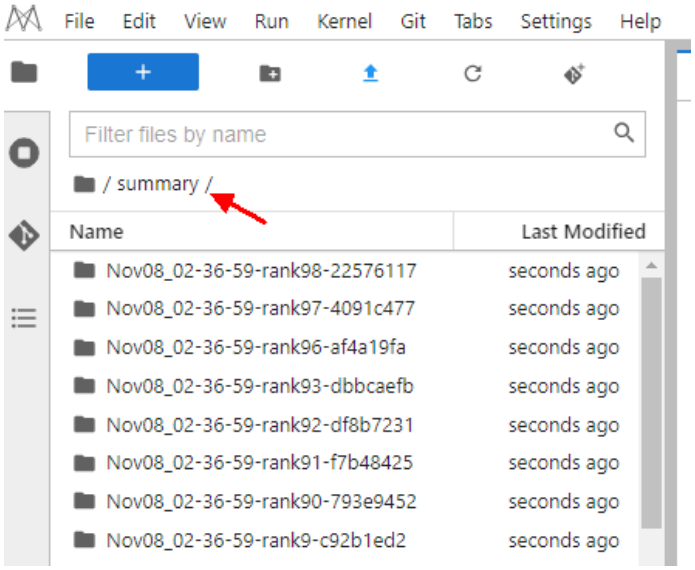
```
import torch
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()

x = torch.arange(-5, 5, 0.1).view(-1, 1)
y = -5 * x + 0.1 * torch.randn(x.size())
model = torch.nn.Linear(1, 1)
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.1)
def train_model(iter):
    for epoch in range(iter):
        y1 = model(x)
        loss = criterion(y1, y)
        writer.add_scalar("Loss/train", loss, epoch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
train_model(10)
writer.flush()
```

Step 3 Uploading Summary Data

You can upload local summary data to the notebook instance as follows:

Create a directory on the left of the notebook, and upload the data to the directory. For details, see **Uploading Files from a Local Path to JupyterLab**.

Figure 1-102 Data to be visualized in the notebook instance



NOTE

You are advised to upload the data to be visualized to a separate directory. If you need to use OBS mounting, mount the OBS path that stores only the visualized data to the notebook instance, instead of mounting the entire bucket, avoiding slow TensorBoard loading or even visualization failure caused by mixed data.

Step 4: Starting TensorBoard

1. Open the Launcher page in JupyterLab of the development environment and click **TensorBoard**.

Figure 1-103 Opening Launcher

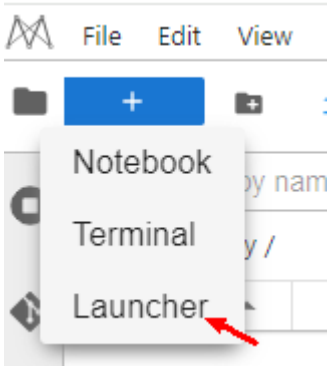
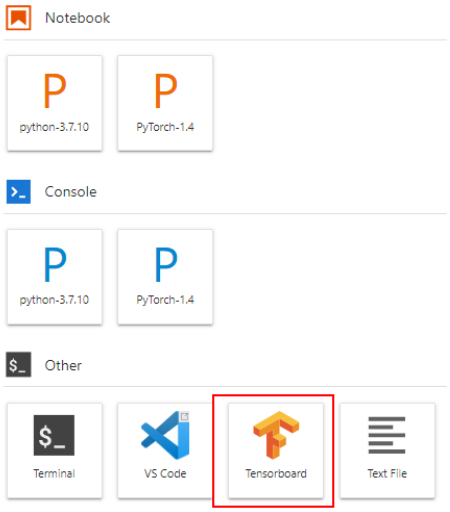


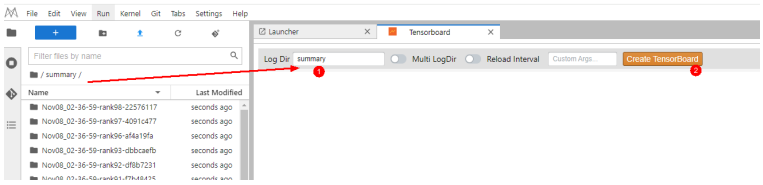
Figure 1-104 Opening TensorBoard in Launcher



2. When you open TensorBoard for the first time, a default initialization panel is displayed, on which you can create a TensorBoard instance.

   Enter the directory that stores the visualized data in the **Log Dir** text box. The path is a relative path under **/home/ma-user/work**. Click **Create TensorBoard** on the right, as shown in **Figure 1-103**. If you have opened TensorBoard before, the first active TensorBoard instance is displayed.

Figure 1-105 TensorBoard page



   The parameter details are as follows:

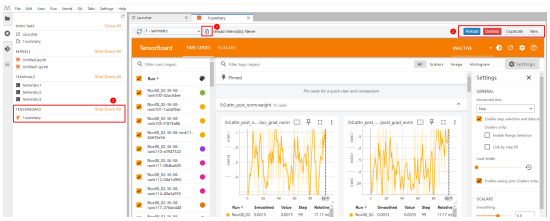
   - **Log Dir**: The default value is the directory in the sidebar when you click TensorBoard. When you enter a directory, specify the directory as detailed as possible to improve the initialization speed.
   - **Multi LogDir**: You can enter multiple directory parameters and separate them with commas (,). This parameter corresponds to the **--logdir_spec** parameter of the native TensorBoard. This function is not fully supported by official TensorBoard and is not recommended.
   - **Reload Interval**: The interval at which TensorBoard rescans the corresponding directory. This function is disabled by default. Manual reload can meet your requirements for daily use. Once this parameter is configured, the TensorBoard backend continuously scans the directory, affecting the stability and file system of JupyterLab.

   For more details about this plugin, see the **JupyterLab TensorBoard Pro website**.

3. **Figure 1-106** shows the created TensorBoard panel.
   - Click the button marked with 1, the visualization panel will be displayed in a separate tab.

- The buttons in the box marked with 2 allow you to manage instances, including restarting a TensorBoard instance (**Reload**), closing an instance (**Destroy**), duplicating a frontend visualization panel (**Duplicate**), and creating a new TensorBoard instance (**New**).
- The buttons in the box marked with 3 allow you to manage instances on the Kernel management panel of JupyterLab. You can switch to the corresponding instance and delete the instance.

Figure 1-106 Visualization page after creation



NOTE

Do not to enable **Reload Interval**. Otherwise, the notebook instance may freeze or even become unavailable due to frequent background refresh. To view new data, click the refresh button in the upper right corner.

Do not create multiple TensorBoard instances by clicking **New**. Otherwise, the CPU or memory usage may be too high, causing the notebook instance to freeze or even become unavailable. If you need to visualize a new directory, close the current TensorBoard instance and specify a new directory to create a TensorBoard instance.

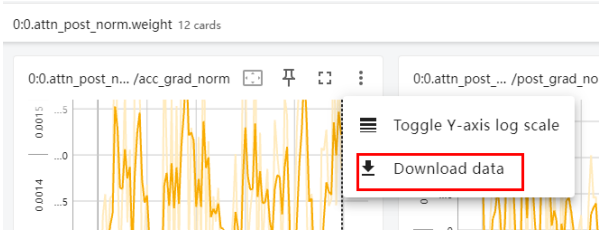
Step 5 Viewing Visualized Data on the Dashboard

The visualization dashboard is important for TensorBoard visualization. The dashboard allows for scalar visualization, image visualization, and computational graph visualization.

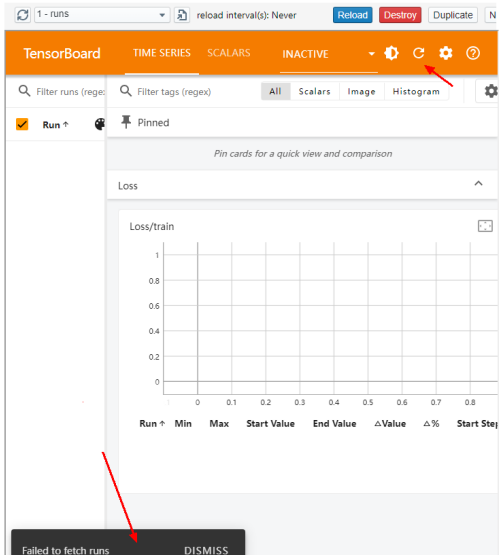
For more functions, see **Get started with TensorBoard**.

NOTE

1. Browser restrictions: Due to security settings in browsers like Chrome, data cannot be downloaded from iFrame. The **Download data** button is currently unsupported.



2. No data displayed: Ensure the correct visualization data directory is selected and that the data is complete. Avoid manually adjusting or deleting summary data from training, as this can cause visualization failures.
3. Data not displaying: If the refresh button keeps spinning, open a Terminal window and run the **top** command to check CPU usage. High CPU or memory usage due to large data volumes may be the issue. Consider reducing the data volume or using a notebook instance with more resources.
4. Occasional data loading issues: If data fails to be loaded, wait a moment and click the refresh button again. The data should display properly after a short wait.



1.6.10 Previewing Markdown, PDF, and Math Formula in JupyterLab

By default, this function is disabled. The preview function has cross-site scripting (XSS) security risks. To ensure system security, do not open untrusted files. Exercise caution when using these functions and ensure that the file source is reliable.

Enabling/Disabling the Preview Function

1. Open the terminal of the notebook instance. For details, see **Common Functions of JupyterLab**.
2. Run the commands below to enable or disable the preview function.
   - Enable the PDF file preview function.
     ```
     $CONDA_BIN/jupyter labextension enable @jupyterlab/pdf-extension
     ```


- Disable the PDF file preview function.
  ```
  $CONDA_BIN/jupyter labextension disable @jupyterlab/pdf-extension
  ```
- Enable the Markdown file preview function.
  ```
  $CONDA_BIN/jupyter labextension enable @jupyterlab/markdownviewer-extension:plugin
  ```
- Disable the Markdown file preview function.
  ```
  $CONDA_BIN/jupyter labextension disable @jupyterlab/markdownviewer-extension:plugin
  ```
- Enable the math formula preview function.
  ```
  $CONDA_BIN/jupyter labextension enable @jupyterlab/mathjax-extension:plugin
  $CONDA_BIN/jupyter labextension enable @jupyterlab/mathjax2-extension:plugin
  ```
- Disable the math formula preview function.
  ```
  $CONDA_BIN/jupyter labextension disable @jupyterlab/mathjax-extension:plugin
  $CONDA_BIN/jupyter labextension disable @jupyterlab/mathjax2-extension:plugin
  ```

3. After enabling or disabling the preview function, refresh the web page.

1.7 Using Notebook Instances Remotely Through VS Code

1.7.1 Connecting to a Notebook Instance Through VS Code

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

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

- **Using VS Code Toolkit to Connect to a Notebook Instance**

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

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

Installing VS Code

When connecting to the development environment using VS Code, you must first install VS Code. The recommended VS Code versions, constraints, and installation guide are as follows.

- **Recommended VS Code versions and constraints**
  - **VS Code 1.86**: No current constraints.

    Figure 1-107 Download location for VS Code version 1.86

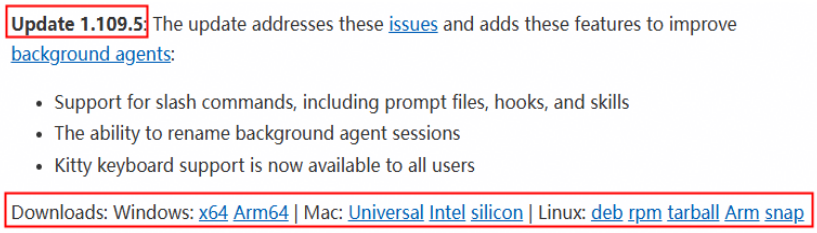
    

  - **VS Code 1.109.5**
    - The image used by the notebook instance requires GLIBC ≥ 2.28.

      You can run the **ldd --version** command to check the GLIBC version. For example, the Ubuntu 18.04 image has GLIBC < 2.28, which means it can only use VS Code 1.86 or lower.
    - The notebook instance must connect to the internet.

      If the notebook instance does not have internet access, you are advised to use VS Code 1.86. If you need to use VS Code 1.109.5, you must manually install the VS Code Server package. For details, see **How Do I Install the VS Code Server of the Local VS Code Version Offline in a Notebook Instance?**

    Figure 1-108 Download location for VS Code version 1.109.5

- **VS Code installation guide**:

  In Windows, double-click the installation package to complete the installation.

  In a Linux system, execute the command **sudo dpkg -i** <*VS-Code-package-name*> to perform the installation.

  For example, if the VS Code installation package name is **code_1.86.2-1707854558_amd64.deb**, the command example is as follows:

  ```
  sudo dpkg -i code_1.86.2-1707854558_amd64.deb
  ```

  NOTE

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

1.7.2 Using VS Code Toolkit to Connect to a Notebook Instance

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

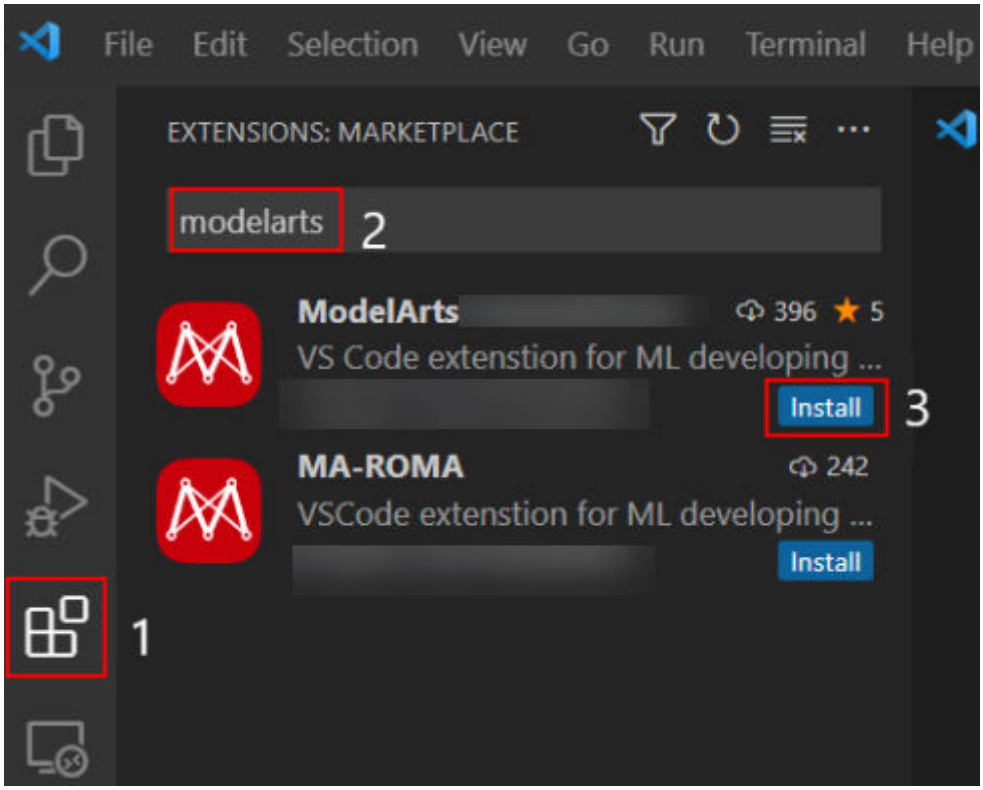
Prerequisites

You have downloaded and installed VS Code. For details, see **Connecting to a Notebook Instance Through VS Code**.

Step 1 Install the VS Code Plug-in

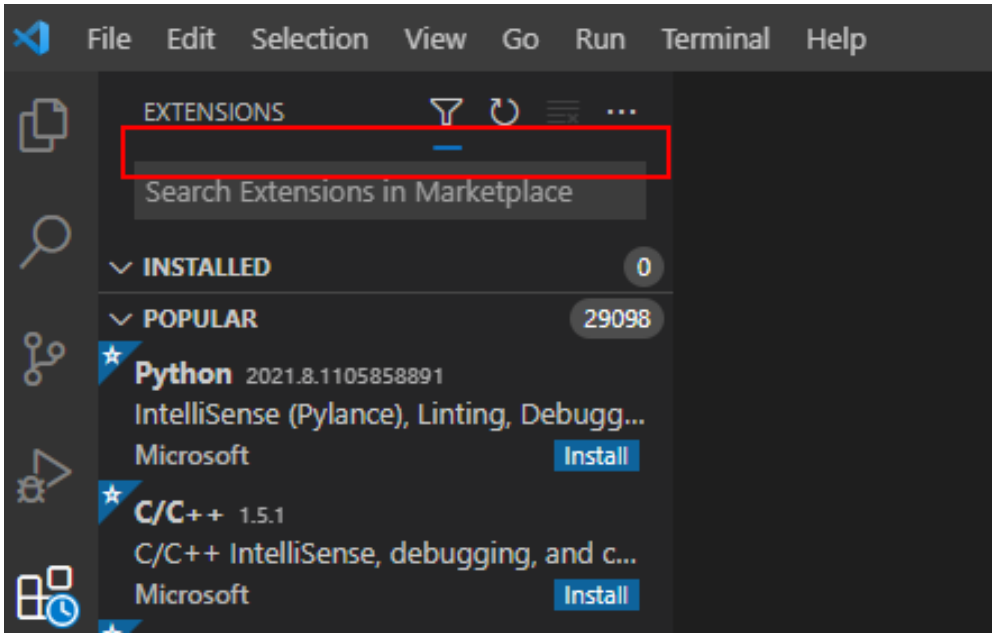
1. Search for **ModelArts-HuaweiCloud** in the **EXTENSIONS** text box and click **Install**.

Figure 1-109 Installing the VS Code plug-in



2. Wait for about 1 to 2 minutes.

Figure 1-110 Installation process



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

Figure 1-111 Installation completion message

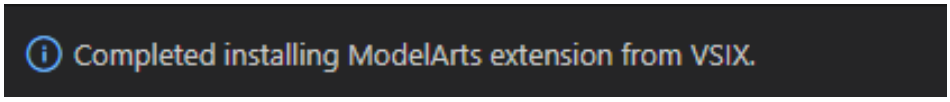
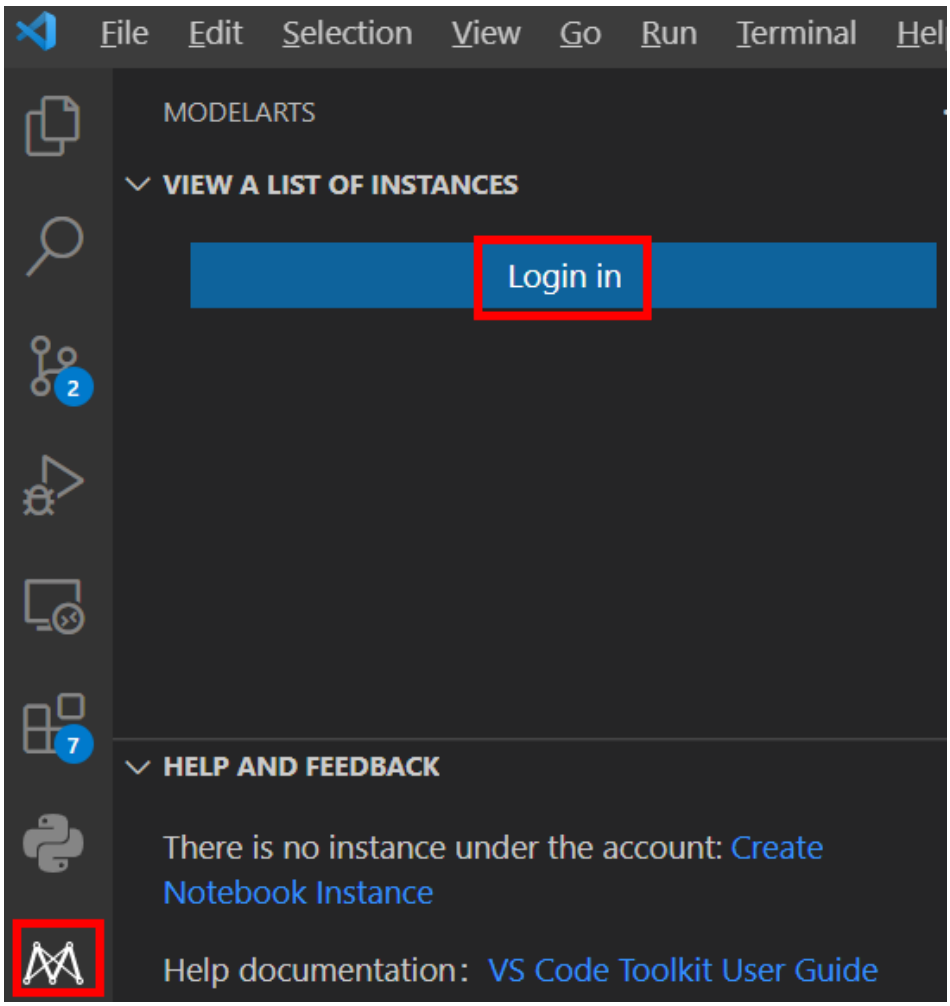
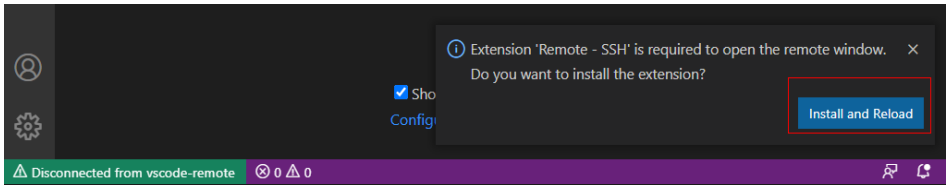


Figure 1-112 Installation completed



Network issues may cause an installation failure. If this occurs, proceed with follow-up operations. After **1** in **Step4 Access the Notebook Instance** is performed, the system will automatically display a dialog box shown in the following figure. In this case, click **Install and Reload**.

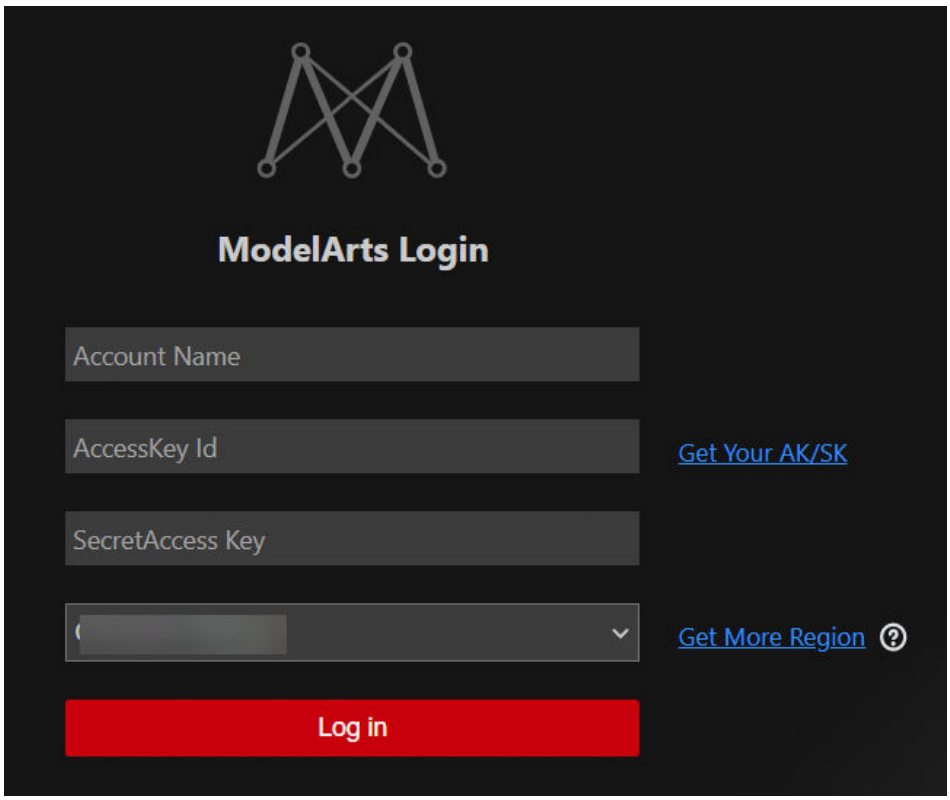
Figure 1-113 Reconnecting remote SSH



Step 2 Log In to the VS Code Plug-in

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

   Figure 1-114 Logging in to the plug-in

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

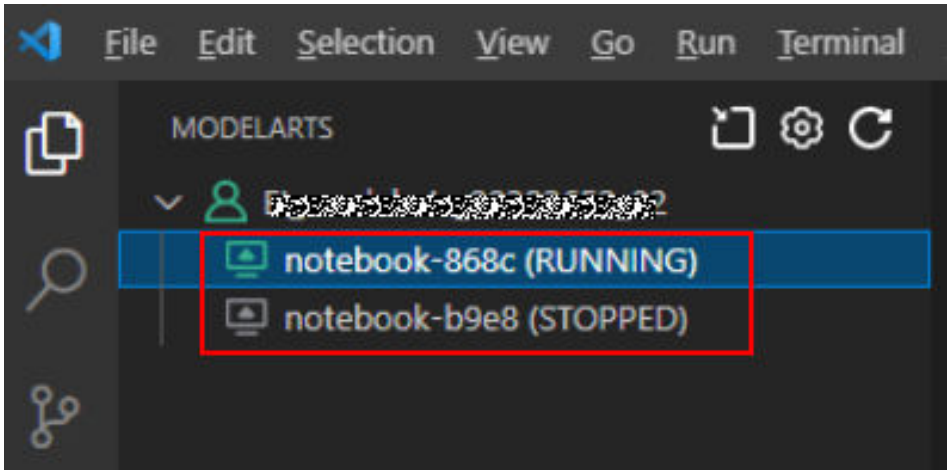
   - **Name**: Custom username, which is displayed only on the VS Code page and is not associated with any Huawei Cloud account.
   - AK and SK: Access key pair. To create a key pair, choose **My Credentials** > **API Credentials** > **Access Keys**, and click **Create Access Key**. For details about how to obtain the AK/SK, see **How Do I Obtain an Access Key?**
   - Region: must be the same as that of the notebook instance to be remotely connected. Otherwise, the connection will fail.

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

   NOTE

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

Figure 1-115 Login succeeded



Step3 Create a Notebook Instance

CAUTION

- Create a notebook instance with remote SSH enabled, and download the key file to either of the following directories based on your OS:

  Windows: **C:\Users\{{user}}**

  macOS/Linux: ~
- A key pair is automatically downloaded after you create it. Securely store your key pair. If an existing key pair is lost, create a new one.

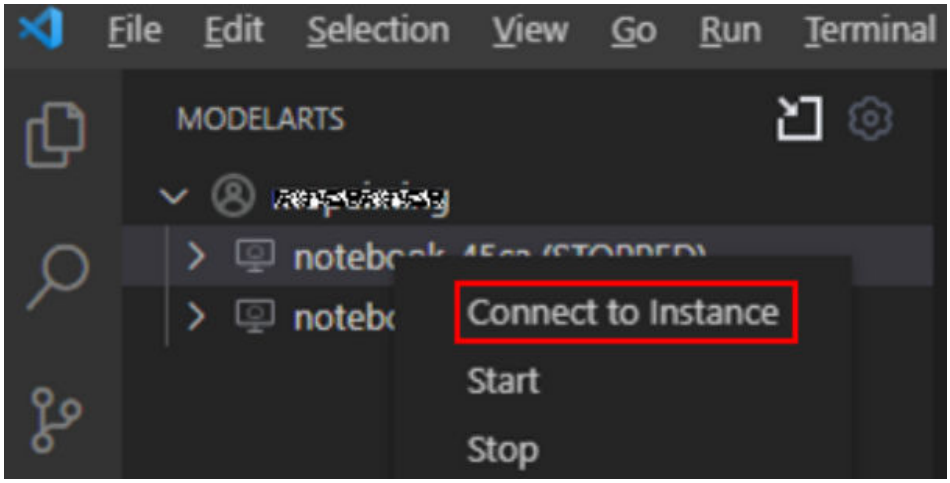
Create a notebook instance with remote SSH enabled. For details, see **Creating a Notebook Instance (Old Page)**.

Step4 Access the Notebook Instance

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

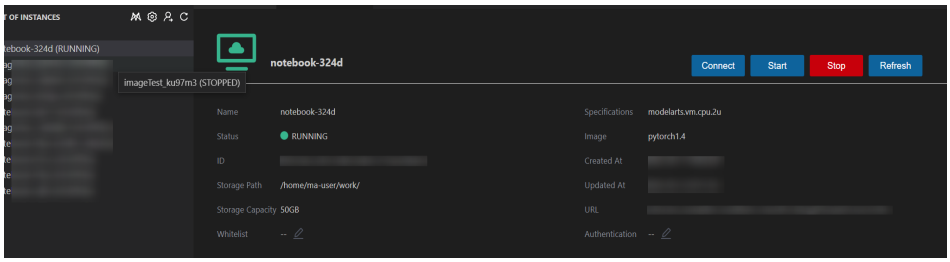
   The notebook instance can either be running or stopped. If it is stopped, the VS Code plug-in starts the instance and then connects to it.

Figure 1-116 Connecting to a notebook instance



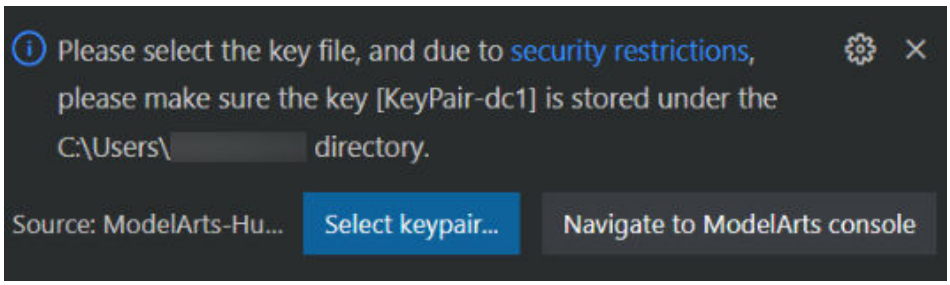
Alternatively, click the instance name. On the instance details page, click **Connect**. Then, the system automatically starts and connects to the notebook instance.

Figure 1-117 Viewing details about a notebook instance



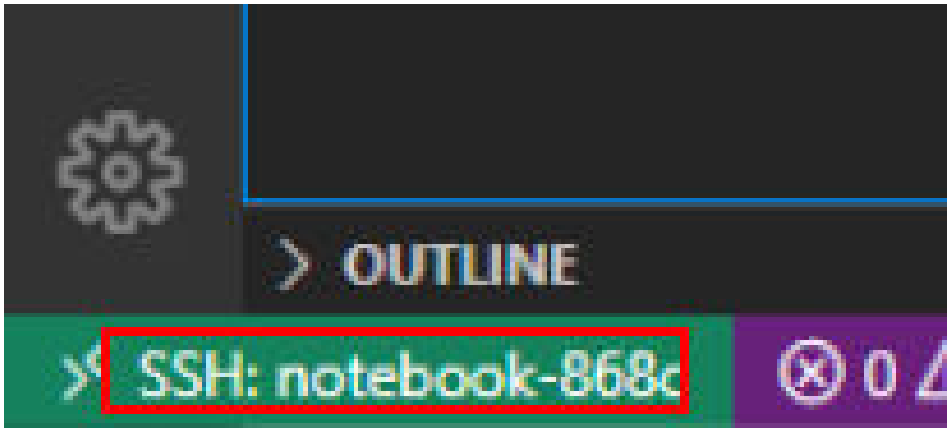
2. When you connect to a notebook instance for the first time, the system prompts you in the lower right corner to configure the key file. In this case, select the local .pem key file and click **OK**.

Figure 1-118 Configuring the key file



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

Figure 1-119 Connection succeeded



Debugging Code Remotely

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

Figure 1-120 Open Folder

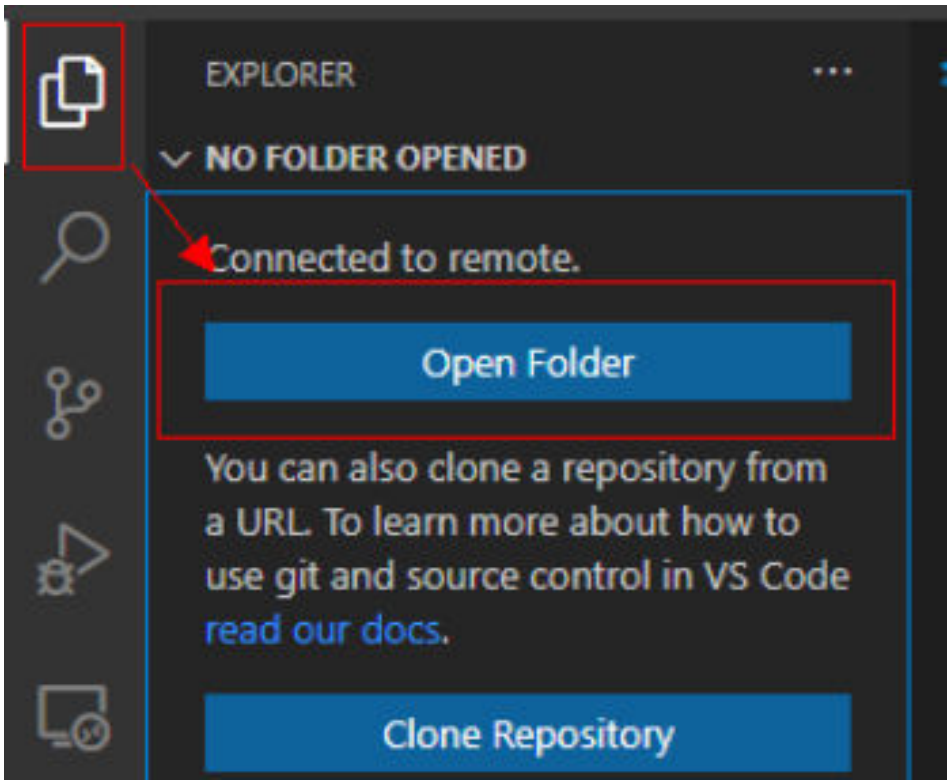
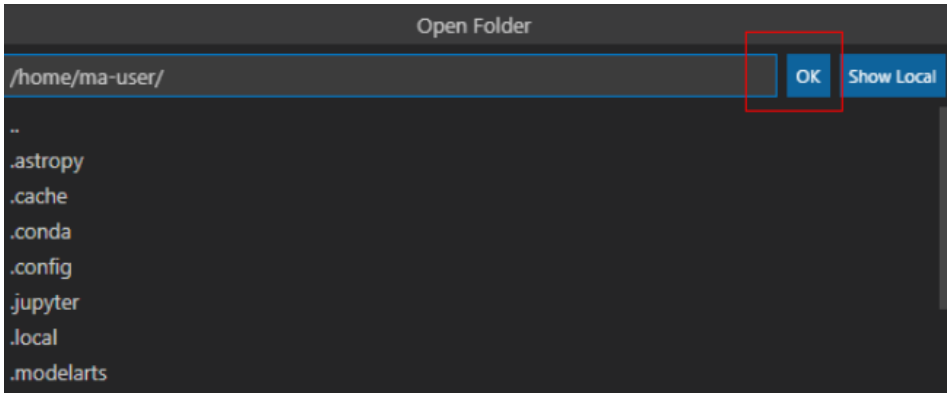


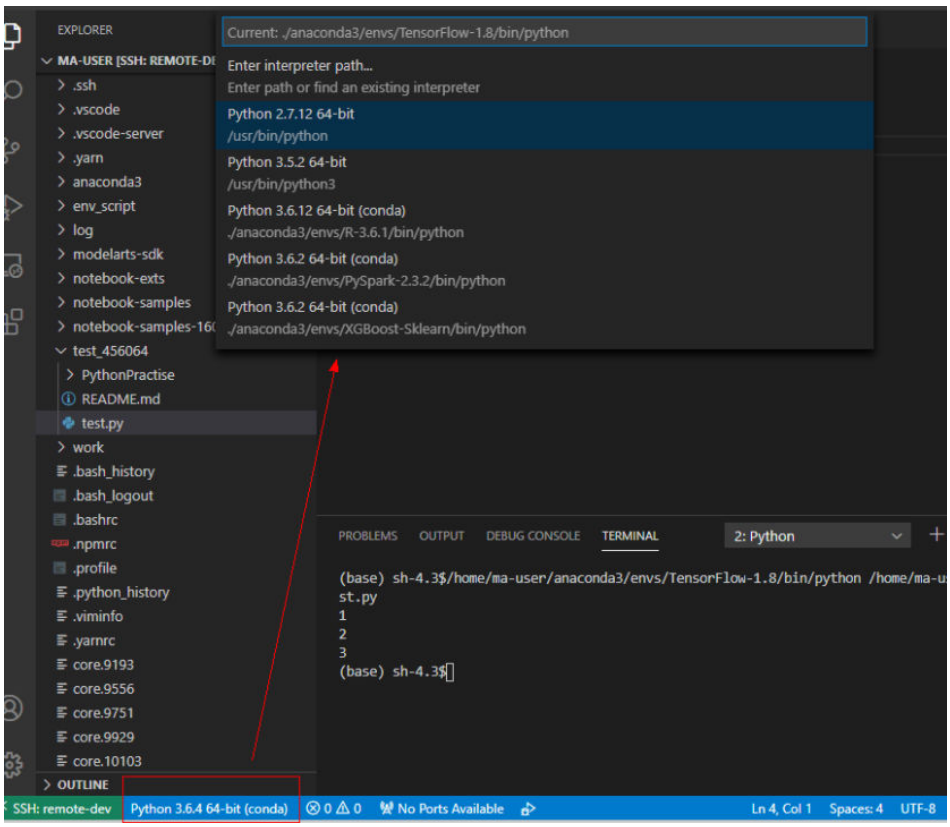
Figure 1-121 Selecting a file path



b. In the displayed directory structure on the left of the IDE, drag the code and files you want to upload to the corresponding folders. Then, the code is uploaded to the cloud development environment.

c. Open the code file to be debugged in VS Code. Before running the code, click the default Python version in the lower left part and select a version as required.

Figure 1-122 Selecting a Python version



2. Click the execution button to run the code. The code output is shown in the **TERMINAL** tab.
   - If a training job takes a long time to execute, run the job at the backend using the **nohup** command. This prevents the disconnection of an SSH

session or a network failure from affecting job execution. The following shows an example **nohup** command:

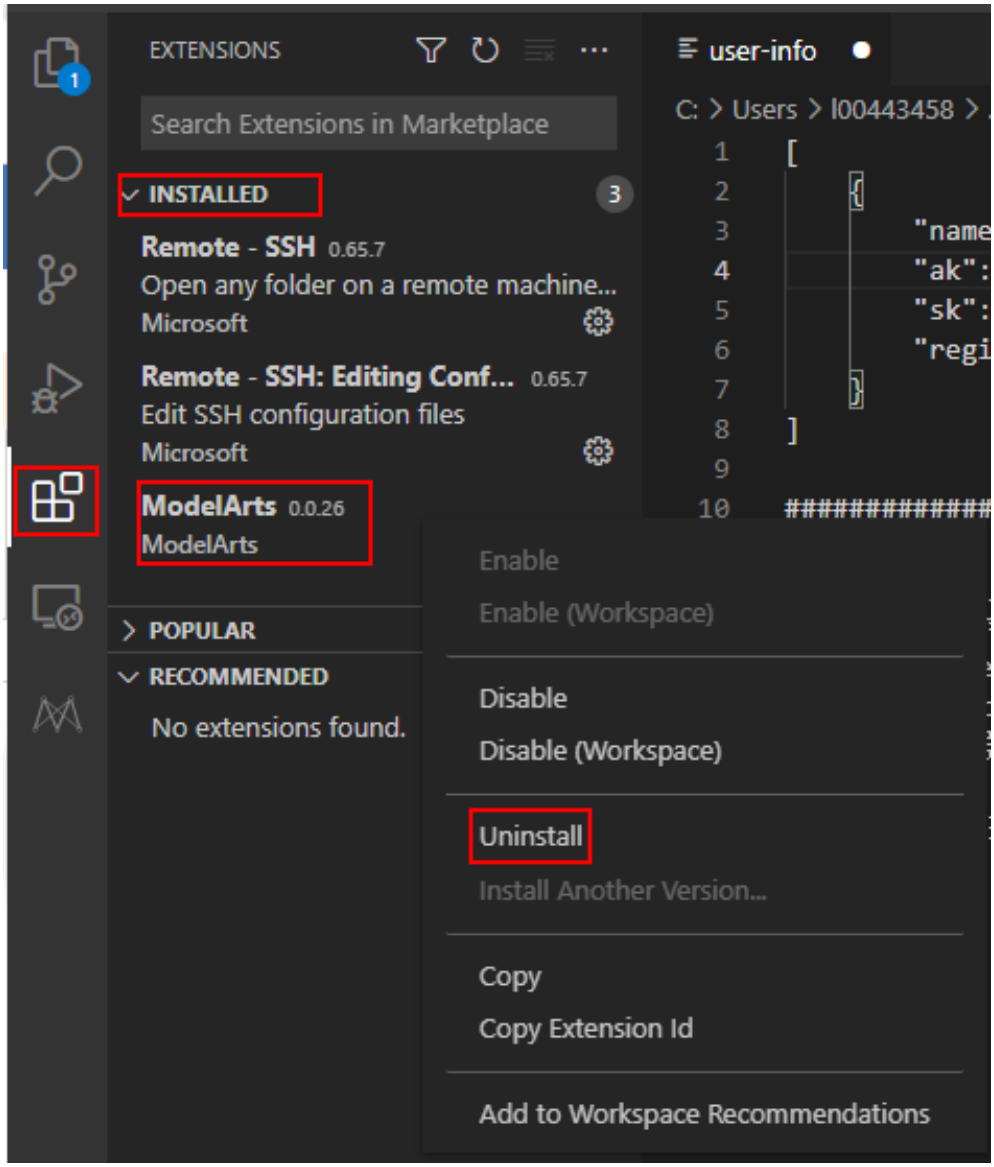
```
nohup your_train_job.sh > output.log 2>&1  &  tail -f  output.log
```

- To debug the code, perform the following operations:
  i. Choose **Run** > **Run and Debug** on the left.
  ii. Select the default Python code file.
  iii. Click on the left of the code to set breakpoints.
  iv. Debug the code according to the debug procedure which is displayed above the code, and the debug information is displayed on the left of the page.

Related Operations

For details about uninstalling the VS Code plug-in, see **Figure 1-123**.

Figure 1-123 Uninstalling the VS Code plug-in



1.7.3 Manually Connecting to a Notebook Instance Through VS Code

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

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

Prerequisites

- You have downloaded and installed VS Code. For details, see **Connecting to a Notebook Instance Through VS Code**.
- Python has been installed on your local PC or server. For details, see **VS Code official documentation**.
- A notebook instance has been created with remote SSH enabled. The instance is running. For details, see **Creating a Notebook Instance (New Page)**.
- The address for accessing the development environment has been obtained on the notebook instance details page.

  Figure 1-124 Instance details page

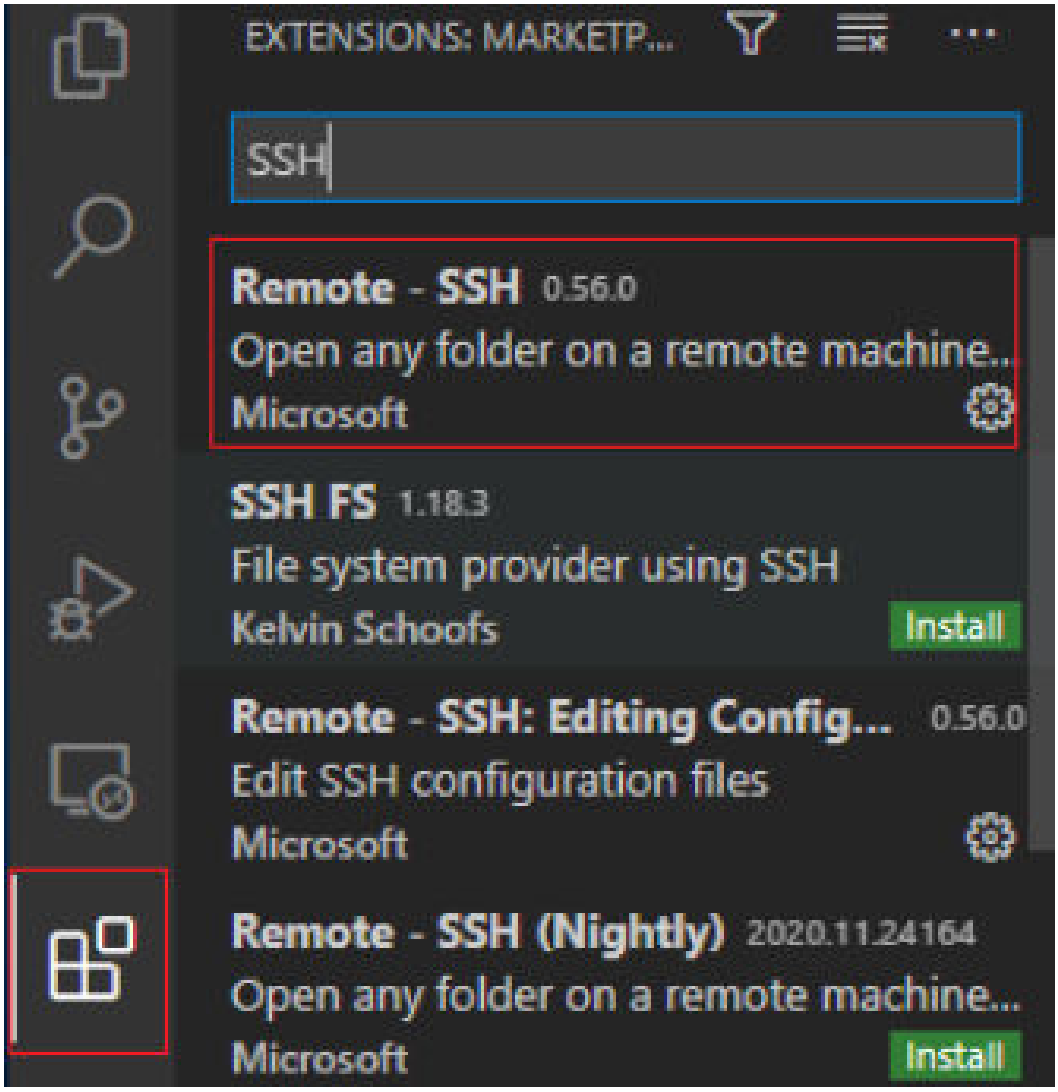
- The key pair is available.

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

Step 1 Add the Remote-SSH Plug-in

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

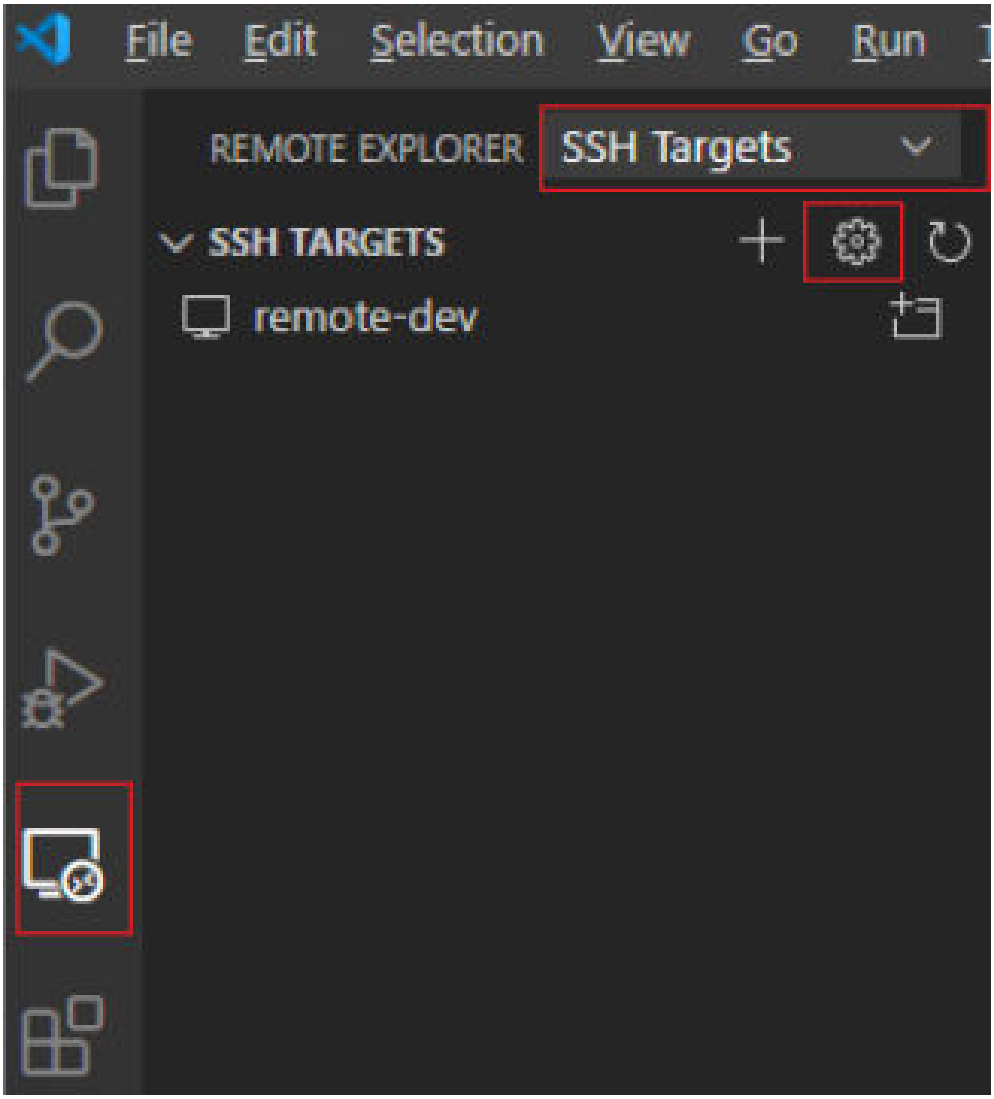
Figure 1-125 Adding the Remote-SSH plug-in



Step 2 Configure SSH

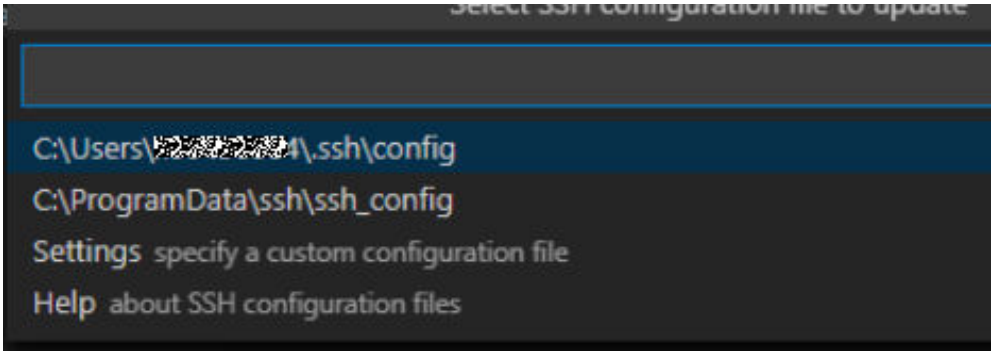
1. In the local VS Code development environment, click  on the left, select **SSH Targets** from the drop-down list box, and click  . The SSH configuration file path is displayed.

Figure 1-126 Configuring SSH Targets



2. Click the SSH configuration path and configure SSH.

Figure 1-127 SSH configuration file path




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

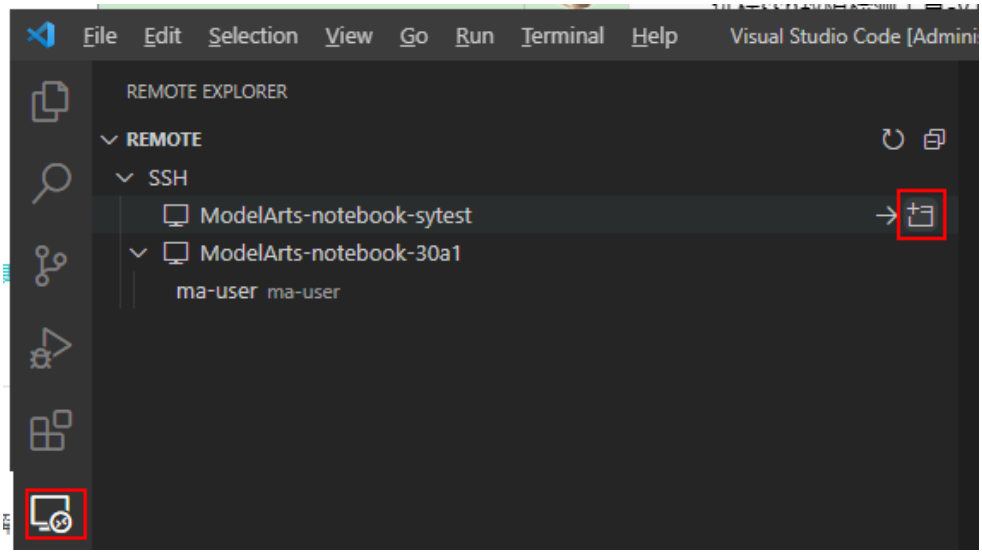


```
UserKnownHostsFile=/dev/null
StrictHostKeyChecking no
```

- **HOST**: name of the cloud development environment
- **HostName**: address for accessing the cloud development environment. Obtain the address on the page providing detailed information of the target notebook instance.
- **Port**: port number for accessing the cloud development environment. Obtain the port number on the page providing detailed information of the target notebook instance.
- **User**: The default login user is **ma-user**.
- **IdentityFile**: locally stored private key file of the cloud development environment. It is the key pair file in **Prerequisites**.

3. Return to the **SSH Targets** page, locate the target remote development environment, and click  on the right to connect to host in a new window.

Figure 1-128 Opening the development environment



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

Figure 1-129 Selecting a notebook running environment

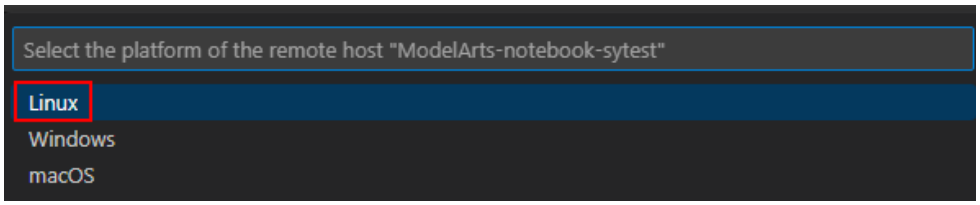
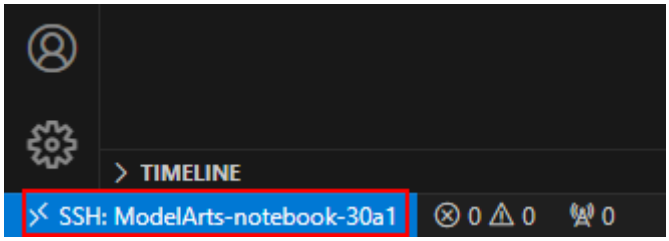


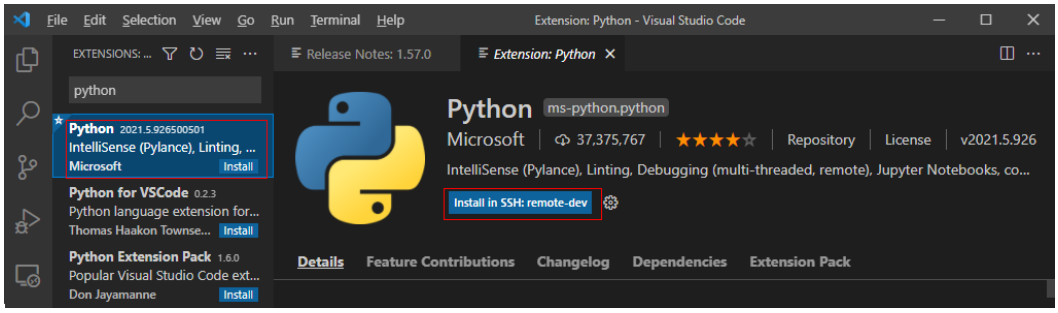
Figure 1-130 Remote connection succeeded



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

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

Figure 1-131 Installing the Python plug-in in the cloud development environment



If the remote Python plugin fails to be installed, you are advised to install it using an offline package.

Step 4 Install the Dependent Library for the Cloud Environment

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

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

1.7.4 Uploading and Downloading Files in VS Code

Uploading Data to a Notebook Instance Using VS Code

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

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

Procedure

1. Upload data to OBS. For details, see **Using OBS Console**.

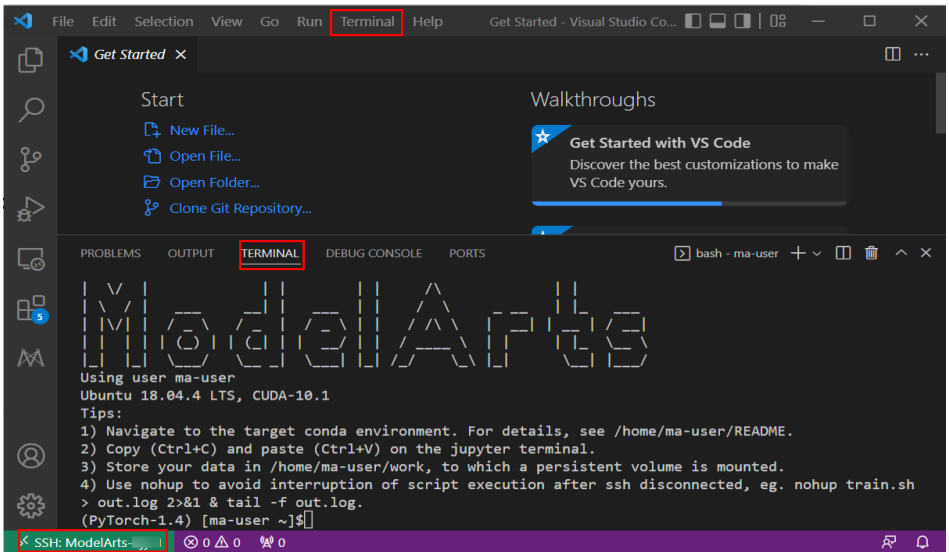
   Alternatively, use ModelArts SDK in the terminal of local VS Code to upload data to OBS. To do so, click **Terminal** on the top menu bar. Enter **python** and press **Enter** in terminal to access the Python environment.

   ```
   python
   ```

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

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

   Figure 1-132 Opening the terminal in the remote VS Code environment

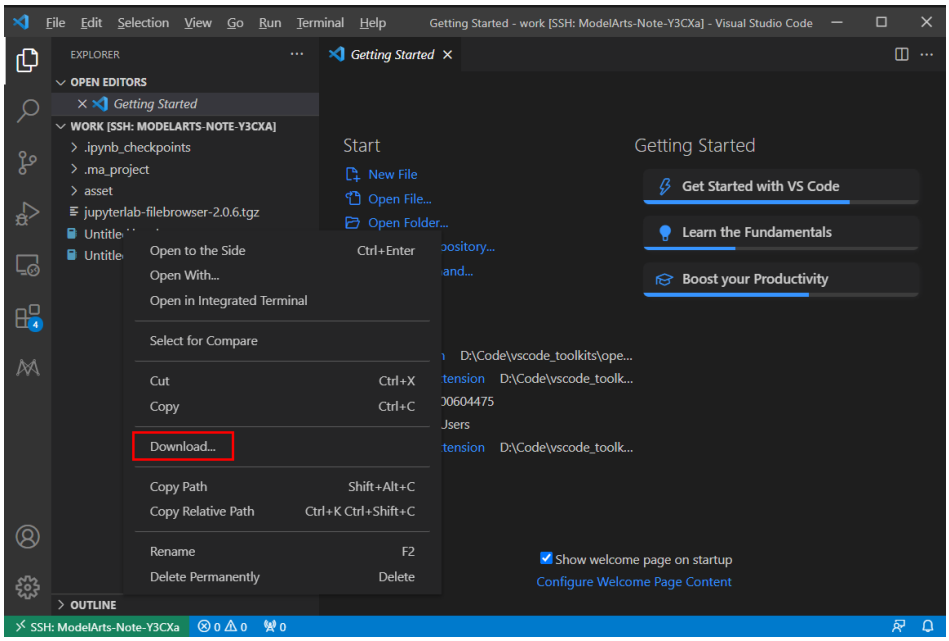

   ```
   # Manually access the development environment using the source command.
   cat /home/ma-user/README
   # Select the target environment.
   source /home/ma-user/miniconda3/bin/activate MindSpore-python3.7-aarch64
   # Enter python and press Enter to access the Python environment.
   python
   ```

   Then, perform OBS transfer operations by referring to **Uploading a File to OBS**.

Downloading Files from a Notebook Instance to a Local Directory

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

Figure 1-133 Downloading files from a notebook instance to a local directory in VS Code



1.8 Using a Notebook Instance Remotely with SSH

This topic describes how to use CMD and MobaXterm to remotely log in to a notebook instance using SSH in a Windows environment.

Constraints

To connect to a notebook instance using SSH, pay attention to the following constraints:

- The local user key and permission must match.
- The local user key must be stored in the specified directory.
  - Windows: **C:\Users\{{user}}**
  - macOS/Linux: ~

    NOTE

    On macOS and Linux, ~ indicates the home directory of the current user.
- The **ma-user** or **root** user in the remote image cannot be locked.
- Set the permission on the remote **~/.ssh** directory to 750 or 755.
- The OpenSSH version on the local or remote server cannot be earlier than 8.0.
- Do not create more than 10 connections at the same time.

If SSH connection problems occur, rectify the fault by referring to **Accessing a Notebook Instance Using SSH for Troubleshooting**.

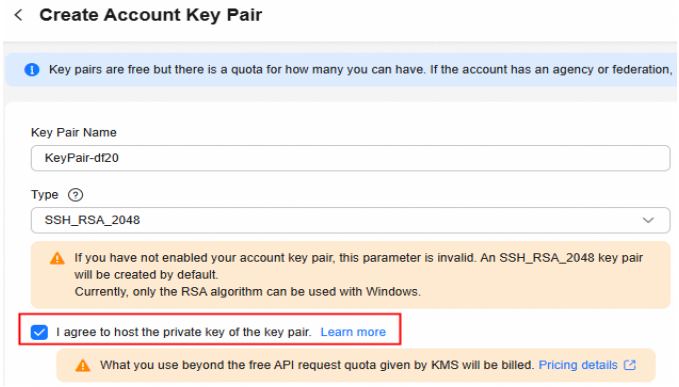
Prerequisites

- You have created an account key pair on the **DEW console** and saved the key file to a specified directory. For details, see **Creating a Key Pair**.
  - Windows: **C:\Users\{{user}}**
  - macOS/Linux: ~

CAUTION

If you choose **I agree to host the private key of the key pair** when creating a key pair, you can download the key again by clicking **Export Private Key** on the **Account Key Pairs** page. If you do not choose this option, the key will only be available for download once, during creation. Keep your key safe.

Figure 1-134 Creating an account key pair



- You have created a notebook instance with remote SSH enabled (select the created account key pair). The instance is in the **Running** state.
  - For details about how to create a notebook instance, see **Creating a Notebook Instance (New Page)**.
  - For details about how to enable remote SSH for an existing notebook instance, see **Configuring Remote SSH Connection for a Notebook Instance**.
- On the notebook instance details page, obtain the access address for remote SSH development from the basic information tab.

Figure 1-135 Creating a notebook instance for remote SSH development

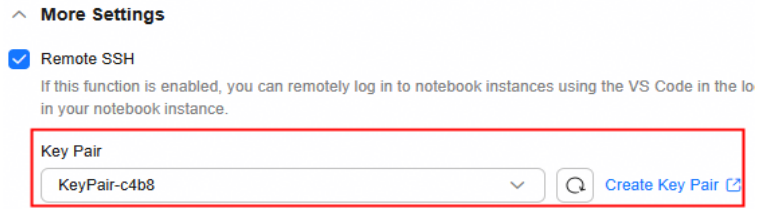


Figure 1-136 Notebook instance details page

Method 1: Connecting to a Notebook Instance Using CMD

The following uses Windows as an example.

1. Open CMD.
   - Option A: Using the start menu
     i. Click the start button in the lower-left corner of your screen, or press the Windows key on your keyboard.
     ii. Type **CMD** in the start menu and click **Command Prompt**.
   - Option B: Using the run dialog box
     i. Press **Windows + R** on your keyboard to open the **Run** dialog box.
     ii. Type **CMD** and press **Enter** or click **OK**.
2. In CMD, navigate to the directory where your key file is located and execute the following command to connect to the notebook instance:

   ```
   ssh -o StrictHostKeyChecking=no -i {key-file-name} {launch-user}@{domain-name} -p {port}
   ```

   Note: Replace {key-file-name}, {launch-user}, {domain-name}, and {port} with your actual information obtained from the **Prerequisites**.

   Example: If your key file name is **KeyPair-1234-test.pem**, and the remote SSH access address is **ssh://ma-user@authoring-ssh-modelarts-*****.huawei.com:31092** (where **ma-user** is the launch user, **authoring-ssh-modelarts-*****.huawei.com** is the domain name, and **31092** is the port), the command will be:

   ```
   ssh -o StrictHostKeyChecking=no -i KeyPair-1234-test.pem ma-user@authoring-ssh-modelarts-*****.huawei.com -p 31092
   ```

   As shown in the figure below, the appearance of the interactive interface indicates a successful connection.

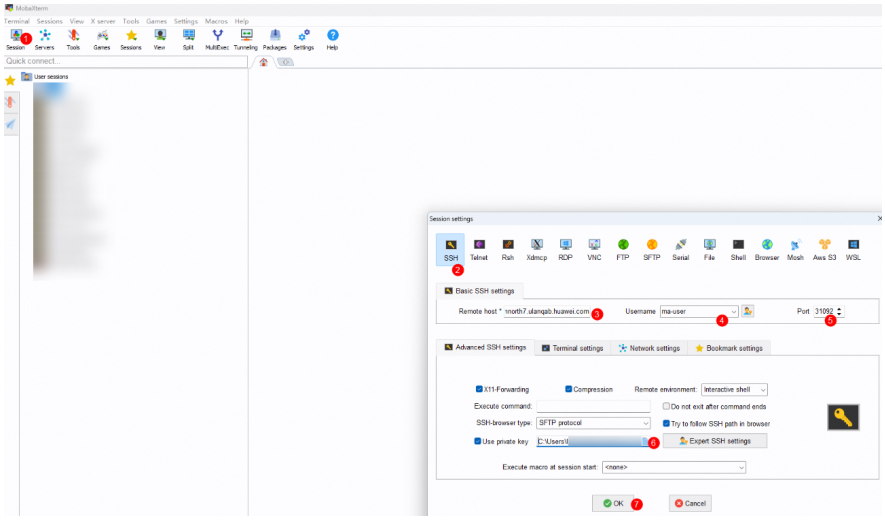
   Figure 1-137 Connecting to a notebook instance using CMD

Method 2: Connecting to a Notebook Instance Using MobaXterm

1. Download and install **MobaXterm**.
2. Open MobaXterm, click **Session** in the upper-left corner, and navigate to the SSH tab in the **Session settings** dialog box. Configure the **Remote host**, **Username**, **Port**, and **Use private key**, and click **OK**.

Figure 1-138 Configuring a session



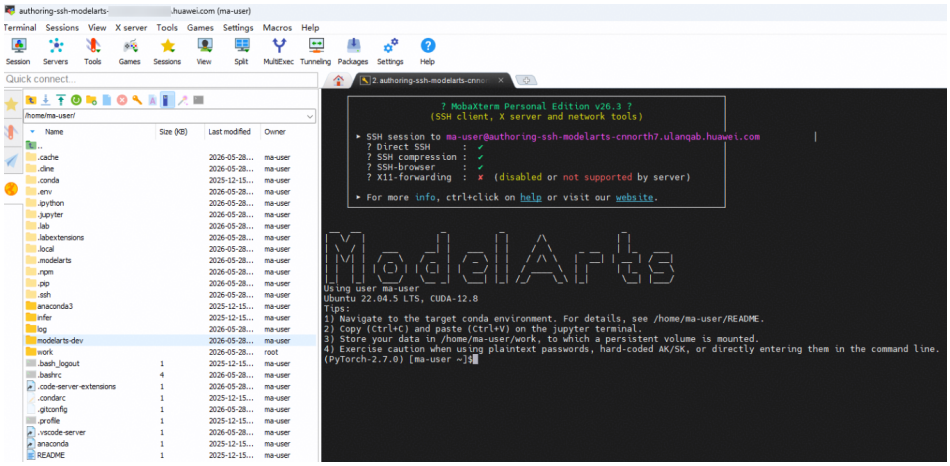
Obtain the remote SSH access address and the key file path from the **Prerequisites**. For example, if the remote SSH access address is **ssh://ma-user@authoring-ssh-modelarts-*****.huawei.com:31092**, configure the settings as follows.

Table 1-24 Parameters

Parameter		Description	Example Value
Basic SSH settings	Remote host	The address of the remote host, obtained from the remote SSH access address.	**authoring-ssh-modelarts-*****.huawei.com**
	Username	The username used to connect to the remote host, obtained from the remote SSH access address.	**ma-user**
	Port	The port number used to connect to the remote host, obtained from the remote SSH access address.	**31092**
Advanced SSH settings	Use private key	The local file path of your private key, obtained from the **Prerequisites**.	C:\Users***\KeyPair-1234-test.pem

As shown in the figure below, the appearance of the interactive interface indicates a successful connection.

Figure 1-139 Connecting to a notebook instance using MobaXterm



1.9 ModelArts CLI Command Reference

1.9.1 ModelArts CLI Commands

Description

ModelArts CLI, also called ma-cli, is a cross-platform command line tool used to connect to ModelArts and run management commands on ModelArts resources. You can use the interactive command prompt or script to run commands on a terminal. ma-cli allows you to interact with cloud services through ModelArts notebook and on-premises VMs. You can run ma-cli commands for command autocomplete and authentication, as well as creating images, submitting ModelArts training jobs and DLI Spark jobs, and copying OBS data.

Application Scenarios

- ma-cli has been integrated into ModelArts notebook and can be directly used.
- Log in to the **ModelArts console**. In the navigation pane, choose **Development Workspace** > **Notebook**. Create a notebook instance, open the terminal, and run the **ma-cli** command. For details about how to open the terminal, see **Creating a Terminal in JupyterLab**.
- In local Windows or Linux, install ma-cli and then use it on a local terminal. For details, see **(Optional) Installing ma-cli Locally**.

  NOTE

  - ma-cli cannot be used in Git Bash.
  - Terminals such as Linux Bash, Zsh, Fish, WSL, and PowerShell are recommended. To ensure the security of your sensitive information, it is important to prevent any potential leakage when using terminals.

Command Preview

```
$ ma-cli -h
Usage: ma-cli [OPTIONS] COMMAND [ARGS]...

Options:
  -V, -v, --version       1.2.1
  -C, --config-file TEXT  Configure a file path for authorization.
  -D, --debug             Debugging mode, in which the full stack trace will be displayed when an error occurs.
  -P, --profile TEXT      CLI connection profile to be used. The default profile is DEFAULT.
  -h, -H, --help          Show the help information and exit.

Commands:
  configure        Configure authentication and endpoints for the CLI.
  image            Obtain registered images, register or unregister images, debug images, and create images in
Notebook.
  obs-copy         Copy files or directories between OBS and a local path.
  ma-job           Submit ModelArts jobs and obtain jod details.
  dli-job          Submit DLI spark jobs and obtain jod details.
  auto-completion  Auto complete ma-cli command in terminal, support "bash(default)/zsh/fish".
```

Among the preceding parameters, parameters **-C**, **-D**, **-P**, and **-h** are globally optional.

- **-C** indicates that you can manually specify the authentication configuration file when running this command. By default, the **~/.modelarts/ma-cli-profile.yaml** configuration file is used.
- **-P** indicates a group of authentication information in the authentication file. The default value is **DEFAULT**.
- **-D** indicates whether to enable the debugging mode (disabled by default). After the debugging mode is enabled, the error stack information of the command will be printed. If this mode is disabled, only the error information will be printed.
- **-h** indicates that the help information about the command will be displayed.

Commands

Table 1-25 ma-cli commands

Command	Description
auto-completion	Command autocomplete
configure	ma-cli authentication using username and password or AK/SK
image	ModelArts image creation, registration, and registered image query
ma-job	ModelArts training job management, including job submission and resource query
dli-job	DLI Spark job submission and resource management
obs-copy	Copying files or folders between a local path and OBS

1.9.2 (Optional) Installing ma-cli Locally

Application Scenarios

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

Step 1: Install ModelArts SDKs

Install ModelArts SDKs by referring to **Installing the ModelArts SDK Locally**.

Step 2: Download ma-cli

1. **Download the ma-cli software package**.
2. Verify the software package signature.
   a. **Download the signature verification file of the software package**.
   b. Install OpenSSL and run the following command to verify the signature:

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

NOTE

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

Step 3: Install ma-cli

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

```
C:\Users\xxx>python --version
Python *.*.*
```

2. Run **pip --version** to check whether the general package management tool pip is available.

```
C:\Users\xxx>pip --version
pip **.*.* from c:\users\xxx\appdata\local\programs\python\python**\lib\site-packages\pip (python *.*)
```

3. Install ma-cli.

   pip install *{Path to the ma-cli software package}***ma_cli-latest-py3-none-any.whl**

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

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

   NOTE

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

   pip install xxxx

   xxxx is the name of the dependency package.

1.9.3 Autocompletion for ma-cli Commands

CLI autocomplete enables you to get a list of supported **ma-cli** commands by typing a command prefix and pressing **Tab** on your terminal. Autocomplete for **ma-cli** commands needs to be enabled in Terminal. After running the **ma-cli auto-completion** command, you can copy and run the commands as prompted on the current terminal to automatically complete the **ma-cli** commands. Bash, Fish, and Zsh shells are supported. The default shell is Bash.

Take the Bash command as an example. Run the **eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"** command in Terminal to enable autocomplete.

```
eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

Run the **ma-cli auto-completion Zsh** or **ma-cli auto-completion Fish** command to view the autocomplete command in Zsh or Fish.

Available Commands

```
$ ma-cli auto-completion -h
Usage: ma-cli auto-completion [OPTIONS] [[Bash|Zsh|Fish]]

  Auto complete ma-cli command in terminal.

  Example:

  # print bash auto complete command to terminal
  ma-cli auto-completion Bash

Options:
  -H, -h, --help  Show this message and exit.
```

```
# By default, the autocomplete command for Bash is displayed.

$ ma-cli auto-completion

Tips: please paste following shell command to your terminal to activate auto completion.

[ OK ] eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"

# After the preceding command is executed, autocomplete has been enabled on the terminal.

$ eval "$(_MA_CLI_COMPLETE=bash_source ma-cli)"
```

```
# The autocomplete command for Fish is displayed.
$ ma-cli auto-completion Fish
Tips: please paste following shell command to your terminal to activate auto completion.

[ OK ] eval (env _MA_CLI_COMPLETE=fish_source ma-cli)
```

1.9.4 ma-cli Authentication

Overview

- VMs and personal computers require the configuration of authentication. Both a username and password (default) and an AK/SK can be used for authentication.
- When using an account for authentication, specify a username and password. When using an IAM account for authentication, specify an account, username, and password.

- In ModelArts notebook, you do not need to manually configure authentication because an agency is used for authentication by default.
- If you have configured authentication in ModelArts notebook, the specified authentication is preferentially used.

  NOTE

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

CLI Parameters

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

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

Table 1-26 Authentication CLI parameters

Parameter	Type	Mandatory	Description
-auth / --auth	String	No	Authentication mode, which can be **PWD** (username and password) or **AKSK** (AK/SK). The default value is **PWD**.
-rp / --region-profile	String	No	ModelArts region configuration file
-a / --account	String	No	IAM tenant account, which needs to be specified when authentication using an IAM account is used. It is required in authentication using a username and password.
-u / --username	String	No	Username, which is a username or an IAM username for authentication using an account or an IAM account. It is required in authentication using a username and password.
-p / --password	String	No	Password, which is required in authentication using a username and password

Parameter	Type	Mandatory	Description
-ak / --access-key	String	No	Access key, which is required in authentication using an AK/SK
-sk / --secret-key	String	No	Secret key, which is required in authentication using an AK/SK
-r / --region	String	No	Region name. If this parameter is left blank, the value of the **REGION_NAME** environment variable will be used by default.
-pi / --project-id	String	No	Project ID. If this parameter is left blank, the **region** value (default) or the value of the **PROJECT_ID** environment variable will be used.
-P / --profile	String	No	Authentication configuration, which defaults to **DEFAULT**
-C / --config-file	String	No	Local path to the configuration file, which defaults to **~/.modelarts/ma-cli-profile.yaml**

Authentication Using Username and Password

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

NOTE

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

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

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

Authentication Using an AK/SK

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

CAUTION

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

For example, you need to replace **${access key}** with your access key.


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

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

1.9.5 ma-cli image Commands for Building Images

The **ma-cli image** command can be used to obtain registered images, obtain or load image creation templates, create images using Dockerfiles, obtain or clear image creation caches, register or deregister images, and debug whether images can be used in notebook instances. For details, run the **ma-cli image -h** command.

Commands for Creating an Image

```
$ ma-cli image -h
Usage: ma-cli image [OPTIONS] COMMAND [ARGS]...
  Obtain registered images, register or unregister images, debug images, and create images in Notebook.

Options:
  -H, -h, --help  Show this message and exit.

Commands:
  add-template, at  List build-in dockerfile templates.
  build             Build docker image in Notebook.
  debug             Debug SWR image as a Notebook in ECS.
  df                Query disk usage.
  get-image, gi     Query registered image in ModelArts.
  get-template, gt  List build-in dockerfile templates.
  prune             Prune image build cache.
  register          Register image to ModelArts.
  unregister        Unregister image from ModelArts.
```

Table 1-27 Commands for creating an image

Command	Description
get-template	Obtain an image creation template.
add-template	Load an image creation template.
get-image	Obtain registered ModelArts images.
register	Register SWR images with ModelArts image management.
unregister	Deregister a registered image from ModelArts image management.
build	Build an image using a Dockerfile (only supported in ModelArts Notebook).

Command	Description
df	Obtain image creation cache, which can only be used in ModelArts notebook.
prune	Clear image creation cache, which can only be used in ModelArts notebook.
debug	Debug an SWR image on an ECS to check whether the image can be used in ModelArts notebook. (Only the ECSs with Docker installed can be used.)

Using ma-cli image get-template to Query an Image Building Template

ma-cli provides some common image building templates, which contain the guidance for developing Dockerfiles on ModelArts notebook.

```
$ ma-cli image get-template -h
Usage: ma-cli image get-template [OPTIONS]

  List build-in dockerfile templates.

  Example:

  # List build-in dockerfile templates
  ma-cli image get-template [--filer <filter_info>]  [--page-num <yourPageNum>] [--page-size
<yourPageSize>]

Options:
  --filter TEXT                  filter by keyword.
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -D, --debug                    Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT             CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                 Show this message and exit.
(PyTorch-1.4) [ma-user work]$
```

Table 1-28 Parameters

Parameter	Data Type	Mandatory	Description
--filter	String	No	Template name keyword for filtering templates.
-pn / --page-num	Int	No	Image page index. The default value is page 1.
-ps / --page-size	Int	No	Number of images displayed on each page. The default value is **20**.

Example: Obtain an image building template.

```
ma-cli image get-template
```



```
(PyTorch-1.8) [ma-user work]$ma-cli image get-template
Template Name                                    Description
-----------------------------------------------  ----------------------------------------------------------------------------------------------------
customize_from_ubuntu_18.04_to_modelarts         Add ma-user, apt install packages and create a new conda environment with pip based on scratch ubuntu 18.04
upgrade_current_notebook_apt_packages            Install apt packages like ffmpeg, gcc-8, g++-8 based on current Notebook image
migrate_3rd_party_image_to_modelarts             General template for migrating your own or open source image to ModelArts
migrate_official_torch_110_cu113_image_to_modelarts  Reconstructing and migrating the official torch 1.10.0 with cuda11.3 image to ModelArts
build_handwritten_number_inference_application   Create a new AI application, used to generate an image to deploy and infer in ModelArts
update_dli_image_pip_package                     Install pip packages based on DLI image
forward_compat_cuda_11_image_to_modelarts        Migrate and forward compat cuda-11.x to ModelArts by upgrading only user-mode CUDA components
```

Using ma-cli image add-template to Load an Image Building Template

The **add-template** command is used to load image templates to a specified folder. By default, the path where the current command is located is used,

for example, **${current_dir}/.ma/${template_name}/**. You can also use **--dest** to specify the path. If a template folder with the same name already exists in the target path, use the **--force | -f** parameter to forcibly overwrite the existing template folder.

```
$ ma-cli image add-template -h
Usage: ma-cli image add-template [OPTIONS] TEMPLATE_NAME

  Add buildin dockerfile templates into disk.

  Example:

  # List build-in dockerfile templates
  ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts --force

Options:
  --dst TEXT          target save path.
  -f, --force         Override templates that has been installed.
  -D, --debug         Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT  CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help      Show this message and exit.
```

Table 1-29 Parameters

Parameter	Data Type	Mandatory	Description
--dst	String	No	Target path for loading a template. The current path is used by default.
-f / --force	Bool	No	Whether to forcibly overwrite an existing template with the same name. By default, the template is not overwritten.

Example: Load the **customize_from_ubuntu_18.04_to_modelarts** image building template.

```
ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image add-template customize_from_ubuntu_18.04_to_modelarts
[ OK ] Successfully add configuration template [ customize_from_ubuntu_18.04_to_modelarts ] under folder [ /home/ma-user/work/.ma/customize_from_ubuntu_18.04_to_modelarts ]
```

Using ma-cli image get-image to Query Registered ModelArts Images

A path to a base image is provided in a Dockerfile typically. Public images and SWR public or private images can be obtained from open-source image

repositories such as Docker Hub. ma-cli allows you to obtain ModelArts preset images, registered images, and their SWR addresses.

```
$ma-cli image get-image -h
Usage: ma-cli image get-image [OPTIONS]

  Get registered image list.

  Example:

  # Query images by image type and only image id, show name and swr_path
  ma-cli image get-image --type=DEDICATED

  # Query images by image id
  ma-cli image get-image --image-id ${image_id}

  # Query images by image type and show more information
  ma-cli image get-image --type=DEDICATED -v

  # Query images by image name
  ma-cli image get-image --filter=torch

Options:
  -t, --type [BUILD_IN|DEDICATED|ALL]
                                  Image type(default ALL)
  -f, --filter TEXT               Image name to filter
  -v, --verbose                   Show detailed information on image.
  -i, --image-id TEXT             Get image details by image id
  -n, --image-name TEXT           Get image details by image name
  -wi, --workspace-id TEXT        The workspace where you want to query image(default "0")
  -pn, --page-num INTEGER RANGE   Specify which page to query  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query  [x>=1]
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```

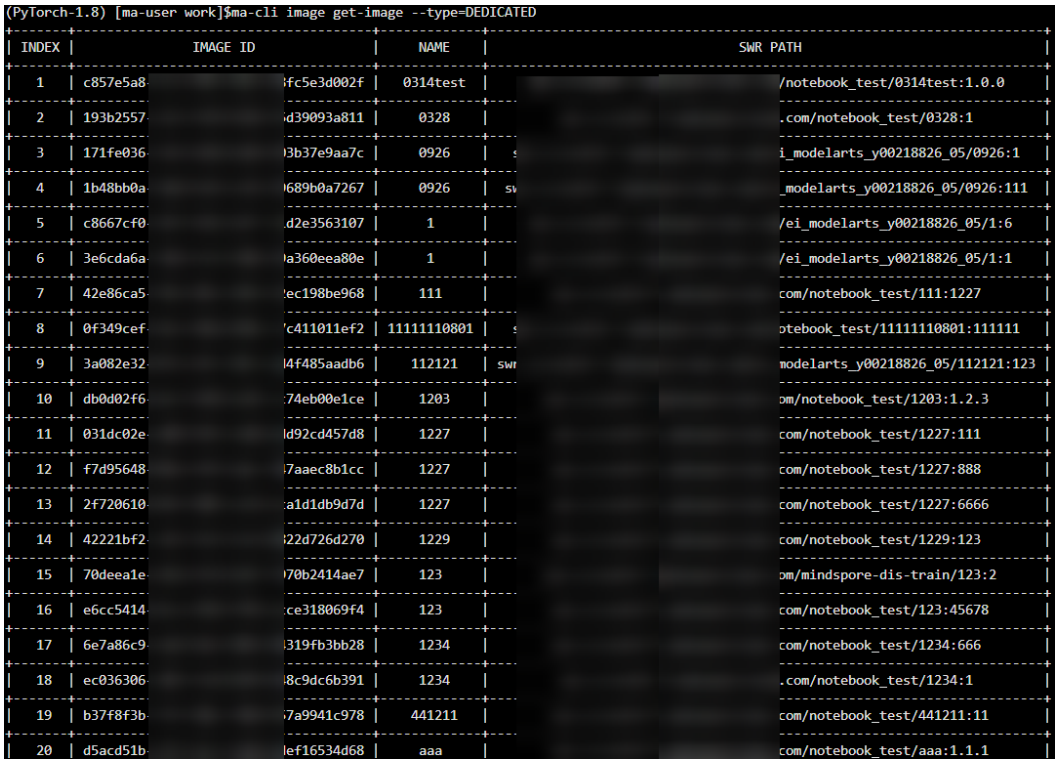
Table 1-30 Parameters

Parameter	Data Type	Mandatory	Description
-t / --type	String	No	Type of the images to be obtained. The options are **BUILD_IN**, **DEDICATED**, and **ALL**. • **BUILD_IN**: preset images • **DEDICATED**: custom images registered with ModelArts • **ALL**: all images
-f / --filter	String	No	Image name keyword for filtering images.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.
-i / --image-id	String	No	ID of the image whose details are to be obtained.
-n / --image-name	String	No	Name of the image whose details are to be obtained.

Parameter	Data Type	Mandatory	Description
-wi / --workspace-id	String	No	Workspace in which the image information is to be obtained.
-pn / --page-num	Int	No	Image page index. The default value is page 1.
-ps / --page-size	Int	No	Number of images displayed on each page. The default value is **20**.

Example: Obtain custom images registered with ModelArts.

```
ma-cli image get-image --type=DEDICATED
```



Using ma-cli image build to Build an Image in ModelArts Notebook

Run the **ma-cli image build** command to build an image based on a specified Dockerfile. This command is available only on ModelArts notebook instances.

```
$ ma-cli image build -h
Usage: ma-cli image build [OPTIONS] FILE_PATH

  Build docker image in Notebook.

  Example:

  # Build a image and push to SWR
```



```
 ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1

 # Build a image and push to SWR, dockerfile context path is current dir
 ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr my_organization/
my_image:0.0.1 -context .

 # Build a local image and save to local path and OBS
 ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile --target ./build.tar --obs-
path obs://bucket/object --swr-path my_organization/my_image:0.0.1

Options:
 -t, --target TEXT       Name and optionally a tag in the 'name:tag' format.
 -swr, --swr-path TEXT   SWR path without swr endpoint, eg:organization/image:tag.  [required]
 --context DIRECTORY     build context path.
 -arg, --build-arg TEXT  build arg for Dockerfile.
 -obs, --obs-path TEXT   OBS path to save local built image.
 -f, --force             Force to overwrite the existing swr image with the same name and tag.
 -C, --config-file PATH  Configure file path for authorization.
 -D, --debug             Debug Mode. Shows full stack trace when error occurs.
 -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
 -H, -h, --help          Show this message and exit.
```

Table 1-31 Parameters

Parameter	Data Type	Mandatory	Description
FILE_PATH	String	Yes	Path where the Dockerfile is stored.
-t / --target	String	No	Local path for storing the generated TAR package. The current directory is used by default.
-swr / --swr-path	String	Yes	SWR image name, which is in the format of "organization/image_name:tag". This parameter can be omitted when a TAR package is saved for building an image.
--context	String	No	Context used for copying data when a Dockerfile is used.
-arg / --build-arg	String	No	Parameter for building an image. If there are multiple parameters, run **--build-arg VERSION=18.04 --build-arg ARCH=X86_64**.
-obs / --obs-path	String	No	OBS path for automatically uploading the generated TAR package.
-f / --force	Bool	No	Whether to forcibly overwrite an existing SWR image with the same name. By default, the SWR image is not overwritten.

Example: Build an image in ModelArts notebook.

```
ma-cli image build .ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile -swr
notebook_test/my_image:0.0.1
```


In this command, **.ma/customize_from_ubuntu_18.04_to_modelarts/Dockerfile** is the path where the Dockerfile is stored, and **notebook_test/my_image:0.0.1** is the SWR path of the new image.

Using ma-cli image df to Obtain Image Building Caches in ModelArts Notebook

Run the **ma-cli image df** command to obtain image creation caches. This command is available only on ModelArts notebook instances.

```
$ ma-cli image df -h
Usage: ma-cli image df [OPTIONS]

  Query disk usage used by image-building in Notebook.

  Example:

  # Query image disk usage
  ma-cli image df

Options:
  -v, --verbose      Show detailed information on disk usage.
  -D, --debug        Debug Mode. Shows full stack trace when error occurs.

  -h, -H, --help     Show this message and exit.
```

Table 1-32 Parameters

Parameter	Data Type	Mandatory	Description
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: View all image caches in ModelArts notebook.

```
ma-cli image df
```



```
(PyTorch-1.8) [ma-user work]$ma-cli image df
ID                                                              RECLAIMABLE     SIZE            LAST ACCESSED
iwrrwsi9pdcjafe1ij6d0r918                                       true            98.50MB
cp52c4q81ud2abu2vp7sj5vyt                                       true            1.04MB
4jbo6v06r2wl575ddq3w8g12e                                       true            139.68kB
ojdjw5mok71s1nh2cauant051                                       true            86.86kB
k2jm6g061n5twmz7gmonmqjsh                                       true            16.55kB
efu5kwgig1ve44fe7smbrncnh*                                      true            8.19kB
uzikwqk5taxnslvajm14jrbje*                                      true            4.10kB
2g8p0qcb014g3qva7ucawkv87*                                      true            4.10kB
Reclaimable:    99.80MB
Total:          99.80MB
```

Example: Obtain details about the disk space occupied by image caches.

```
ma-cli image df --verbose
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image df --verbose
ID:             iwrrwsi9pdcjafe1ij6d0r918
Created at:     2023-03-28 12:23:28.353759532 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           98.50MB
Description:    pulled from swr                 .com/atelier/ubuntu:18.04@sha256:b5874                                  ea65d84f
Usage count:    1
Last used:      2023-03-28 12:23:28.37337776 +0000 UTC
Type:           regular

ID:             cp52c4q81ud2abu2vp7sj5vyt
Parents:        iwrrwsi9pdcjafe1ij6d0r918
Created at:     2023-03-28 12:23:28.366910223 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           1.04MB
Description:    pulled from swr.                .com/atelier/ubuntu:18.04@sha256:b5874                                  2e235eea65d84f
Usage count:    1
Last used:      2023-03-28 12:23:28.38560437 +0000 UTC
Type:           regular

ID:             4jbo6v06r2wl575ddq3w8g12e
Parents:        k2jm6g061n5twmz7gmonmqjsh
Created at:     2023-03-28 12:23:30.681643727 +0000 UTC
Mutable:        false
Reclaimable:    true
Shared:         false
Size:           139.68kB
Description:    mount / from exec /bin/sh -c default_user=$(getent passwd 1000 | awk -F ':' '{print $1}') || echo "uid: 1000 does not exist" &&     default_group=$(getent
up 100 | awk -F ':' '{print $1}') || echo "gid: 100 does not exist" &&     if [ ! -z ${default_user} ] && [ ${default_user} != "ma-user" ]; then         userdel -r ${defau
user};     fi &&     if [ ! -z ${default_group} ] && [ ${default_group} != "ma-group" ]; then         groupdel -f ${default_group};     fi &&     groupadd -g 100 ma-group
useradd -d /home/ma-user -m -u 1000 -g 100 -s /bin/bash ma-user &&     chmod -R 750 /home/ma-user
Usage count:    2
Last used:      2023-03-28 12:25:39.149080471 +0000 UTC
Type:           regular
```

Using ma-cli image prune to Clear Image Building Caches in ModelArts Notebook

Run the **ma-cli image prune** command to clear image creation caches. This command is available only on ModelArts notebook instances.

```
$ ma-cli image prune -h
Usage: ma-cli image prune [OPTIONS]

  Prune image build cache by image-building in Notebook.

  Example:

  # Prune image build cache
  ma-cli image prune

Options:
  -ks, --keep-storage INTEGER  Amount of disk space to keep for cache below this limit (in MB) (default: 0).
  -kd, --keep-duration TEXT    Keep cache newer than this limit, support second(s), minute(m) and hour(h)
(default:   0s).
  -v, --verbose                Show more verbose output.
  -D, --debug                  Debug Mode. Shows full stack trace when error occurs.
  -h, -H, --help               Show this message and exit.
```


Table 1-33 Parameters

Parameter	Data Type	Mandatory	Description
-ks / --keep-storage	Int	No	Size of the cache to be retained, in MB. The default value is **0**, indicating that all caches will be cleared.
-kd / --keep-duration	String	No	Duration in which the caches are to be retained. The unit can be **s** (second), **m** (minute), or **h** (hour). The default value is **0s**, indicating that all caches will be cleared.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Retain 1 MB of image caches not to be cleared.

```
ma-cli image prune -ks 1
```

```
(PyTorch-1.8) [ma-user work]$ma-cli image prune -ks 1
ID                                                                      RECLAIMABLE     SIZE    LAST ACCESSED
uzikwqk5taxnslvajm14jrbje*                                              true            4.10kB
4jbo6v06r2wl575ddq3w8g12e                                               true            139.68kB
k2jm6g061n5twmz7gmonmqjsh                                               true            16.55kB
ojdjw5mok71s1nh2cauant051                                               true            86.86kB
cp52c4q81ud2abu2vp7sj5vyt                                               true            1.04MB
iwrrwsi9pdcjafe1ij6d0r918                                               true            98.50MB
Total:  99.79MB
```

Using ma-cli image register to Register an SWR Image with ModelArts Image Management

After an image is debugged, run the **ma-cli image register** command to register it with ModelArts image management so that it can be used in ModelArts.

```
$ma-cli image register -h
Usage: ma-cli image register [OPTIONS]

  Register image to ModelArts.

  Example:

  # Register image into ModelArts service
  ma-cli image register --swr-path=xx

  # Share SWR image to DLI service
  ma-cli image register -swr xx -td

  # Register image into ModelArts service and specify architecture to be 'AARCH64'
  ma-cli image register --swr-path=xx --arch AARCH64

Options:
  -swr, --swr-path TEXT          SWR path without swr endpoint, eg:organization/image:tag.  [required]
  -a, --arch [X86_64|AARCH64]    Image architecture (default: X86_64).
  -s, --service [NOTEBOOK|MODELBOX]
                                 Services supported by this image(default NOTEBOOK).
  -rs, --resource-category [CPU|GPU|ASCEND]
                                 The resource category supported by this image (default: CPU and GPU).
  -wi, --workspace-id TEXT       The workspace to register this image (default: "0").
  -v, --visibility [PUBLIC|PRIVATE]
```



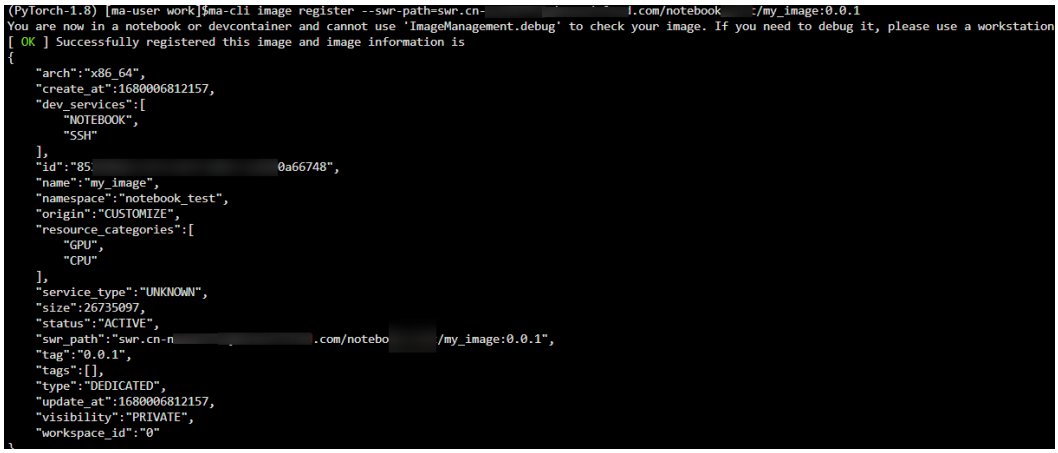
```
                        PUBLIC: every user can use this image. PRIVATE: only image owner can use this
image (Default: PRIVATE).
  -td, --to-dli                 Register swr image to DLI, which will share SWR image to DLI service.
  -d, --description TEXT          Image description (default: "").
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                   Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT             CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                Show this message and exit.
```

Table 1-34 Parameters

Parameter	Data Type	Mandatory	Description
-swr / --swr-path	String	Yes	SWR path to the image to be registered.
-a / --arch	String	No	Architecture of the image to be registered. The value can be **X86_64** or **AARCH64**. The default value is **X86_64**.
-s / --service	String	No	Service type of the image to be registered. The value can be **NOTEBOOK** or **MODELBOX**. The default value is **NOTEBOOK**. You can also specify multiple values, for example, **-s NOTEBOOK -s MODELBOX**.
-rs / --resource-category	String	No	Resource type that can be used by the image to be registered. The default values are **CPU** and **GPU**.
-wi / --workspace-id	String	No	Workspace to which the image is to be registered. The default workspace ID is **0**.
-v / --visibility	Bool	No	Visibility of the image to be registered. The value can be **PRIVATE** (visible only to the image owner) or **PUBLIC** (visible to all users). The default value is **PRIVATE**.
-td / --to-dli	Bool	No	Whether to register the image with DLI.
-d/ --description	String	No	Description of the image. By default, this parameter is left blank.

Example: Register an SWR image with ModelArts.

```
ma-cli image register --swr-path=xx
```

Using ma-cli image unregister to Deregister an Image

Run the **ma-cli image unregister** command to deregister an image from ModelArts.

```
$ ma-cli image unregister -h
Usage: ma-cli image unregister [OPTIONS]

  Unregister image from ModelArts.

  Example:

  # Unregister image
  ma-cli image unregister --image-id=xx

  # Unregister image and delete it from swr
  ma-cli image unregister --image-id=xx -d

Options:
  -i, --image-id TEXT     Unregister image details by image id.  [required]
  -d, --delete-swr-image  Delete the image from swr.
  -C, --config-file PATH  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help          Show this message and exit.
```

Table 1-35 Parameters

Parameter	Data Type	Mandatory	Description
-i / -image-id	String	Yes	ID of the image to be deregistered.
-d / --delete-swr-image	Bool	No	Whether to delete a deregistered SWR image. It is disabled by default.

Debugging an SWR Image on an ECS

ma-cli allows you to debug an SWR image on an ECS to determine whether the image can be used in a ModelArts development workspace.


```
ma-cli image debug -h
Usage: ma-cli image debug [OPTIONS]

  Debug SWR image as a Notebook in ECS.

  Example:

  # Debug cpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region=

  # Debug gpu notebook image
  ma-cli image debug --swr-path=xx --service=NOTEBOOK --region=  --gpu

Options:
  -swr, --swr-path TEXT           SWR path without SWR endpoint, eg:organization/image:tag.  [required]
  -r, --region TEXT               Region name.  [required]
  -s, --service [NOTEBOOK|MODELBOX]
                                  Services supported by this image(default NOTEBOOK).
  -a, --arch [X86_64|AARCH64]     Image architecture(default X86_64).
  -g, --gpu                       Use all gpus to debug.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help                  Show this message and exit.
```

Table 1-36 Parameters

Parameter	Data Type	Mandatory	Description
-swr / --swr-path	String	Yes	SWR path to the image to be debugged.
-r / --region	String	Yes	Region where the image to be debugged is located.
-s / --service	String	No	Service type of the image to be debugged. The value can be **NOTEBOOK** or **MODELBOX**. The default value is **NOTEBOOK**.
-a / --arch	String	No	Architecture of the image to be debugged. The value can be **X86_64** or **AARCH64**. The default value is **X86_64**.
-g / --gpu	Bool	No	Whether to use GPUs for debugging. It is disabled by default.

1.9.6 ma-cli ma-job Commands for Training Jobs

Run the **ma-cli ma-job** command to submit training jobs, obtain training job logs, events, used AI engines, and resource specifications, and stop training jobs.

```
$ ma-cli ma-job -h
Usage: ma-cli ma-job [OPTIONS] COMMAND [ARGS]...

  ModelArts job submission and query job details.
```



```
Options:
  -h, -H, --help  Show this message and exit.

Commands:
  delete       Delete training job by job id.
  get-engine   Get job engines.
  get-event    Get job running event.
  get-flavor   Get job flavors.
  get-job      Get job details.
  get-log      Get job log details.
  get-pool     Get job engines.
  stop         Stop training job by job id.
  submit       Submit training job.
```

Table 1-37 Commands supported by training jobs

Command	Description
get-job	Obtain ModelArts training jobs and their details.
get-log	Obtain runtime logs of a ModelArts training job.
get-engine	Obtain ModelArts AI engines for training.
get-event	Obtain ModelArts training job events.
get-flavor	Obtain ModelArts resource specifications for training.
get-pool	Obtain ModelArts resource pools dedicated for training.
stop	Stop a ModelArts training job.
submit	Submit a ModelArts training job.
delete	Delete a training job with a specified job ID.

Using ma-cli ma-job get-job to Obtain a ModelArts Training Job

Run the **ma-cli ma-job get-job** command to obtain training jobs or details about a specific job.

```
$ ma-cli ma-job get-job -h
Usage: ma-cli ma-job get-job [OPTIONS]

  Get job details.

  Example:

  # Get train job details by job name
  ma-cli ma-job get-job -n ${job_name}

  # Get train job details by job id
  ma-cli ma-job get-job -i ${job_id}

  # Get train job list
  ma-cli ma-job get-job --page-size 5 --page-num 1

Options:

  -i, --job-id TEXT               Get training job details by job id.
  -n, --job-name TEXT             Get training job details by job name.
  -pn, --page-num INTEGER         Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [1<=x<=50]
```

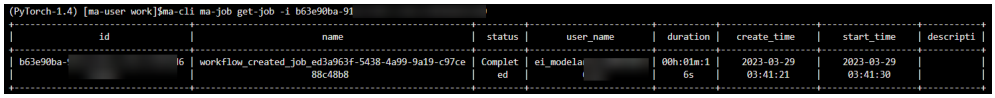


```
-v, --verbose          Show detailed information about training job details.
-C, --config-file TEXT    Configure file path for authorization.
-D, --debug            Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT     CLI connection profile to use. The default profile is "DEFAULT".
-h, -H, --help         Show this message and exit.
```

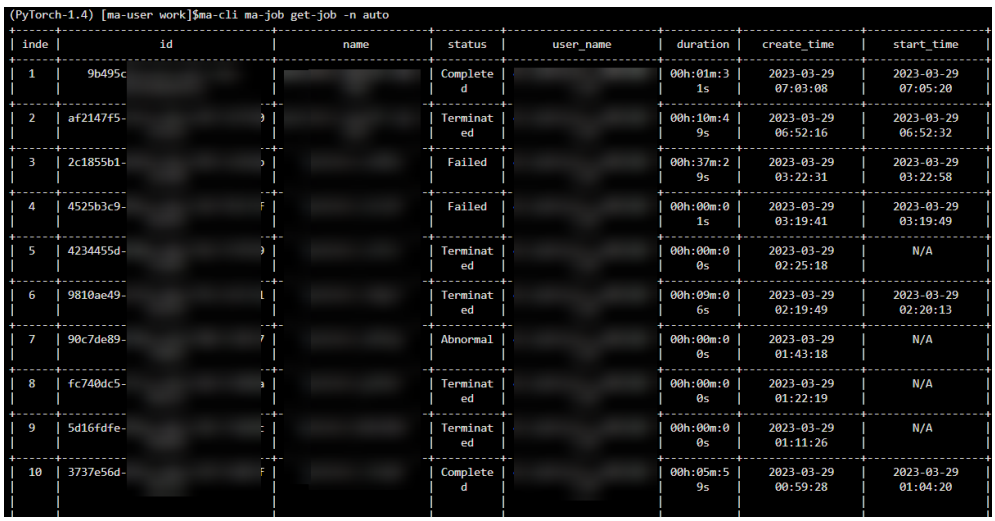
Table 1-38 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	No	ID of the job whose details are to be obtained.
-n / --job-name	String	No	Name of the job to be queried or name keyword used to filter training jobs.
-pn / --page-num	Int	No	Page number. The default value is **1**.
-ps / --page-size	Int	No	Number of training jobs displayed on each page. The default value is **10**.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

- Example: Obtain a training job with a specified job ID.
  ma-cli ma-job get-job -i *b63e90xxx*



- Example: Filter training jobs by job name keyword **auto**.
  ma-cli ma-job get-job -n *auto*



Using ma-cli ma-job submit to Submit a ModelArts Training Job

Run the **ma-cli ma-job submit** command to submit a ModelArts training job.

When running this command, use the **YAML_FILE** parameter to specify the path to the configuration file of the target job. If this parameter is not specified, the

configuration file is empty. The configuration file is in YAML format, and its parameters are values of **OPTIONS** in the command. If you specify both the **YAML_FILE** and the **OPTIONS** parameters, the **OPTIONS** value will overwrite the same items in the configuration file.

```
$ma-cli ma-job submit -h
Usage: ma-cli ma-job submit [OPTIONS] [YAML_FILE]...

  Submit training job.

  Example:

  ma-cli ma-job submit --code-dir obs://your_bucket/code/
                      --boot-file main.py
                      --framework-type PyTorch
                      --working-dir /home/ma-user/modelarts/user-job-dir/code
                      --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
                      --data-url obs://your_bucket/dataset/
                      --log-url obs://your_bucket/logs/
                      --train-instance-type modelarts.vm.cpu.8u
                      --train-instance-count 1

Options:
  --name TEXT                   Job name.
  --description TEXT            Job description.
  --image-url TEXT              Full swr custom image path.
  --uid TEXT                    Uid for custom image (default: 1000).
  --working-dir TEXT            ModelArts training job working directory.
  --local-code-dir TEXT         ModelArts training job local code directory.
  --user-command TEXT           Execution command for custom image.
  --pool-id TEXT                Dedicated pool id.
  --train-instance-type TEXT    Train worker specification.
  --train-instance-count INTEGER  Number of workers.
  --data-url TEXT               OBS path for training data.
  --log-url TEXT                OBS path for training log.
  --code-dir TEXT               OBS path for source code.
  --output TEXT                 Training output parameter with OBS path.
  --input TEXT                  Training input parameter with OBS path.
  --env-variables TEXT          Env variables for training job.
  --parameters TEXT             Training job parameters (only keyword parameters are supported).
  --boot-file TEXT              Training job boot file path behinds `code_dir`.
  --framework-type TEXT         Training job framework type.
  --framework-version TEXT      Training job framework version.
  --workspace-id TEXT           The workspace where you submit training job(default "0")
  --policy [regular|economic|turbo|auto]
                                Training job policy, default is regular.
  --volumes TEXT                Information about the volumes attached to the training job.
  -q, --quiet                   Exit without waiting after submit successfully.
  -C, --config-file PATH        Configure file path for authorization.
  -D, --debug                   Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT            CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                Show this message and exit.
```

Table 1-39 Parameters

Parameter	Data Type	Mandatory	Description
YAML_FILE	String	No	Configuration file of a training job. If this parameter is not specified, the configuration file is empty.

Parameter	Data Type	Mandatory	Description
--code-dir	String	Yes	OBS path to the training source code.
--data-url	String	Yes	OBS path to the training data.
--log-url	String	Yes	OBS path to training logs.
--train-instance-count	String	Yes	Number of training job instances. The default value is **1**, indicating a single node.
--boot-file	String	No	Boot file specified when you use a preset command to submit a training job. This parameter can be omitted when you use a custom image or a custom command to submit a training job.
--name	String	No	Training job name.
--description	String	No	Description of a training job.
--image-url	String	No	SWR URL of a custom image, which is in the format of "organization/image_name:tag".
--uid	String	No	UID of the custom image. The default value is **1000**.
--working-dir	String	No	Work directory where an algorithm is executed.
--local-code-dir	String	No	Local directory of the training container to which the algorithm code directory is downloaded.
--user-command	String	No	Command for executing a custom image. The directory must be under **/home**. When **code-dir** is prefixed with **file://**, this parameter does not take effect.
--pool-id	String	No	Resource pool ID selected for a training job. To view the resource pool ID, log in to the **ModelArts console**. In the navigation pane on the left, choose **Resource Management** > **Dedicated Compute Resources** > **Resource Pools** or **Resource Management** > **Dedicated Resource Pool**.
--train-instance-type	String	No	Resource flavor selected for a training job.

Parameter	Data Type	Mandatory	Description
--output	String	No	Training output. After this parameter is specified, the training job will upload the output directory of the training container corresponding to the specified output parameter in the training script to a specified OBS path. To specify multiple parameters, use **--output output1=obs://bucket/output1 --output output2=obs://bucket/output2**.
--input	String	No	Training input. After this parameter is specified, the training job will download the data from OBS to the training container and transfer the data storage path to the training script through the specified parameter. To specify multiple parameters, use **--input data_path1=obs://bucket/data1 --input data_path2=obs://bucket/data2**.
--env-variables	String	No	Environment variables input during training. To specify multiple parameters, use **--env-variables ENV1=env1 --env-variables ENV2=env2**.
--parameters	String	No	Training input parameters. To specify multiple parameters, use **--parameters "--epoch 0 --pretrained"**.
--framework-type	String	No	Framework type selected for a training job.
--framework-version	String	No	Framework version selected for a training job.
-q / --quiet	Bool	No	Whether to exit directly without printing the job status synchronously after a training job is submitted.
--workspace-id	String	No	Workspace where a training job is deployed. The default value is **0**.
--policy	String	No	Training resource flavor mode. The options are **regular**, **economic**, **turbo**, and **auto**.

Parameter	Data Type	Mandatory	Description
--volumes	String	No	EFS disks to be mounted. To specify multiple parameters, use **--volumes**. "local_path=/xx/yy/zz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/" -volumes "local_path=/xxx/yyy/zzz;read_only=false;nfs_server_path=xxx.xxx.xxx.xxx:/"

Example: Submitting a Training Job Based on a Preset ModelArts Image

Submit a training job by specifying the **OPTIONS** parameter.

```
ma-cli ma-job submit --code-dir obs://your-bucket/mnist/code/ \
                --boot-file main.py \
                --framework-type PyTorch \
                --working-dir /home/ma-user/modelarts/user-job-dir/code \
                --framework-version pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```

Example of **train.yaml** using a preset image:

```
# Example .ma/train.yaml (preset image)
# pool_id: pool_xxxx
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
working-dir: /home/ma-user/modelarts/user-job-dir/code
framework-type: PyTorch
framework-version: pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64
boot-file: main.py
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
output:
   - name: output_dir
     obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
   - name: data_url
     obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
   - epoch: 10
   - learning_rate: 0.01
   - pretrained:
```



```
##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
  - efs:
      local_path: /xx/yy/zz
      read_only: false
      nfs_server_path: xxx.xxx.xxx.xxx:/
```

Example: Using a Custom Image to Create a Training Job

Submit a training job by specifying the **OPTIONS** parameter.

```
ma-cli ma-job submit --image-url atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e \
                --code-dir obs://your-bucket/mnist/code/ \
                --user-command "export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH &&
cd /home/ma-user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/
python main.py" \
                --data-url obs://your-bucket/mnist/dataset/MNIST/ \
                --log-url obs://your-bucket/mnist/logs/ \
                --train-instance-type modelarts.vm.cpu.8u \
                --train-instance-count 1  \
                -q
```

Example of **train.yaml** using a custom image:

```
# Example .ma/train.yaml (custom image)
image-url: atelier/pytorch_1_8:pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-
x86_64-20220926104358-041ba2e
user-command: export LD_LIBRARY_PATH=/usr/local/cuda/compat:$LD_LIBRARY_PATH && cd /home/ma-
user/modelarts/user-job-dir/code && /home/ma-user/anaconda3/envs/PyTorch-1.8/bin/python main.py
train-instance-type: modelarts.vm.cpu.8u
train-instance-count: 1
data-url: obs://your-bucket/mnist/dataset/MNIST/
code-dir: obs://your-bucket/mnist/code/
log-url: obs://your-bucket/mnist/logs/

##[Optional] Uncomment to set uid when use custom image mode
uid: 1000

##[Optional] Uncomment to upload output file/dir to OBS from training platform
output:
   - name: output_dir
     obs_path: obs://your-bucket/mnist/output1/

##[Optional] Uncomment to download input file/dir from OBS to training platform
input:
   - name: data_url
     obs_path: obs://your-bucket/mnist/dataset/MNIST/

##[Optional] Uncomment pass hyperparameters
parameters:
   - epoch: 10
   - learning_rate: 0.01
   - pretrained:

##[Optional] Uncomment to use dedicated pool
pool_id: pool_xxxx

##[Optional] Uncomment to use volumes attached to the training job
volumes:
  - efs:
      local_path: /xx/yy/zz
      read_only: false
      nfs_server_path: xxx.xxx.xxx.xxx:/
```


Using ma-cli ma-job get-log to Obtain ModelArts Training Job Logs

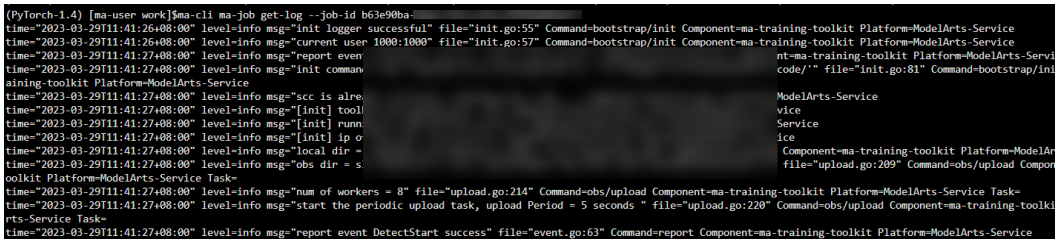
Run the **ma-cli ma-job get-log** command to obtain ModelArts training job logs.

```
$ ma-cli ma-job get-log -h
Usage: ma-cli ma-job get-log [OPTIONS]

  Get job log details.

  Example:

  # Get job log by job id
  ma-cli ma-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job details by job id.  [required]
  -t, --task-id TEXT      Get training job details by task id (default "worker-0").
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -h, -H, --help          Show this message and exit.
```

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the job whose logs are to be obtained.
-t / --task-id	String	No	ID of the task whose logs are to be obtained. The default value is **work-0**.

Example: Obtain logs of a specified training job.

```
ma-cli ma-job get-log --job-id b63e90baxxx
```



Using ma-cli ma-job get-event to Obtain ModelArts Training Job Events

Run the **ma-cli ma-job get-event** command to obtain ModelArts training job events.

```
$ ma-cli ma-job get-event -h
Usage: ma-cli ma-job get-event [OPTIONS]

  Get job running event.

  Example:

  # Get training job running event
  ma-cli ma-job get-event --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -C, --config-file TEXT  Configure file path for authorization.
```

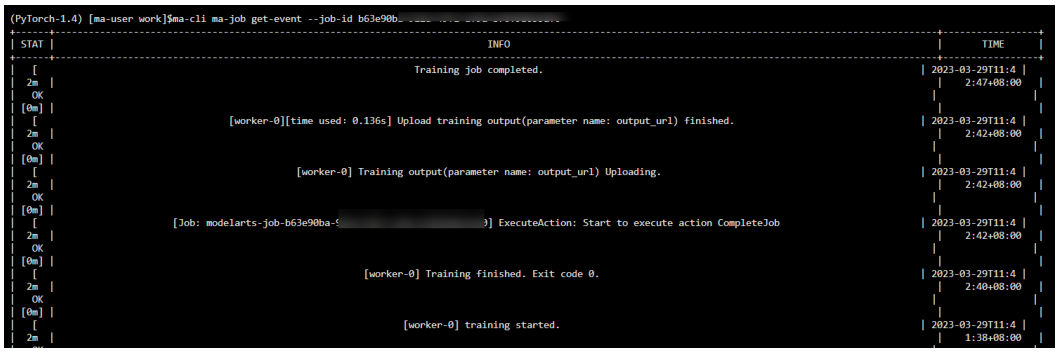


```
-D, --debug          Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT   CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help       Show this message and exit.
```

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the job whose events are to be obtained.

Example: Obtain events of a specified training job.

```
ma-cli ma-job get-event --job-id b63e90baxxx
```



Using ma-cli ma-job get-engine to Obtain the AI Engines Used by ModelArts Training Jobs

Run the **ma-cli ma-job get-engine** command to obtain the AI engines used by ModelArts training jobs.

```
$ ma-cli ma-job get-engine -h
Usage: ma-cli ma-job get-engine [OPTIONS]

  Get job engine info.

  Example:

  # Get training job engines
  ma-cli ma-job get-engine

Options:
  -v, --verbose          Show detailed information about training engines.
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug            Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT     CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help         Show this message and exit.
```

Table 1-40 Parameters

Parameter	Data Type	Mandatory	Description
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain the AI engines used by training jobs.

```
ma-cli ma-job get-engine
```

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-engine
+-------+-----------------------------------------------------------+-----------------------+----------+
| index |                         engine id                         |      engine name      | run user |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   1   |                    caffe-1.0.0-python2.7                  |         Caffe         |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   2   |                    horovod-cp36-tf-1.16.2                 |        Horovod        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   3   | horovod_0.20.0-pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   4   | horovod_0.20.0-tensorflow_2.1.0-cuda_10.1-py_3.7-ubuntu_18.04-x86_64 |        Horovod        |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   5   |                kungfu-0.2.2-tf-1.13.1-python3.6           |        KungFu         |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   6   |        mindspore_1.3.0-cuda_10.1-py_3.7-ubuntu_1804-x86_64        |          MPI          |   1102   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   7   |       mindspore_1.7.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   8   |       mindspore_1.8.0-cann_5.1.2-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   9   |       mindspore_1.9.0-cann_6.0.0-py_3.7-euler_2.8.3-aarch64       | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   10  |                    mxnet-1.2.1-python3.6                  |         MXNet         |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   11  |    optverse_0.2.0-pygrassland_1.1.0-py_3.7-ubuntu_18.04-x86_64    |          OR           |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   12  |                     pytorch-cp36-1.0.0                    |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   13  |                     pytorch-cp36-1.3.0                    |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   14  |                     pytorch-cp36-1.4.0                    |        PyTorch        |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   15  |       pytorch_1.8.0-cann_5.1.0-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   16  |        pytorch_1.8.0-cuda_10.2-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   17  |       pytorch_1.8.1-cann_5.1.2-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   18  |       pytorch_1.8.1-cann_6.0.0-py_3.7-euler_2.8.3-aarch64         | Ascend-Powered-Engine |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   19  |        pytorch_1.8.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   20  |        pytorch_1.8.2-cuda_10.2-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   21  |        pytorch_1.9.1-cuda_11.1-py_3.7-ubuntu_18.04-x86_64         |        PyTorch        |   1000   |
+-------+-----------------------------------------------------------+-----------------------+----------+
|   22  |                       ray-cp36-0.7.4                      |          Ray          |          |
+-------+-----------------------------------------------------------+-----------------------+----------+
```

Using ma-cli ma-job get-flavor to Obtain the Resource Flavors Used by ModelArts Training Jobs

Run the **ma-cli ma-job get-flavor** command to obtain the resource flavors used by ModelArts training jobs.

```
$ ma-cli ma-job get-flavor -h
Usage: ma-cli ma-job get-flavor [OPTIONS]

  Get job flavor info.

  Example:

  # Get training job flavors
  ma-cli ma-job get-flavor

Options:
  -t, --flavor-type [CPU|GPU|Ascend]
                                  Type of training job flavor.
  -v, --verbose                   Show detailed information about training flavors.
  -C, --config-file TEXT          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```


Table 1-41 Parameters

Parameter	Data Type	Mandatory	Description
-t / --flavor-type	String	No	Resource flavor type. If this parameter is not specified, all resource flavors are returned by default.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain the resource flavors and types of training jobs.

```
ma-cli ma-job get-flavor
```

```
(PyTorch-1.4) [ma-user work]$ma-cli ma-job get-flavor
+-------+-----------------------------+-----------------------------------+-------------+
| index |          flavor id          |           flavor name             | flavor type |
+-------+-----------------------------+-----------------------------------+-------------+
|   1   |    modelarts.kat1.8xlarge   | Computing NPU(8*Ascend) instance  |    Ascend   |
+-------+-----------------------------+-----------------------------------+-------------+
|   2   |    modelarts.kat1.xlarge    |  Computing NPU(Ascend) instance   |    Ascend   |
+-------+-----------------------------+-----------------------------------+-------------+
|   3   |     modelarts.vm.cpu.2u     |    Computing CPU(2U) instance     |     CPU     |
+-------+-----------------------------+-----------------------------------+-------------+
|   4   |     modelarts.vm.cpu.8u     |    Computing CPU(8U) instance     |     CPU     |
+-------+-----------------------------+-----------------------------------+-------------+
|   5   |  modelarts.vm.cpu.8u16g.119 |    Computing CPU(8U) instance     |     CPU     |
+-------+-----------------------------+-----------------------------------+-------------+
|   6   |   modelarts.vm.v100.large   |   Computing GPU(V100) instance    |     GPU     |
+-------+-----------------------------+-----------------------------------+-------------+
|   7   | modelarts.vm.v100.large.free|   Computing GPU(V100) instance    |     GPU     |
+-------+-----------------------------+-----------------------------------+-------------+
```

Using ma-cli ma-job stop to Stop a ModelArts Training Job

Run the **ma-cli ma-job stop** command to stop a training job with a specified job ID.

```
$ ma-cli ma-job stop -h
Usage: ma-cli ma-job stop [OPTIONS]

  Stop training job by job id.

  Example:

  Stop training job by job id
  ma-cli ma-job stop --job-id ${job_id}
```



```
Options:
  -i, --job-id TEXT       Get training job event by job id.  [required]
  -y, --yes               Confirm stop operation.
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

Table 1-42 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of a ModelArts training job
-y / --yes	Bool	No	Whether to forcibly stop a training job

Example: Stop a running training job.

```
ma-cli ma-job stop --job-id efd3e2f8xxx
```

1.9.7 ma-cli dli-job Commands for Submitting DLI Spark Jobs

```
$ma-cli dli-job -h
Usage: ma-cli dli-job [OPTIONS] COMMAND [ARGS]...

  DLI spark job submission and query job details.

Options:
  -h, -H, --help  Show this message and exit.

Commands:
  get-job       Get DLI spark job details.
  get-log       Get DLI spark log details.
  get-queue     Get DLI spark queues info.
  get-resource  Get DLI resources info.
  stop          Stop DLI spark job by job id.
  submit        Submit dli spark batch job.
  upload        Upload local file or OBS object to DLI resources.
```

Table 1-43 Commands for submitting DLI Spark jobs

Command	Description
get-job	Obtain DLI Spark jobs and their details.
get-log	Obtain runtime logs of a DLI Spark job.
get-queue	Obtain DLI queues.
get-resource	Obtain DLI group resources.
stop	Stop a DLI Spark job.
submit	Submit a DLI Spark job.

Command	Description
upload	Upload local files or OBS files to a DLI group.

Using ma-cli dli-job get-job to Obtain a DLI Spark Job

Run the **ma-cli dli-job get-job** command to obtain the DLI Spark jobs or details about a job.

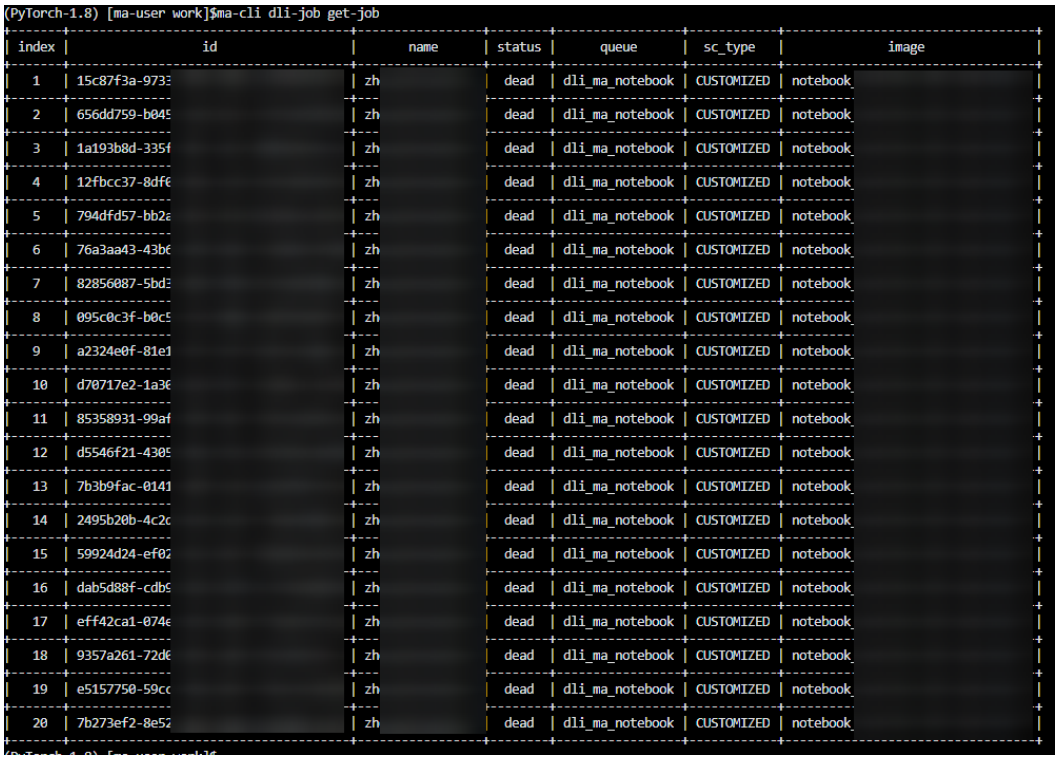
```
ma-cli dli-job get-job -h
Usage: ma-cli dli-job get-job [OPTIONS]

  Get DLI Spark details.

  Example:

  # Get DLI Spark job details by job name
  ma-cli dli-job get-job -n ${job_name}

  # Get DLI Spark job details by job id
  ma-cli dli-job get-job -i ${job_id}

  # Get DLI Spark job list
  ma-cli dli-job get-job --page-size 5 --page-num 1

Options:
  -i, --job-id TEXT              Get DLI Spark job details by job id.
  -n, --job-name TEXT            Get DLI Spark job details by job name.
  -pn, --page-num INTEGER RANGE  Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -v, --verbose                  Show detailed information about DLI Spark job details.
  -C, --config-file PATH         Configure file path for authorization.
  -D, --debug                    Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT             CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                 Show this message and exit.
```

Table 1-44 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	No	ID of the DLI Spark job whose details are to be obtained.
-n / --job-name	String	No	Name of the DLI Spark job to be queried or name keyword used to filter DLI Spark jobs.
-pn / --page-num	Int	No	Page number. The default value is **1**.
-ps / --page-size	Int	No	Number of jobs displayed on each page. The default value is **20**.
-v / --verbose	Bool	No	Whether to display detailed information. It is disabled by default.

Example: Obtain all DLI Spark jobs.


```
ma-cli dli-job get-job
```



Using ma-cli dli-job submit to Submit a DLI Spark Job

Run the **ma-cli dli-job submit** command to submit a DLI Spark job.

When running this command, use the **YAML_FILE** parameter to specify the path to the configuration file of the target job. If this parameter is not specified, the configuration file is empty. The configuration file is in YAML format, and its parameters are values of **OPTIONS** in the command. If you specify both the **YAML_FILE** and the **OPTIONS** parameters, the **OPTIONS** value will overwrite the same items in the configuration file.

Command parameters preview

```
ma-cli dli-job submit -h
Usage: ma-cli dli-job submit [OPTIONS] [YAML_FILE]...

  Submit DLI Spark job.

  Example:

  ma-cli dli-job submit  --name test-spark-from-sdk
                         --file test/sub_dli_task.py
                         --obs-bucket dli-bucket
                         --queue dli_test
                         --spark-version 2.4.5
                         --driver-cores 1
                         --driver-memory 1G
                         --executor-cores 1
                         --executor-memory 1G
                         --num-executors 1

Options:
  --file TEXT                 Python file or app jar.
  -cn, --class-name TEXT      Your application's main class (for Java / Scala apps).
```



```
--name TEXT                  Job name.
--image TEXT                 Full swr custom image path.
--queue TEXT                 Execute queue name.
-obs, --obs-bucket TEXT      DLI obs bucket to save logs.
-sv, --spark-version TEXT    Spark version.
-st, --sc-type [A|B|C]       Compute resource type.
--feature [basic|custom|ai]  Type of the Spark image used by a job (default: basic).
-ec, --executor-cores INTEGER  Executor cores.
-em, --executor-memory TEXT  Executor memory (eg. 2G/2048MB).
-ne, --num-executors INTEGER  Executor number.
-dc, --driver-cores INTEGER  Driver cores.
-dm, --driver-memory TEXT    Driver memory (eg. 2G/2048MB).
--conf TEXT                  Arbitrary Spark configuration property (eg. <PROP=VALUE>).
--resources TEXT             Resources package path.
--files TEXT                 Files to be placed in the working directory of each executor.
--jars TEXT                  Jars to include on the driver and executor class paths.
-pf, --py-files TEXT         Python files to place on the PYTHONPATH for Python apps.
--groups TEXT                User group resources.
--args TEXT                  Spark batch job parameter args.
-q, --quiet                  Exit without waiting after submit successfully.
-C, --config-file PATH       Configure file path for authorization.
-D, --debug                  Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT           CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help               Show this message and exit.
```

YAML file preview

```
# dli-demo.yaml
name: test-spark-from-sdk
file: test/sub_dli_task.py
obs-bucket: ${your_bucket}
queue: dli_notebook
spark-version: 2.4.5
driver-cores: 1
driver-memory: 1G
executor-cores: 1
executor-memory: 1G
num-executors: 1

## [Optional]
jars:
  - ./test.jar
  - obs://your-bucket/jars/test.jar
  - your_group/test.jar

## [Optional]
files:
  - ./test.csv
  - obs://your-bucket/files/test.csv
  - your_group/test.csv

## [Optional]
python-files:
  - ./test.py
  - obs://your-bucket/files/test.py
  - your_group/test.py

## [Optional]
resources:
  - name: your_group/test.py
    type: pyFile
  - name: your_group/test.csv
    type: file
  - name: your_group/test.jar
    type: jar
  - name: ./test.py
    type: pyFile
  - name: obs://your-bucket/files/test.py
    type: pyFile
```



```
## [Optional]
groups:
  - group1
  - group2
```

Example of submitting a DLI Spark job by specifying **OPTIONS**:

```
$ ma-cli dli-job submit --name test-spark-from-sdk \
                        --file test/sub_dli_task.py \
                        --obs-bucket ${your_bucket} \
                        --queue dli_test \
                        --spark-version 2.4.5 \
                        --driver-cores 1 \
                        --driver-memory 1G \
                        --executor-cores 1 \
                        --executor-memory 1G \
                        --num-executors 1
```

Table 1-45 Parameters

Parameter	Data Type	Mandatory	Description
YAML_FILE	String	No	Local path to the configuration file of a DLI Spark job. If this parameter is not specified, the configuration file is empty.
--file	String	Yes	Program entry file. It can be a local file path, an OBS path, or the name of a JAR or PyFile package that has been uploaded to the DLI resource management system.
-cn / --class_name	String	Yes	Java/Spark main class of the batch processing job.
--name	String	No	Specified job name. The value consists of a maximum of 128 characters.
--image	String	No	Path to a custom image in the format of "Organization name/Image name:Image version". This parameter is valid only when **feature** is set to **custom**. You can use this parameter with the **feature** parameter to specify a custom Spark image for running a job.
-obs / --obs-bucket	String	No	OBS bucket for storing a Spark job. Configure this parameter when you need to save jobs. It can also be used as a transit station for submitting local files to **resources**.
-sv/ --spark-version	String	No	Spark version used by a job.

Parameter	Data Type	Mandatory	Description
-st / --sc-type	String	No	If the current Spark version is 2.3.2, leave this parameter blank. If the current Spark version is 2.3.3, configure this parameter when **feature** is set to **basic** or **ai**. If this parameter is not specified, the default Spark version 2.3.2 will be used.
--feature	String	No	Job feature, indicating the type of the Spark image used by a job. The default value is **basic**. • **basic**: A base Spark image provided by DLI is used. • **custom**: A custom Spark image is used. • **ai**: An AI image provided by DLI is used.
--queue	String	No	Queue name. Set this parameter to the name of a created DLI queue. The queue must be of the common type. For details about how to obtain a queue name, see **Table 1-47**.
-ec / --executor-cores	String	No	Number of CPU cores of each Executor in a Spark application. This configuration will replace the default setting in **sc_type**.
-em / --executor-memory	String	No	Executor memory of a Spark application, for example, 2 GB or 2048 MB. This configuration will replace the default setting in **sc_type**. The unit must be provided. Otherwise, the startup fails.
-ne / --num-executors	String	No	Number of Executors in a Spark application. This configuration will replace the default setting in **sc_type**.
-dc / --driver-cores	String	No	Number of CPU cores of the Spark application driver. This configuration will replace the default setting in **sc_type**.
-dm / --driver-memory	String	No	Driver memory of a Spark application, for example, 2 GB or 2048 MB. This configuration will replace the default setting in **sc_type**. The unit must be provided. Otherwise, the startup fails.
--conf	Array of String	No	**batch** configuration. For details, see **Spark Configuration**. To specify multiple parameters, use **--conf conf1 --conf conf2**.

Parameter	Data Type	Mandatory	Description
--resources	Array of String	No	Name of a resource package, which can be a local file, OBS path, or a file that has been uploaded to the DLI resource management system. To specify multiple parameters, use **--resources resource1 --resources resource2**.
--files	Array of String	No	Name of the file package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--files file1 --files file2**.
--jars	Array of String	No	Name of the JAR package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--jars jar1 --jars jar2**.
-pf /--python-files	Array of String	No	Name of the PyFile package that has been uploaded to the DLI resource management system. You can also specify an OBS path, for example, **obs://***Bucket name***/***Package name*. Local files are also supported. To specify multiple parameters, use **--python-files py1 --python-files py2**.
--groups	Array of String	No	Resource group name. To specify multiple parameters, use **--groups group1 --groups group2**.
--args	Array of String	No	Parameters passed to the main class, that is, program parameters. To specify multiple parameters, use **--args arg1 --args arg2**.
-q / --quiet	Bool	No	Whether to exit directly without printing the job status synchronously after a DLI Spark job is submitted.

Example

- Submit a DLI Spark job using **YAML_FILE**.
  ```
  $ma-cli dli-job submit dli_job.yaml
  ```



```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit ./dli-job.yaml
[ OK ] Current DLI job id is: 01b698b8-9fd6-4a8e-bc3c-6821c6405b14
[ OK ] starting
[ OK ] running
[ OK ] success
[ OK ] Successfully submit DLI spark job [ 01b698b8-9fd6-4a8e-bc3c-6821c6405b14 ].
```

- Submit a DLI Spark job by specifying the **OPTIONS** parameter.

```
$ma-cli dli-job submit --name test-spark-from-sdk \
>                     --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                     --queue dli_ma_notebook \
>                     --spark-version 2.4.5 \
>                     --driver-cores 1 \
>                     --driver-memory 1G \
>                     --executor-cores 1 \
>                     --executor-memory 1G \
>                     --num-executors 1
```

```
(PyTorch-1.4) [ma-user work]$ma-cli dli-job submit --name test-spark-from-sdk \
>                         --file test/jumpstart-trainingjob-gallery-pytorch-sample.ipynb \
>                         --queue dli_ma_notebook \
>                         --spark-version 2.4.5 \
>                         --driver-cores 1 \
>                         --driver-memory 1G \
>                         --executor-cores 1 \
>                         --executor-memory 1G \
>                         --num-executors 1
[ OK ] Current DLI job id is: ae856c20-e9ae-49ca-8409-7a02652297b8
[ OK ] starting
```

Using ma-cli dli-job get-log to Obtain Run Logs of a DLI Spark Job

Run the **ma-cli dli-job get-log** command to obtain the run logs of a DLI Spark job.

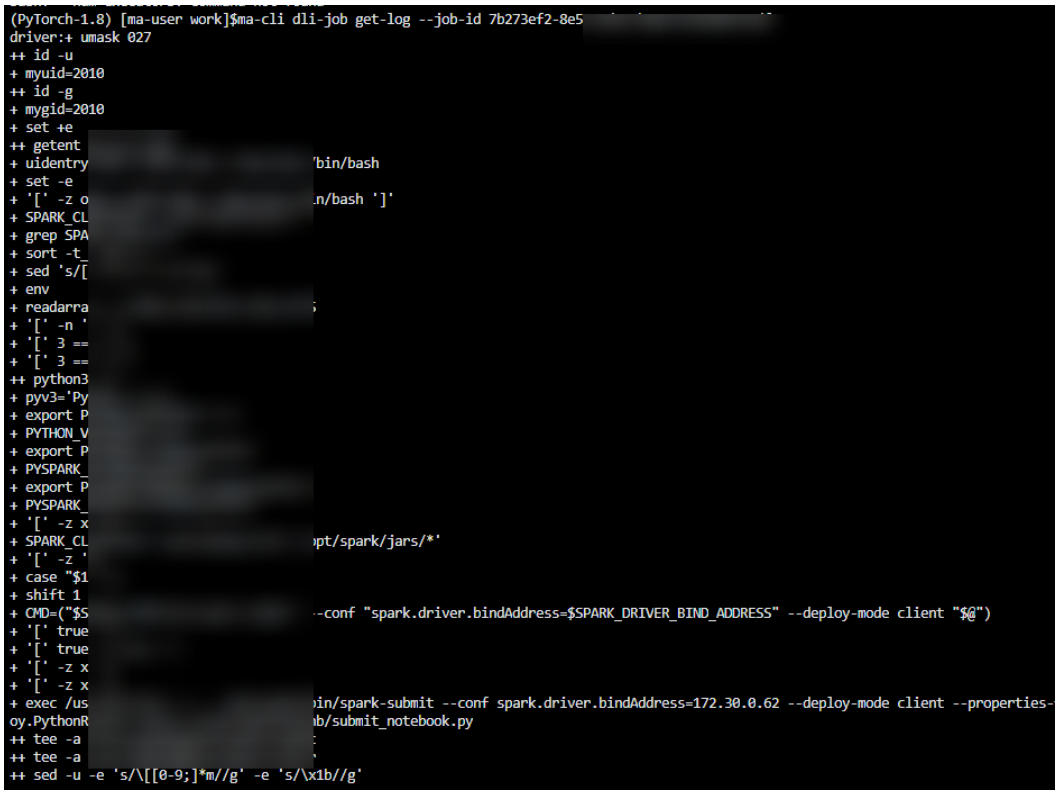
```
$ ma-cli dli-job get-log -h
Usage: ma-cli dli-job get-log [OPTIONS]

  Get DLI spark job log details.

  Example:

  # Get job log by job id
  ma-cli dli-job get-log --job-id ${job_id}

Options:
  -i, --job-id TEXT       Get DLI spark job details by job id.  [required]
  -C, --config-file TEXT  Configure file path for authorization.
  -D, --debug             Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help          Show this message and exit.
```

Table 1-46 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	ID of the DLI Spark job whose logs are to be obtained.

Example: Obtain the run logs of a specified DLI Spark job using its ID.

```
ma-cli dli-job get-log --job-id ${your_job_id}
```

Using ma-cli dli-job get-queue to Obtain a DLI Queue

Run **ma-cli dli-job get-queue** to obtain a DLI queue.

```
ma-cli dli-job get-queue -h
Usage: ma-cli dli-job get-queue [OPTIONS]

  Get DLI queues info.

  Example:

  # Get DLI queue details by queue name
  ma-cli dli-job get-queue --queue-name $queue_name}

Options:
  -pn, --page-num INTEGER RANGE   Specify which page to query.  [x>=1]
  -ps, --page-size INTEGER RANGE  The maximum number of results for this query.  [x>=1]
  -n, --queue-name TEXT           Get DLI queue details by queue name.
  -t, --queue-type [sql|general|all]
                                  DLI queue type (default "all").
  -tags, --tags TEXT              Get DLI queues by tags.
  -C, --config-file PATH          Configure file path for authorization.
  -D, --debug                     Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT              CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                  Show this message and exit.
```

Table 1-47 Parameters

Parameter	Data Type	Mandatory	Description
-n / --queue-name	String	No	Name of the DLI queue to be obtained.

Parameter	Data Type	Mandatory	Description
-t / --queue-type	String	No	Type of the DLI queue to be obtained. The value can be **sql**, **general**, or **all**. The default value is **all**.
-tags / --tags	String	No	Tags of the DLI queue to be obtained.
-pn / --page-num	Int	No	DLI queue page number. The default value is **1**.
-ps / --page-size	Int	No	Number of DLI queues displayed on each page. The default value is **20**.

Example: Query information about the queue named **dli_ma_notebook**.

```
ma-cli dli-job get-queue --queue-name dli_ma_notebook
```

```
(PyTorch-1.8) [ma-user work]$ ma-cli dli-job get-queue --queue-name dli_ma_notebook
{'chargingMode': 1,
 'create_time': 1668585417422,
 'cuCount': 16,
 'cu_spec': 16,
 'description': '',
 'enterprise_project_id': '0',
 'is_success': True,
 'message': '',
 'owner': '                          ',
 'queueName': 'dli_ma_notebook',
 'queueType': 'general',
 'queue_id': 8____,
 'resource_id': '242e7af8-c9d1                          e',
 'resource_mode': 1,
 'resource_type': 'container',
 'support_spark_versions': ['2.3.2', '2.4.5', '3.1.1']}
```

Using ma-cli dli-job get-resource to Obtain DLI Group Resources

Run the **ma-cli dli-job get-resource** command to obtain details about DLI resources, such as the resource name and resource type.

```
$ ma-cli dli-job get-resource -h
Usage: ma-cli dli-job get-resource [OPTIONS]

  Get DLI resource info.

  Example:

  # Get DLI resource details by resource name
  ma-cli dli-job get-resource --resource-name ${resource_name}

Options:
  -n, --resource-name TEXT        Get DLI resource details by resource name.
  -k, --kind [jar|pyFile|file|modelFile]
                                  DLI resources type.
  -g, --group TEXT                Get DLI resources by group.
```



```
-tags, --tags TEXT            Get DLI resources by tags.
-C, --config-file TEXT        Configure file path for authorization.
-D, --debug                   Debug Mode. Shows full stack trace when error occurs.
-P, --profile TEXT            CLI connection profile to use. The default profile is "DEFAULT".
-H, -h, --help                Show this message and exit.
```

Table 1-48 Parameters

Parameter	Data Type	Mandatory	Description
-n / --resource-name	String	No	Resource name of the DLI group resources to be queried.
-k / --kind	String	No	Resource type of the DLI group resources to be queried. The type can be JAR, PyFile, file, or modelFile.
-g / --group	String	No	Group name of the DLI group resources to be queried.
-tags / --tags	String	No	Tags of the DLI group resources to be queried.

Example: Obtain information about all DLI group resources.

```
ma-cli dli-job get-resource
```


Using ma-cli dli-job upload to Upload Files to a DLI Group

Run the **ma-cli dli-job upload** command to upload local files or OBS files to a DLI group.

```
$ ma-cli dli-job upload -h
Usage: ma-cli dli-job upload [OPTIONS] PATHS...

  Upload DLI resource.

  Tips: --obs-path is need when upload local file.

  Example:

  # Upload an OBS path to DLI resource
  ma-cli dli-job upload obs://your-bucket/test.py -g test-group --kind pyFile

  # Upload a local path to DLI resource
  ma-cli dli-job upload ./test.py -g test-group -obs ${your-bucket} --kind pyFile

  # Upload local path and OBS path to DLI resource
  ma-cli dli-job upload ./test.py obs://your-bucket/test.py -g test-group -obs ${your-bucket}
```



```
Options:
  -k, --kind [jar|pyFile|file]  DLI resources type.
  -g, --group TEXT              DLI resources group.
  -tags, --tags TEXT            DLI resources tags, follow --tags `key1`=`value1`.
  -obs, --obs-bucket TEXT       OBS bucket for upload local file.
  -async, --is-async            whether to upload resource packages in asynchronous mode. The default value is
False.
  -C, --config-file TEXT        Configure file path for authorization.
  -D, --debug                   Debug Mode. Shows full stack trace when error occurs.
  -P, --profile TEXT            CLI connection profile to use. The default profile is "DEFAULT".
  -H, -h, --help                Show this message and exit.
```

Table 1-49 Parameters

Parameter	Data Type	Mandatory	Description
PATHS	String	Yes	Paths to the local files or OBS files to be uploaded to a DLI group. Multiple paths can be specified at a time.
-k / --kind	String	No	Type of the file to be uploaded, which can be JAR, PyFile, or file.
-g / --group	String	No	Name of the DLI group to which files are to be uploaded.
-tags / --tags	String	No	Tag of the files to be uploaded.
-obs / --obs-bucket	String	No	If the files to be uploaded contain a local path, specify an OBS bucket for transit.
-async / --is-async	Bool	No	Whether to asynchronously upload files. This method is recommended.

Example

- Upload local files to a DLI group.
  ma-cli dli-job upload ./test.py -obs ${your-bucket} --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload ./test.py -obs obs://[redacted] --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

- Upload OBS files to a DLI group.
  ma-cli dli-job upload obs://your-bucket/test.py --kind pyFile

  ```
  (PyTorch-1.8) [ma-user work]$ma-cli dli-job upload obs://[redacted]/test.py --kind pyFile
  [ OK ] Upload ['test.py']  successfully.
  ```

Using ma-cli dli-job stop to Stop a DLI Spark Job

Run the **ma-cli dli-job stop** command to stop a DLI Spark job.

```
$ ma-cli dli-job stop -h
Usage: ma-cli dli-job stop [OPTIONS]

  Stop DLI spark job by job id.

  Example:
```



```
 Stop training job by job id
 ma-cli dli-job stop --job-id ${job_id}

Options:
 -i, --job-id TEXT       Get DLI spark job event by job id.  [required]
 -y, --yes               Confirm stop operation.
 -C, --config-file TEXT  Configure file path for authorization.
 -D, --debug             Debug Mode. Shows full stack trace when error occurs.
 -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
 -H, -h, --help          Show this message and exit.
```

Table 1-50 Parameters

Parameter	Data Type	Mandatory	Description
-i / --job-id	String	Yes	DLI Spark job ID
-y / --yes	Bool	No	Whether to forcibly stop a specified DLI Spark job

Example

```
ma-cli dli-job stop -i ${your_job_id}
```

```
(PyTorch-1.8) [ma-user work]$ma-cli dli-job stop -i 4b[redacted]861
[ WARN ] Spark job 4b[redacted]1 will be stopped, do you want to continue (y for confirm)? [y/N]: y
[ OK ] Successfully stop spark batch job [ 4b[redacted]1 ].
```

1.9.8 Using ma-cli to Copy OBS Data

Run the **ma-cli obs-copy [SRC] [DST]** command to copy a local file to an OBS folder or an OBS file or folder to a local path.

```
$ma-cli obs-copy -h
Usage: ma-cli obs-copy [OPTIONS     ] SRC DST

 Copy file or directory between OBS and local path. Example:

 # Upload local file to OBS path
 ma-cli obs-copy ./test.zip obs://your-bucket/copy-data/

 # Upload local directory to OBS path
 ma-cli obs-copy ./test/ obs://your-bucket/copy-data/

 # Download OBS file to local path
 ma-cli obs-copy obs://your-bucket/copy-data/test.zip ./test.zip

 # Download OBS directory to local path
 ma-cli obs-copy obs://your-bucket/copy-data/ ./test/

Options:
 -d, --drop-last-dir     Whether to drop last directory when copy folder. if True, the last directory of the
source folder will not copy to the destination folder.  [default: False]
 -C, --config-file PATH  Configure file path for authorization.
 -D, --debug             Debug Mode. Shows full stack trace when error occurs.
 -P, --profile TEXT      CLI connection profile to use. The default profile is "DEFAULT".
 -H, -h, --help          Show this message and exit.
```


Table 1-51 Parameters

Parameter	Type	Mandatory	Description
-d / --drop-last-dir	Bool	No	If you specify this parameter, the last-level directory of the source folder will not be copied to the destination folder. This parameter is valid only for copying folders.

Examples

Upload a file to OBS.

```
$ ma-cli obs-copy ./test.csv obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/test.csv ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Upload a folder to **obs://${your_bucket}/test-copy/data/**.

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/
[ OK ] local src path: [ /home/ma-user/work/data/ ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Upload a folder to **obs://${your_bucket}/test-copy/** with **--drop-last-dir** specified.

```
$ ma-cli obs-copy /home/ma-user/work/data/ obs://${your_bucket}/test-copy/ --drop-last-dir
[ OK ] local src path: [ /home/ma-user/work/data ]
[ OK ] obs dst path: [ obs://${your_bucket}/test-copy/ ]
```

Download a folder from OBS to a local disk.

```
$ ma-cli obs-copy obs://${your_bucket}/test-copy/ ~/work/test-data/
[ OK ] obs src path: [ obs://${your_bucket}/test-copy/ ]
[ OK ] local dst path: [ /home/ma-user/work/test-data/ ]
```

1.10 Using MoXing Commands in a Notebook Instance

1.10.1 MoXing Framework Functions

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

NOTE

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

Why mox.file

Use Python to open a local file.

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

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

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

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

The following Python code lists a local path:

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

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

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

Importing MoXing Framework

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

Import the MoXing Framework module:

```
import moxing as mox
```

Related Notes

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

```
import logging

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

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

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

Data Downloading Acceleration

You can use MoXing Framework to accelerate data downloading for training jobs created using ModelArts preset images. This is suitable when the number of files

ranges from 1 million to 10 million, a single large file, or the file size is greater than 20 GB.

1. Log in to the **ModelArts console** and perform the following operations as required:
   - New version: Choose **Model Build** > **Model Training**.
   - Old version: Choose **Model Training** > **Training Jobs**.
2. In the upper right corner of the page, click **Create Training Job**, and set **MA_MOXING_FWVER=2.2.8.0aa484aa** in **Environment Variable** to install the latest MoXing Framework. For details about other parameters, see **Creating a Production Training Job**. Then, you can use **moxing.file.copy_parallel** in the training job script to accelerate data downloading.
3. Set **MOX_C_ACCELERATE=0** in **Environment Variable** to disable data downloading acceleration if it is not needed.

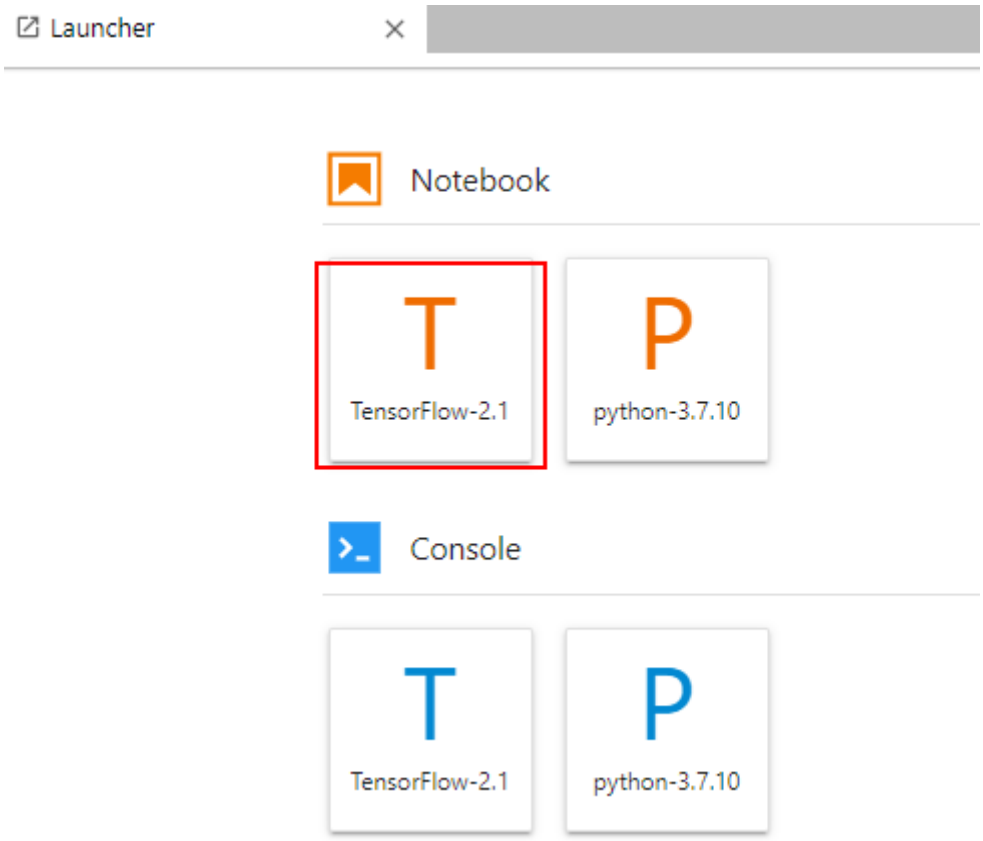
1.10.2 Using MoXing in Notebook

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

Logging In to ModelArts and Creating a Notebook Instance

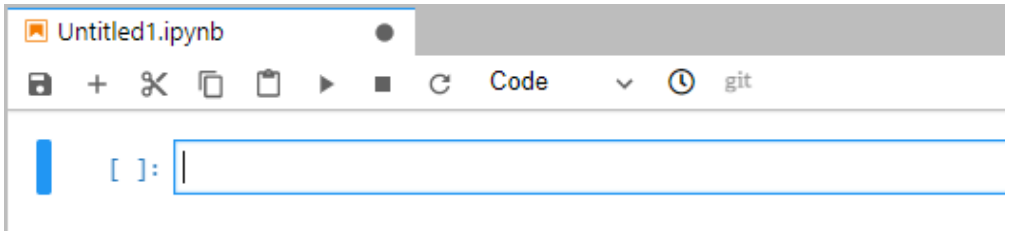
1. Log in to the **ModelArts console** and perform the following operations as required:
   - New version: Choose **Model Build** > **Notebook**.
   - Old version: Choose **Development Workspace** > **Notebook**.
2. Click **Create Notebook** in the upper right corner. On the **Create Notebook** page that is displayed, create a notebook instance by referring to **Creating a Notebook Instance**.
3. Once your notebook instance starts running, choose **Access Environment** in the **Operation** column. Then, click **Access** next to **JupyterLab Access** to open the JupyterLab notebook page.
4. On the **Launcher** page of JupyterLab, for example, click **TensorFlow** to create a file for encoding.

Figure 1-140 Selecting an AI engine



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

Figure 1-141 Encoding page



Calling mox.file.

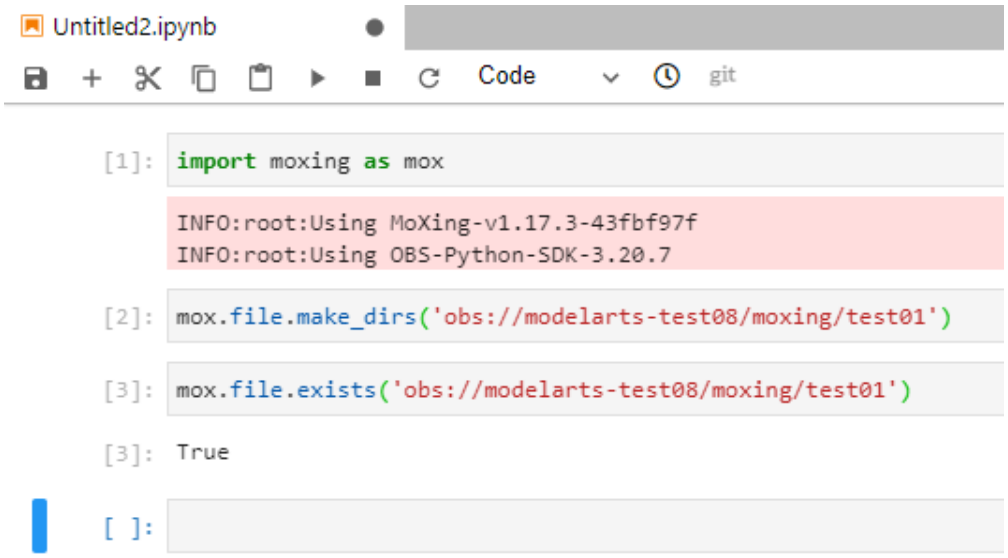
Enter the following code to implement the following simple functions:

1. Introduce MoXing Framework.
   ```
   import moxing as mox
   ```
2. Create the **test01** folder in the existing **modelarts-test08/moxing** directory.
   ```
   mox.file.make_dirs('obs://modelarts-test08/moxing/test01')
   ```
3. Check whether the **test01** folder exists. If the folder exists, the preceding operation is successful.
   ```
   mox.file.exists('obs://modelarts-test08/moxing/test01')
   ```

Figure 1-142 shows the result. Note that each time you enter a line of code, click **Run**. You can also go to OBS Console and check whether the **test01** folder has

been created in the **modelarts-test08/moxing** directory. For more common MoXing operations, see **Sample Code for Common Operations**.

Figure 1-142 Example



Copying Data to OBS

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

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

1.10.3 Mapping Between mox.file and Local APIs and Switchover

API Mapping

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

Table 1-52 API mapping

Python (Local File Operation API)	mox.file (MoXing File Operation API)	tf.gfile (TensorFlow File Operation API)
glob.glob	mox.file.glob	tf.gfile.Glob

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

1.10.4 Sample Code for Common Operations

Data Reads and Writes

- Read an OBS file.

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

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

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

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

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

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


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

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

- Read an OBS file in binary mode.

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

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

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

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

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

- Write a string to a file.

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

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

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

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

  NOTE

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

- Append content to an OBS file.

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

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

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

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

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

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

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

List

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

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

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

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

  Call the following code:

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

  The following list is returned:

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

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

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

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

  Call the following code:

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

  The following list is returned:

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

Create a Folder

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

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

Query

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

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


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

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

  NOTE

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

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

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

  NOTE

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

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

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

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

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

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

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

- Obtain the **stat** information about an OBS file or folder. The **stat** information contains the following:
  - **length**: file size
  - **mtime_nsec**: creation timestamp
  - **is_directory**: whether the path is a folder

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

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

Delete

- Delete an OBS file.

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

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

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

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

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


Move and Copy

- Move an OBS file or folder. The move operation is implemented by copying and deleting data.
  - Move an OBS file to another OBS file. For example, move **obs://bucket_name/obs_file.txt** to **obs://bucket_name/obs_file_2.txt**.
    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```
    NOTE

    The move and copy operation must be performed in the same bucket.
  - Move an OBS file to a local file. For example, move **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.
    ```
    import moxing as mox
    mox.file.rename('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```
  - Move a local file to an OBS file. For example, move **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.
    ```
    import moxing as mox
    mox.file.rename('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    ```
  - Move a local file to another local file. For example, move **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**. This operation is equivalent to **os.rename**.
    ```
    import moxing as mox
    mox.file.rename('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```

  You can move folders in the same way. If you move a folder, all content in the folder is moved recursively.
- Copy a file. **mox.file.copy** can be used to perform operations only on files. To perform operations on folders, use **mox.file.copy_parallel**.
  - Copy an OBS file to another OBS path. For example, copy **obs://bucket_name/obs_file.txt** to **obs://bucket_name/obs_file_2.txt**.
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', 'obs://bucket_name/obs_file_2.txt')
    ```
  - Copy an OBS file to a local path, that is, download an OBS file. For example, download **obs://bucket_name/obs_file.txt** to **/tmp/obs_file.txt**.
    ```
    import moxing as mox
    mox.file.copy('obs://bucket_name/obs_file.txt', '/tmp/obs_file.txt')
    ```
  - Copy a local file to OBS, that is, upload an OBS file. For example, upload **/tmp/obs_file.txt** to **obs://bucket_name/obs_file.txt**.
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', 'obs://bucket_name/obs_file.txt')
    ```
  - Copy a local file to another local path. This operation is equivalent to **shutil.copyfile**. For example, copy **/tmp/obs_file.txt** to **/tmp/obs_file_2.txt**.
    ```
    import moxing as mox
    mox.file.copy('/tmp/obs_file.txt', '/tmp/obs_file_2.txt')
    ```
- Copy a folder. **mox.file.copy_parallel** can be used to perform operations only on folders. To perform operations on files, use **mox.file.copy**.
  - Copy an OBS file to another OBS path. For example, copy **obs://bucket_name/sub_dir_0** to **obs://bucket_name/sub_dir_1**.
    ```
    import moxing as mox
    mox.file.copy_parallel('obs://bucket_name/sub_dir_0', 'obs://bucket_name/sub_dir_1')
    ```
  - Copy an OBS folder to a local path, that is, download an OBS folder. For example, download **obs://bucket_name/sub_dir_0** to **/tmp/sub_dir_0**.


```
import moxing as mox
mox.file.copy_parallel('obs://bucket_name/sub_dir_0', '/tmp/sub_dir_0')
```

- Copy a local folder to OBS, that is, upload an OBS folder. For example, upload **/tmp/sub_dir_0** to **obs://bucket_name/sub_dir_0**.

```
import moxing as mox
mox.file.copy_parallel('/tmp/sub_dir_0', 'obs://bucket_name/sub_dir_0')
```

- Copy a local folder to another local path. This operation is equivalent to **shutil.copytree**. For example, copy **/tmp/sub_dir_0** to **/tmp/sub_dir_1**.

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

1.10.5 Sample Code for Advanced MoXing Usage

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

Closing a File After File Reading Is Completed

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

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

Reading or Writing an OBS File Using pandas

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

```
import pandas as pd
import moxing as mox
with mox.file.File("obs://bucket_name/b.txt", "r") as f:
  csv = pd.read_csv(f)
```

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

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

Reading an Image Using a File Object

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

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

Modify the code as follows:

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


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

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

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

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

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

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

```
import os
import moxing as mox
import pandas as pd
from pandas.io import pytables
from pandas.core.generic import NDFrame

to_hdf_origin = getattr(NDFrame, 'to_hdf')
read_hdf_origin = getattr(pytables, 'read_hdf')


def to_hdf_override(self, path_or_buf, key, **kwargs):
  tmp_dir = '/cache/hdf_tmp'
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  to_hdf_origin(self, local_file, key, **kwargs)
  mox.file.copy(local_file, path_or_buf)
  mox.file.remove(local_file)


def read_hdf_override(path_or_buf, key=None, mode='r', **kwargs):
  tmp_dir = '/cache/hdf_tmp'
  file_name = os.path.basename(path_or_buf)
  mox.file.make_dirs(tmp_dir)
  local_file = os.path.join(tmp_dir, file_name)
  mox.file.copy(path_or_buf, local_file)
  result = read_hdf_origin(local_file, key, mode, **kwargs)
  mox.file.remove(local_file)
  return result

setattr(NDFrame, 'to_hdf', to_hdf_override)
setattr(pytables, 'read_hdf', read_hdf_override)
setattr(pd, 'read_hdf', read_hdf_override)
```

Use MoXing to Enable h5py.File to Support OBS

```
import os
import h5py
import numpy as np
import moxing as mox

h5py_File_class = h5py.File

class OBSFile(h5py_File_class):
  def __init__(self, name, *args, **kwargs):
```



```
    self._tmp_name = None
    self._target_name = name
    if name.startswith('obs://'):
      self._tmp_name = name.replace('/', '_')
      if mox.file.exists(name):
        mox.file.copy(name, os.path.join('cache', 'h5py_tmp', self._tmp_name))
      name = self._tmp_name

    super(OBSFile, self).__init__(name, *args, **kwargs)

  def close(self):
    if self._tmp_name:
      mox.file.copy(self._tmp_name, self._target_name)

    super(OBSFile, self).close()


setattr(h5py, 'File', OBSFile)

arr = np.random.randn(1000)
with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  f.create_dataset("default", data=arr)

with h5py.File('obs://bucket/random.hdf5', 'r') as f:
  print(f.require_dataset("default", dtype=np.float32, shape=(1000,)))
```


2 Using Workflows for Low-Code AI Development

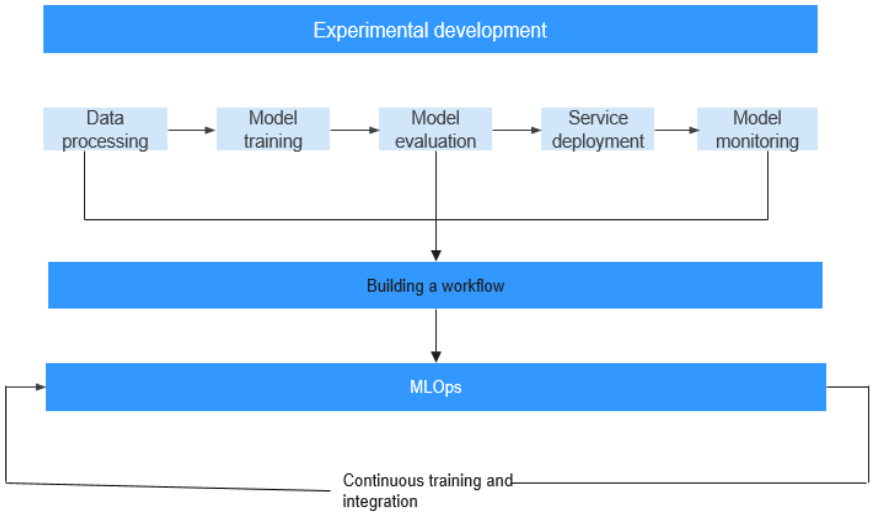
2.1 What Is Workflow?

MLOps Overview

Understanding MLOps is essential before learning about workflows.

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

Figure 2-1 MLOps



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

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

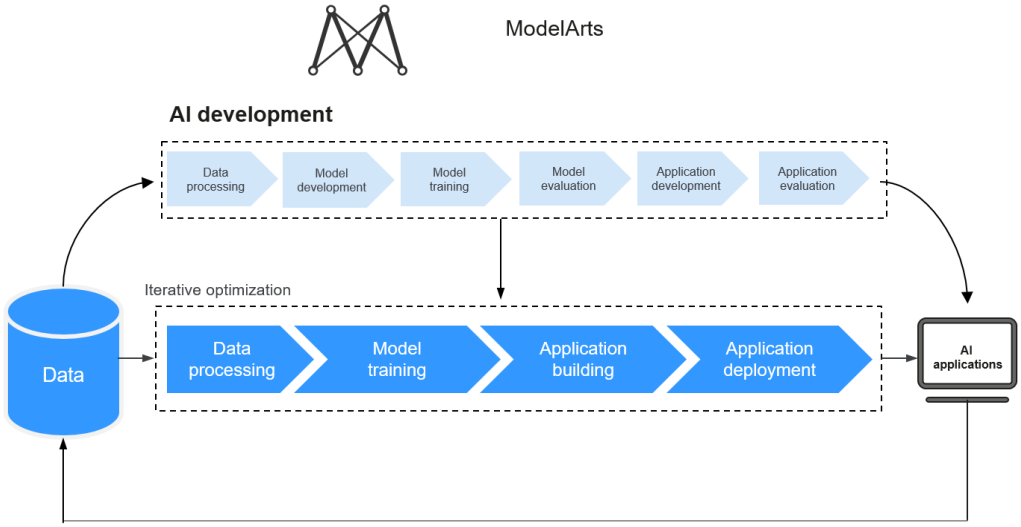
- Process analysis: Accumulated industry sample pipelines help you quickly design AI projects and processes.
- Process definition and redefinition: You can use pipelines to quickly define AI projects and design workflows for model training and release for inference.
- Resource allocation: You can use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

Workflow Overview

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

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

Figure 2-2 Workflow



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

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

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

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

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

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

Workflow Development State

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

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

Workflow Running State

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

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

A running workflow supports:

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

Workflow Components

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

Sample Workflows

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

2.2 Managing a Workflow

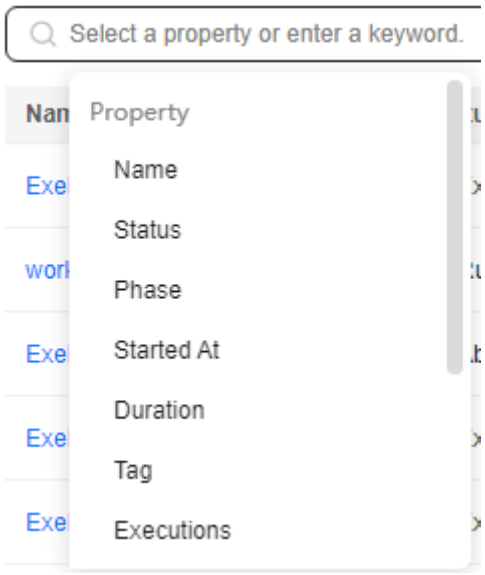
2.2.1 Searching for a Workflow

Procedure

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

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

   Figure 2-3 Property

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

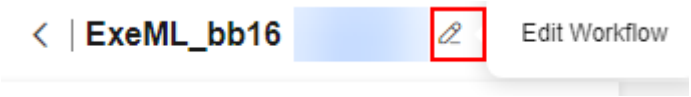
Editing a Workflow Name and Tag

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

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

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

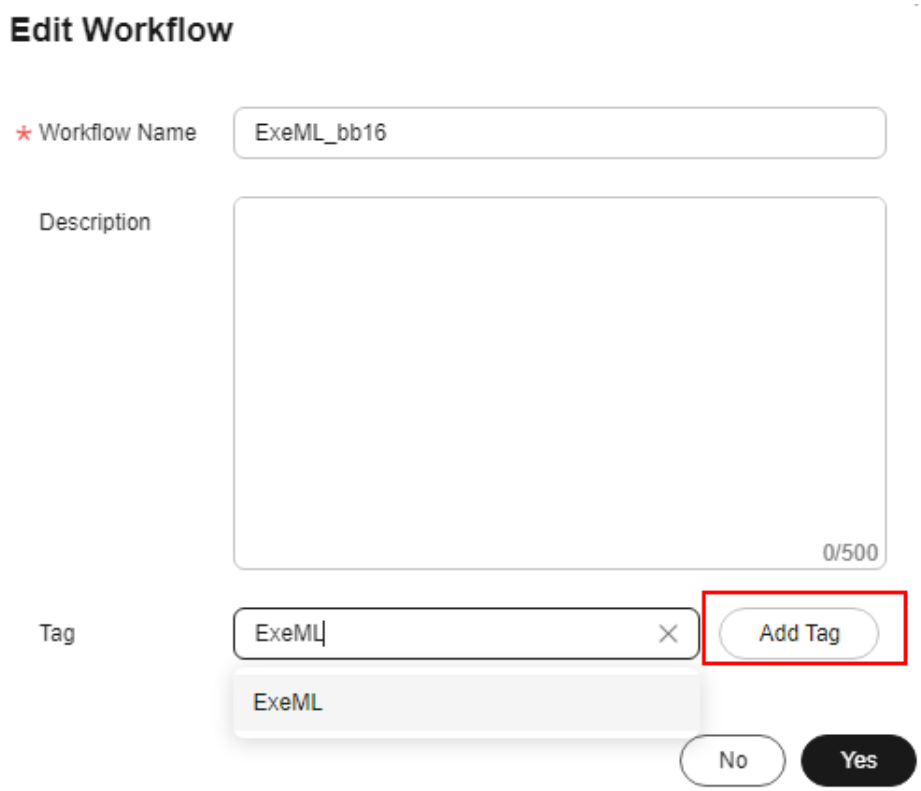
Figure 2-4 Editing a workflow



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

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

Figure 2-5 Adding a tag



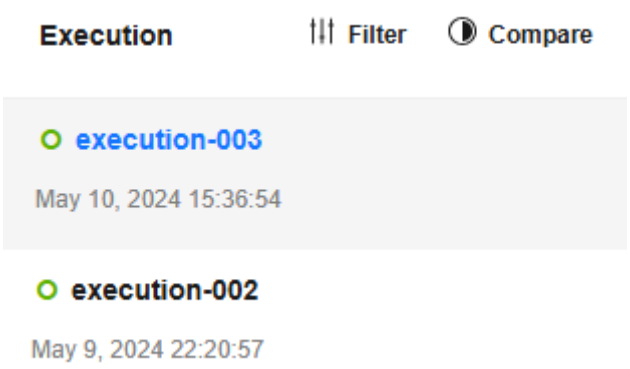
5. Workflows with tags can be filtered by tag in the search box.

2.2.2 Viewing the Running Records of a Workflow

All runtime statuses of a workflow are recorded.

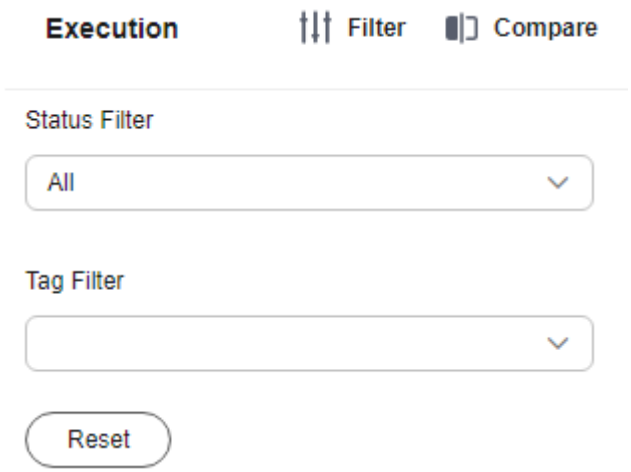
1. On the workflow list page, click the name of the target workflow.
2. On the workflow details page, view all runtime records of the workflow in the left pane.

Figure 2-6 Viewing execution records



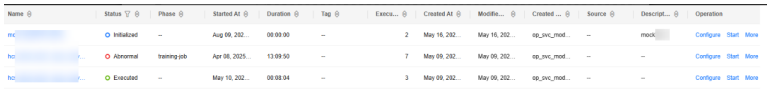
3. Delete or edit the runtime records, or rerun the workflow.
   - To delete a runtime record that is no longer needed, click **Delete**. In the displayed dialog box, click **Yes**.
   - To distinguish a runtime record from others, click **Edit Tag** to add a tag to it.
   - To rerun the workflow, click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

     Figure 2-7 Filtering

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

     Figure 2-8 Comparison

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

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

2.2.3 Managing a Workflow

Starting a Workflow

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

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

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

  NOTE

  After a workflow is started, fees will be generated. Pay attention to the instance status. After a workflow is complete, stop it in a timely manner to avoid unnecessary fees.

Stopping a Workflow

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

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

- Workflow list page

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

  NOTE

  - The **Stop** button is available only for a workflow that is running.
  - After a workflow is stopped, the associated training jobs and real-time services are also stopped.

Copying a Workflow

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

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

NOTE

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

Deleting a Workflow

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

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

- Workflow list page
  a. Click **More** in the **Operation** column and select **Delete**.
  b. In the **Delete Workflow** dialog box, confirm the information, enter **DELETE**, and click **OK**.
- Runtime configuration page
  a. Click **Delete Workflow** in the upper right corner of the page.
  b. In the **Delete Workflow** dialog box, confirm the information, enter **DELETE**, and click **OK**.

NOTE

- Deleted workflows cannot be recovered.
- The instances and generated files related to the deletion operation will not be deleted. The running workflow will stop the running instances.
- When a workflow is deleted, its associated real-time services and training jobs are not deleted. You need to manually delete them on the **Model Training** > **Training Jobs** page and the **Model Deployment** > **Real-Time Services** page.

2.2.4 Retrying, Stopping, or Running a Workflow Phase

Retrying, Stopping, or Proceeding a Workflow Phase

- Retrying a phase

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

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

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

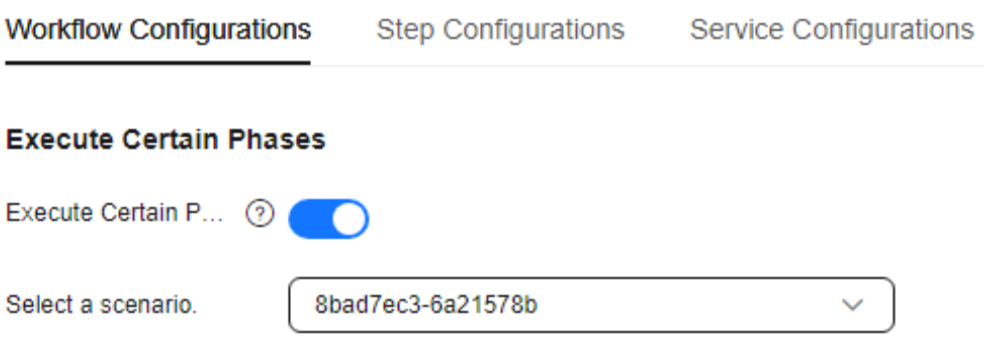
Running Specific Workflow Phases

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

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

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

   Figure 2-9 Partial execution



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

2.3 Workflow Development Command Reference

2.3.1 Core Concepts of Workflow Development

Workflow

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

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

Table 2-1 Workflow

Parameter	Description	Mandatory	Data Type
name	Workflow name. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter.	Yes	str
desc	Workflow description	Yes	str
steps	Phases contained in a workflow	Yes	list[Step]
storages	Unified storage objects	No	Storage or list[Storage]
policy	Workflow configuration policy, which is used for partial execution	No	Policy

Step

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

Table 2-2 Step

Parameter	Description	Mandatory	Data Type
name	Phase name. The name can contain a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-), and must start with a letter.	Yes	str
title	Title of a phase, which is displayed in the DAG. If this parameter is not configured, the name is displayed by default.	No	str
step_type	Type of a phase, which determines the function of the phase	Yes	enum
inputs	Inputs of a phase	No	AbstractInput or list[AbstractInput]
outputs	Outputs of a phase	No	AbstractOutput or list[AbstractOutput]
properties	Node properties	No	dict

Parameter	Description	Mandatory	Data Type
policy	Phase execution policy, which includes the phase scheduling interval, the phase execution timeout interval, and the option to skip phase execution	No	StepPolicy
depend_steps	List of dependency phases. This parameter determines the DAG structure and phase execution sequence.	No	Step or list[Step]

Table 2-3 StepPolicy

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

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

Data

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

- Actual data objects, which are specified when you create a workflow
  - Dataset: defines existing datasets. This object is used for data labeling and model training.
  - LabelTask: defines existing labeling jobs. This object is usually used for data labeling and dataset version release.
  - OBSPath: defines an OBS path. This object is used for model training, dataset import, and model import.
  - ServiceData: defines an existing service. This object is used only for service update.
  - SWRImage: defines an existing SWR path. This object is used for model registration.

  - GalleryModel: defines a model subscribed from AI Gallery. This object is used for model registration.
- Placeholder data objects, which are specified when a workflow is running
  - DatasetPlaceholder: defines datasets to be specified when a workflow is running. This object is used for data labeling and model training.
  - LabelTaskPlaceholder: defines labeling jobs to be specified when a workflow is running. This object is used for data labeling and dataset version release.
  - OBSPlaceholder: defines an OBS path to be specified when a workflow is running. This object is used for model training, dataset import, and model import.
  - ServiceUpdatePlaceholder: defines existing services to be specified when a workflow is running. This object is used only for service update.
  - SWRImagePlaceholder: defines an SWR path to be specified when a workflow is running. This object is used for model registration.
  - ServiceInputPlaceholder: defines model information required for service deployment when a workflow is running. This object is used only for service deployment and update.
  - DataSelector: supports multiple data types. Currently, this object can be used only on the job phase (only OBS or datasets are supported).
- Data selection object:

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

Table 2-4 Dataset

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

Example:

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

NOTE

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

Table 2-5 LabelTask

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

Example:

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

Table 2-6 OBSPath

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

Example:

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

Table 2-7 ServiceData

Parameter	Description	Mandatory	Data Type
service_id	Service ID	Yes	str

Example:

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

Table 2-8 SWRImage

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

Example:

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


Table 2-9 GalleryModel

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

Example:

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

Table 2-10 DatasetPlaceholder

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

Example:

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

Table 2-11 OBSPlaceholder

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

Parameter	Description	Mandatory	Data Type
default	Default value of a data object	No	OBSPath

Example:

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

Table 2-12 LabelTaskPlaceholder

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

Example:

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

Table 2-13 ServiceUpdatePlaceholder

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

Example:

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

Table 2-14 SWRImagePlaceholder

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str

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

Example:

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

Table 2-15 ServiceInputPlaceholder

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

Example:

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

Table 2-16 DataSelector

Parameter	Description	Mandatory	Data Type
name	Name	Yes	str
data_type_list	Supported data types. Currently, only **obs** and **dataset** are supported.	Yes	list

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

Example:

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

Table 2-17 DataConsumptionSelector

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

Example:

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

2.3.2 Configuring Workflow Parameters

Description

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

Parameter Overview (Placeholder)

Parameter	Description	Mandatory	Data Type
name	Parameter name, which must be globally unique.	Yes	str

Parameter	Description	Mandatory	Data Type
placeholder_type	Parameter type. The mapping between placeholder types and actual data types: PlaceholderType.INT -> int PlaceholderType.STR -> str PlaceholderType.BOOL -> bool PlaceholderType.FLOAT -> float PlaceholderType.ENUM -> Enum PlaceholderType.JSON -> dict PlaceholderType.LIST -> list • When the type is PlaceholderType.ENUM, the **enum_list** field cannot be empty. • When the type is PlaceholderType.LIST, the **placeholder_format** field cannot be empty and can only be set to **str**, **int**, **float**, or **bool**, indicating the data types in the list.	Yes	PlaceholderType
default	Default parameter value. The data type must be the same as that of **placeholder_type**.	No	Any
placeholder_format	Supported data formats. Currently, **obs**, **flavor**, **train_flavor**, **swr**, and **pacific** are supported.	No	str
delay	Whether parameters are set when the workflow is running. The default value is **False**, indicating that parameters are set before the workflow runs. If the value is **True**, parameters are set in an action of the phase where they are needed.	No	bool
description	Parameter description.	No	str
enum_list	List of enumerated values of a parameter. This parameter is mandatory only for parameters of PlaceholderType.ENUM type.	No	list
constraint	Constraints on parameters. This parameter only supports the constraints of training specifications and is not visible to you.	No	dict

Parameter	Description	Mandatory	Data Type
required	Whether the parameter is mandatory. • The default value is **True**. • This parameter cannot be set to **False** for **Delay**. This parameter is optional at the frontend during execution.	No	bool

Examples

- Integer parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_int", placeholder_type=wf.PlaceholderType.INT, default=1, description="This is an integer parameter.")
```
- String parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_str", placeholder_type=wf.PlaceholderType.STR, default="default_value", description="This is a string parameter.")
```
- Bool parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_bool", placeholder_type=wf.PlaceholderType.BOOL, default=True, description="This is a bool parameter.")
```
- Float parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_float", placeholder_type=wf.PlaceholderType.FLOAT, default=0.1, description="This is a float parameter.")
```
- Enumeration parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_enum", placeholder_type=wf.PlaceholderType.ENUM, default="a", enum_list=["a", "b"], description="This is an enumeration parameter.")
```
- Dictionary parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_dict", placeholder_type=wf.PlaceholderType.JSON, default={"key": "value"}, description="This is a dictionary parameter.")
```
- List parameter
```
from modelarts import workflow as wf
wf.Placeholder(name="placeholder_list", placeholder_type=wf.PlaceholderType.LIST, default=[1, 2], placeholder_format="int", description="This is a list parameter and its value is an integer.")
```

2.3.3 Configuring the Input and Output Paths of a Workflow

Description

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

- Input directory management: When developing a workflow, you can centrally manage all data storage paths. You can store data and configure the root directory based on your own requirements. This function orchestrates directories but does not create them.
- Output directory management: When developing a workflow, you can centrally manage all output paths. You do not need to create output

directories. Instead, you only need to configure the root path before the workflow runs and view the output data in the specified directories based on your directory orchestration rules. In addition, multiple executions of the same workflow are output to different directories, isolating data for different executions.

Common Usage

- **InputStorage** (Path concatenation)

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

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

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

- **OutputStorage** (Directory creation)

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

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

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

Advanced Usage

Storage

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

Parameter	Description	Mandatory	Data Type
name	Name.	Yes	str
title	If this parameter is left blank, the value of **name** is used by default.	No	str
description	Description.	No	str
create_dir	Whether to create a directory. The default value is **False**.	No	bool

Parameter	Description	Mandatory	Data Type
with_execution_id	Whether to combine **execution_id** when a directory is created. The default value is **False**. This parameter can be set to **True** only when **create_dir** is set to **True**.	No	bool

The following is an example:

- Implementing **InputStorage** capabilities

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

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

- Implementing **OutputStorage** capabilities

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

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

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

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

Chain call is supported for **Storage**.

The following is an example:

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


Examples

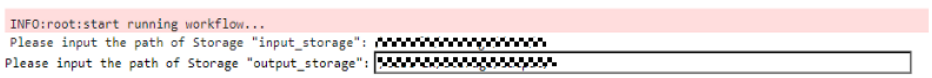
Unified storage is mainly used in the job phase. The following code uses a workflow that contains only the training phase as an example.

```
from modelarts import workflow as wf

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

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

# Use JobStep to define a training phase, and set OBS paths for storing inputs and outputs.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image Classification Training", # Title, which defaults to the value of name
    algorithm=wf.AIGalleryAlgorithm(subscription_id="subscription_ID",
item_version_id="item_version_ID"), # Algorithm used for training. In this example, an algorithm subscribed to from AI Gallery is used.
    inputs=[
        wf.steps.JobInput(name="data_url_1", data=wf.data.OBSPath(obs_path = input_storage.join("/dataset1/new.manifest"))),  # The obtained path is /root/input-data/dataset1/new.manifest.
        wf.steps.JobInput(name="data_url_2", data=wf.data.OBSPath(obs_path = input_storage.join("/dataset2/new.manifest"))),  # The obtained path is /root/input-data/dataset2/new.manifest.
    ],
    outputs=wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/model/"))), # The training output path is /root/output/Execution ID/model/.
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
                flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")
        ),
        log_export_path=wf.steps.job_step.LogExportPath(obs_url=output_storage.join("/logs/"))  # The log output path is /root/output/Execution ID/logs/.
    )# Training flavors
)

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

Configuring Root Paths in the Development State

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

Figure 2-10 Inputting root paths



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

Configuring Root Paths in the Running State

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

Figure 2-11 Configuring root paths

2.3.4 Creating Workflow Phases

2.3.4.1 Creating a Dataset Phase

Description

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

Parameter Overview

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

Table 2-18 CreateDatasetStep

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

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

Table 2-19 CreateDatasetInput

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

Table 2-20 CreateDatasetOutput

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

Table 2-21 DatasetProperties

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

Table 2-22 Importconfig

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

Table 2-23 AnnotationFormatConfig

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

Enumeration	Value
ImportTypeEnum	DIR MANIFEST

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

Examples

There are two scenarios:

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

Creating a dataset using unlabeled data

Data preparation: Store unlabeled data in an OBS folder.

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

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

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

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



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

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

Creating a dataset using labeled data with labels imported

Data preparation: Store labeled data in an OBS folder.

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

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

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

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

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

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


2.3.4.2 Creating a Dataset Labeling Phase

Description

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

Parameter Overview

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

Table 2-24 LabelingStep

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

Table 2-25 LabelingInput

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

Table 2-26 LabelingOutput

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

Table 2-27 LabelTaskProperties

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

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

Table 2-28 Label

Parameter	Description	Mandatory	Data Type
name	Tag name	No	str

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

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

Sample Code of a Dataset Labeling Phase

There are three scenarios:

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

  Scenarios:

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

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

  Data preparation: Create a dataset on the ModelArts console.

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

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

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

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



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

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

- Scenario 2: Labeling a specified job

  Scenarios:

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

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

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

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

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

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

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

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

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

```
from modelarts import workflow as wf

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

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

create_dataset = wf.steps.CreateDatasetStep(
```



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

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

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

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

2.3.4.3 Creating a Dataset Import Phase

Description

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

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

Parameter Overview

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

Table 2-29 DatasetImportStep

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

Table 2-30 DatasetImportInput

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

Table 2-31 DatasetImportOutput

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

Table 2-32 ImportDataInfo

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

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

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

Table 2-33 AnnotationFormatConfig

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

Table 2-34 AnnotationFormatParameters

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

Examples

There are three scenarios:

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

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

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

    # Define a dataset.
    ```



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

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

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

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

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

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

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

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

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

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

workflow = wf.Workflow(
```



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

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

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

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

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

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

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

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

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

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

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

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

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



```
object_type must be file or directory.

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

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

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

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

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

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

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

2.3.4.4 Creating a Dataset Release Phase

Description

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

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

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

Parameter Overview

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

Table 2-35 ReleaseDatasetStep

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

Table 2-36 ReleaseDatasetInput

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

Table 2-37 ReleaseDatasetOutput

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

Table 4 **DatasetVersionConfig**

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

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

NOTE

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

Examples

Scenario 1: Releasing a dataset version

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

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

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

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

release_version = wf.steps.ReleaseDatasetStep(
```



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

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

Scenario 2: Releasing a labeling job version

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

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

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

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

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

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

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

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

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



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

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

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

2.3.4.5 Creating a Training Job Phase

Description

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

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

Parameter Overview

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

Table 2-38 JobStep

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

Table 2-39 JobInput

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

Table 2-40 JobOutput

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

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

Table 2-41 OBSOutputConfig

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

Table 2-42 BaseAlgorithm

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

Table 2-43 Algorithm

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

Table 2-44 AIGalleryAlgorithm

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

Table 2-45 AlgorithmParameters

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

Table 2-46 JobEngine

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

Table 2-47 JobSpec

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

Table 2-48 JobResource

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

Table 2-49 SchedulePolicy

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

Table 2-50 Volume

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

Table 2-51 NFS

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

Table 2-52 PFS

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

Obtaining Resource Flavors

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

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

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

Examples

There are seven scenarios:

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

Using an Algorithm Subscribed from AI Gallery

```
from modelarts import workflow as wf

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

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

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

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

        )
    )# Training flavors
)

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

Using an Algorithm in Algorithm Management

```
from modelarts import workflow as wf

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

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

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



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

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

        )
    )# Training flavors
)

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

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

```
from modelarts import workflow as wf

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

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

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

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



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

        )
    )# Training flavors
)

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

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

```
from modelarts import workflow as wf

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

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

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

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

        )
    )# Training flavors
)
```



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

NOTE

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

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

Creating a Job Phase Based on the Dataset Release Phase

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

```
from modelarts import workflow as wf

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

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

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

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

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



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


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

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

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

Job Phase With Visualization

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

Table 2-53 MetricsConfig

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

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

- Key-value pair data

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

- Line chart data

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

- Histogram data

```
[
  {
    "key": "metric",
    "title": "metric",
    "type": "histogram",
    "data": {
      "x_axis": [
        {
          "title": "step/epoch",
          "value": [
            1,
            2,
            3
          ]
        }
```



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

- Confusion matrix

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

- One-dimensional table

```
[
  {
    "key": "Application Evaluation Results",
    "title": "Application Evaluation Results",
    "type": "one-dimensional-table",
    "data": {
      "cell_value": [
        [
          10,
          2,
          0.5
        ]
      ],
      "labels": [
        "samples",
        "maxResTine",
        "p99"
      ]
    }
```



```
    }
]
```

Example:

```
from modelarts import workflow as wf

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

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

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

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


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

        )
    )# Training flavors
)

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

NOTE

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

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

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

```
from modelarts import workflow as wf

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

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

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

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

        )
    )# Training flavors
)

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

NOTE

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

2.3.4.6 Creating a Model Registration Phase

Description

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

- Registering models trained from ModelArts training jobs
- Registering models from custom images

Parameter Overview

You can use ModelStep to create a model registration phase. The following is an example of defining a ModelStep.

Table 2-54 ModelStep

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

Table 2-55 ModelInput

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

Table 2-56 ModelOutput

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

Table 2-57 ModelConfig

Parameter	Description	Mandatory	Data Type
model_type	Model type. Supported types: **TensorFlow**, **MXNet**, **Caffe**, **Spark_MLlib**, **Scikit_Learn**, **XGBoost**, **Image**, **PyTorch**, **Template**, and **Custom**. The default value is **TensorFlow**.	Yes	str
model_name	Model name. Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed.	No	str, Placeholder
model_version	Model version in the format of *Digit.Digit.Digit*. The value range of the digits is [1, 99]. If this parameter is left blank, the version number automatically increases. **CAUTION** No part of the version number can start with 0. For example, **01.01.01** is not allowed.	No	str, Placeholder
runtime	Model runtime environment. The options of **runtime** are the same as those of **model_type**.	No	str, Placeholder
description	Model description that consists of 1 to 100 characters. The following special characters cannot be contained: &!'"<>=	No	str
execution_code	OBS path for storing the execution code. By default, this parameter is left blank. The name of the execution code file is fixed to **customize_service.py**. The inference code file must be stored in the **model** directory. This parameter is left blank. The system can automatically identify the inference code in the **model** directory.	No	str
dependencies	Package required for the inference code and model. By default, this parameter is left blank. It is read from the configuration file.	No	str
model_metrics	Model precision, which is read from the configuration file.	No	str

Parameter	Description	Mandatory	Data Type
apis	All **apis** input and output parameters of a model (optional), which are parsed from the configuration file.	No	str
initial_config	Model configuration information.	No	dict
template	Template configuration items. This parameter is mandatory when **model_type** is set to **Template**.	No	Template
dynamic_load_mode	Dynamic loading mode. Currently, only **Single** is supported.	No	str, Placeholder
prebuild	Whether the model is prebuilt. The default value is **False**.	No	bool, Placeholder
install_type	Model installation type. The value can be **real_time**, **edge**, **batch**. If this parameter is left blank, all types are supported by default.	No	list[str]

Table 2-58 Template

Parameter	Description	Mandatory	Data Type
template_id	ID of the used template. The template has a built-in input and output mode.	Yes	str, Placeholder
infer_format	Input and output mode. When this parameter is used, the input and output mode built in the template does not take effect.	No	str, Placeholder
template_inputs	Template input configuration, specifying the source path for configuring a model	Yes	list of TemplateInputs object

Table 2-59 TemplateInputs

Parameter	Description	Mandatory	Data Type
input_id	Input item ID, which is obtained from the template details.	Yes	str, Placeholder

Parameter	Description	Mandatory	Data Type
input	Template input path, which can be an OBS file path or OBS directory path. When you use a template with multiple input items to create a model, if the target paths **input_properties** specified in the template are the same, the OBS directory or OBS file name entered here must be unique to prevent files from being overwritten.	Yes	str, Placeholder, Storage

Examples

There are six scenarios:

- Registering models output by JobStep
- Registering a model using OBS data
- Registering a model using a template
- Registering a model using a custom image
- Registering a model using a custom image and OBS
- Registering a model using a subscribed model and OBS

Registering a Model from a Training Job (Model Source: JobStep Output)

```
import modelarts.workflow as wf

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

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

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



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

        )
    )# Training flavors
)

# Define a model registration phase using ModelStep. The output of JobStep is used as the input of ModelStep.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # The output of JobStep is used as the input of ModelStep.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep outputs
    depend_steps=job_step # Preceding job phase
)
# job_step is an instance object of wf.steps.JobStep and train_url is the value of the name field of wf.steps.JobOutput.

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

Registering a Model from a Training Job (Model Source: A Trained Model Stored in OBS)

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from OBS.

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

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=obs),  # ModelStep inputs are configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep outputs
)

workflow = wf.Workflow(
    name="model-step-demo",
```



```
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

Registering a Model Using a Template

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. Register a model using a preset template.

# Define a preset template object. Fields in the template object can be represented by placeholders.
template = wf.steps.Template(
    template_id="fake_template_id",
    infer_format="fake_infer_format",
    template_inputs=[
        wf.steps.TemplateInputs(
            input_id="fake_input_id",
            input="fake_input_file"
        )
    ]
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    outputs=wf.steps.ModelOutput(
          name='model_output',
          model_config=wf.steps.ModelConfig(
              model_name=model_name,
              model_type="Template",
              template=template
          )
    ) # ModelStep outputs
)

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

Registering a Model from a Custom Image

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from the URL of a custom image.

# Define the image data.
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name="input",data=swr), # ModelStep inputs are configured when the workflow is running. You can also use wf.data.SWRImage(swr_path="fake_path") for the data field.

outputs=wf.steps.ModelOutput(name='model_output',model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow"))# ModelStep outputs
)

workflow = wf.Workflow(
```

```
    name="model-step-demo",
    desc="this is a demo workflow",
    steps=[model_registration]
)
```

Registering a Model Using a Custom Image and OBS

```
import modelarts.workflow as wf
# Define a model registration phase using ModelStep. The input is from the URL of a custom image.

# Define the image data.
swr = wf.data.SWRImagePlaceholder(name="placeholder_name")

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

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 
64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and 
must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=[
        wf.steps.ModelInput(name="input",data=swr), # ModelStep inputs are configured when the workflow 
is running. You can also use wf.data.SWRImage(swr_path="fake_path") for the data field.
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep inputs are configured when the 
workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    ],
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    ) # ModelStep outputs
)

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

Registering a Model Using a Subscribed Model and OBS

This mode is similar to the custom image + OBS mode, except that you obtain a custom image from a subscribed model.

Example:

```
import modelarts.workflow as wf

# Define the subscribed model object.
base_model = wf.data.GalleryModel(subscription_id="fake_subscription_id", version_num="fake_version") # 
Model subscribed to from AI Gallery, generally published by a developer

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

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_registration = wf.steps.ModelStep(
```



```
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=[
        wf.steps.ModelInput(name="input",data=base_model) # Use a subscribed model as the ModelStep input.
        wf.steps.ModelInput(name="input",data=model_obs) # ModelStep inputs are configured when the workflow is running. You can also use wf.data.OBSPath(obs_path="fake_obs_path") for the data field.
    ],
    outputs=wf.steps.ModelOutput(
        name='model_output',
        model_config=wf.steps.ModelConfig(
            model_name=model_name,
            model_type="Custom",
            dynamic_load_mode="Single"
        )
    ) # ModelStep outputs
)

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

In the preceding example, the system automatically obtains the custom image from the subscribed model and registers and generates a model based on the entered OBS model path. **model_obs** can be replaced with the dynamic output of JobStep.

NOTE

The value of **model_type** can be **TensorFlow**, **MXNet**, **Caffe**, **Spark_MLlib**, **Scikit_Learn**, **XGBoost**, **Image**, **PyTorch**, **Template**, or **Custom**.

If **model_type** is not set for **wf.steps.ModelConfig**, **TensorFlow** is used by default.

- If the model type of your workflow does not need to be changed, refer to the preceding examples.
- If the model type of your workflow needs to be changed in multiple executions, write the parameter using placeholders.

  model_type = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.ENUM, default="TensorFlow", enum_list=["TensorFlow", "MXNet", "Caffe", "Spark_MLlib", "Scikit_Learn", "XGBoost", "Image", "PyTorch", "Template", "Custom"], description="Model type")

2.3.4.7 Creating a Service Deployment Phase

Description

This phase integrates capabilities of ModelArts service management to enable service deployment and update in a workflow. The application scenarios are as follows:

- Deploying a model as a web service
- Updating an existing service (gray update supported)

Parameter Overview

You can use ServiceStep to create a service deployment phase. The following is an example of defining a ServiceStep.

Table 2-60 ServiceStep

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

Table 2-61 ServiceInput

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

Parameter	Description	Mandatory	Data Type
data	Input data object of the service deployment phase.	Yes	Model list or service object. Currently, only ServiceInputPlaceholder, ServiceData, and ServiceUpdatePlaceholder are supported.

Table 2-62 ServiceOutput

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

Table 4 ServiceConfig

Parameter	Description	Mandatory	Data Type
infer_type	Inference mode. The value can be **real-time**, **batch**, or **edge**. The default value is **real-time**. • **real-time**: real-time service. The model is deployed as a web service. • **batch**: batch service. A batch service can perform inference on batch data and automatically stops after data processing is completed. • **edge**: edge service. A model is deployed as a web service on an edge node through IEF. You must create an edge node on IEF beforehand.	Yes	str
service_name	Service name. Enter 1 to 64 characters. Only letters, digits, hyphens (-), and underscores (_) are allowed. **NOTE** If you do not specify this parameter, the default service name is generated automatically.	No	str, Placeholder
description	Service description, which contains a maximum of 100 characters. By default, this parameter is left blank.	No	str
vpc_id	ID of the VPC to which a real-time service instance is deployed. By default, this parameter is left blank. In this case, ModelArts allocates a dedicated VPC to each user, and users are isolated from each other. To access other service components in the VPC of the service instance, set this parameter to the ID of the corresponding VPC. Once a VPC is configured, it cannot be modified. If both **vpc_id** and **cluster_id** are configured, only the dedicated resource pool takes effect.	No	str

Parameter	Description	Mandatory	Data Type
subnet_network_id	ID of a subnet. By default, this parameter is left blank. This parameter is mandatory when **vpc_id** is configured. Enter the network ID displayed in the subnet details on the VPC management console. A subnet provides dedicated network resources that are isolated from other networks.	No	str
security_group_id	Security group. By default, this parameter is left blank. This parameter is mandatory when **vpc_id** is configured. A security group is a virtual firewall that provides secure network access control policies for service instances. A security group must contain at least one inbound rule to permit the requests whose protocol is TCP, source address is **0.0.0.0/0**, and port number is **8080**.	No	str
cluster_id	ID of a dedicated resource pool. By default, this parameter is left blank, indicating that no dedicated resource pool is used. When using a dedicated resource pool to deploy services, ensure that the cluster is running properly. After this parameter is configured, the network configuration of the cluster is used, and the **vpc_id** parameter does not take effect. If both this parameter and **cluster_id** in **real-time config** are configured, **cluster_id** in **real-time config** is preferentially used.	No	str
additional_properties	Additional configurations.	No	dict
apps	Whether to enable application authentication for service deployment. Multiple application names can be entered.	No	str, Placeholder, list
envs	Environment variables.	No	dict

Example:


```
example = ServiceConfig()
# This object is used in the output of the service deployment phase.
```

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

Examples

There are three scenarios:

- Deploying a real-time service
- Modifying a real-time service
- Getting the inference address from the service deployment phase

Deploying a Real-Time Service

```
import modelarts.workflow as wf
# Use ServiceStep to define a service deployment phase and specify a model for service deployment.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

service_step = wf.steps.ServiceStep(
    name="service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Deploying a Service",  # Title
    inputs=wf.steps.ServiceInput(name="si_service_ph",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder1",
                                                    # Restrictions on the model name: Only the model name specified here can be used in the running state; use the same model name as **model_name** of the model registration phase.
                                                    model_name=model_name)),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)

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

Modifying a Real-Time Service

Scenario: When you use a new model version to update an existing service, ensure that the name of the new model version is the same as that of the deployed service.

```
import modelarts.workflow as wf
# Use ServiceStep to define a service deployment phase and specify a model for service update.

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# Define a service object.
service = wf.data.ServiceUpdatePlaceholder(name="placeholder_name")

service_step  = wf.steps.ServiceStep(
    name="service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Service Update", # Title
    inputs=[wf.steps.ServiceInput(name="si2",
data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
                                                    # Restrictions on the model name: Only the model
```



```
name specified here can be used in the running state.
                                                    model_name=model_name)),
        wf.steps.ServiceInput(name="si_service_data", data=service) # ServiceStep inputs are configured when the workflow is running. You can also use wf.data.ServiceData(service_id="fake_service") for the data field.
    ], # ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)

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

Getting the Inference Address from the Service Deployment Phase

The service deployment phase supports the output of the inference address. You can use the **get_output_variable("access_address")** method to obtain the output and use it in subsequent phases.

- For services deployed in the public resource pool, you can use **access_address** to obtain the inference address registered on the public network from the output.
- For services deployed in a dedicated resource pool, you can get the internal inference address from the output using **cluster_inner_access_address**, in addition to the public inference address. The internal address can only be accessed by other inference services.

```
import modelarts.workflow as wf

# Define model name parameters.
sub_model_name = wf.Placeholder(name="si_placeholder1", placeholder_type=wf.PlaceholderType.STR)

sub_service_step = wf.steps.ServiceStep(
    name="sub_service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Subservice",  # Title
    inputs=wf.steps.ServiceInput(
        name="si_service_ph",
        data=wf.data.ServiceInputPlaceholder(name="si_placeholder1", model_name=sub_model_name)
    ),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output") # ServiceStep outputs
)


main_model_name = wf.Placeholder(name="si_placeholder2", placeholder_type=wf.PlaceholderType.STR)

# Obtain the inference address output by the subservice and transfer the address to the main service through envs.
main_service_config = wf.steps.ServiceConfig(
    infer_type="real-time",
    envs={"infer_address": sub_service_step.outputs["service_output"].get_output_variable("access_address")} # Obtain the inference address output by the subservice and transfer the address to the main service through envs.
)

main_service_step = wf.steps.ServiceStep(
    name="main_service_step", # Name of the service deployment phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Main service",  # Title
    inputs=wf.steps.ServiceInput(
        name="si_service_ph",
```



```
        data=wf.data.ServiceInputPlaceholder(name="si_placeholder2",
model_name=main_model_name)
    ),# ServiceStep inputs
    outputs=wf.steps.ServiceOutput(name="service_output", service_config=main_service_config), #
ServiceStep outputs
    depend_steps=sub_service_step
)

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

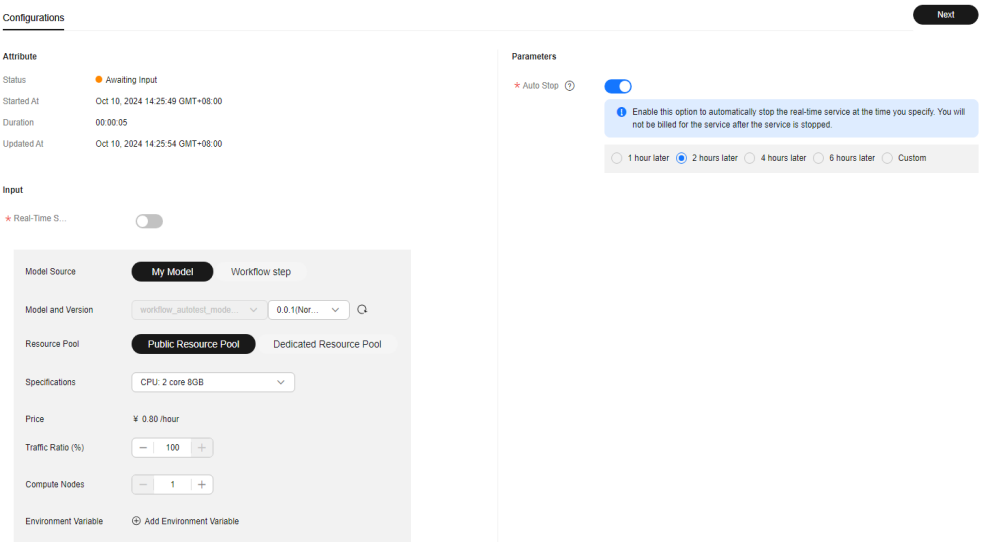
Configuring Information for Deploying a Synchronous Service

After the service deployment phase is started in the development state (usually a notebook instance), configure the information based on the following format in the logs.

```
Please enter the ServiceInputPlaceholder "service_model" in the following format:
"[{'model_name': '******', 'model_version': '*.*.*', 'specification':'[redacted]', 'weight': 50}, {'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 30,
'envs':{'k1': 'v1', 'k2': 'v2'}},{'model_name': '******', 'model_version': '*.*.*', 'specification':'******', 'weight': 20, 'envs':{'******': '******', '******': '******'}},]"
Note that:(1) The "[" at the beginning and "]" at the end are required.
          (2) The sum of the weights must be equal to 100.
          (3) All model must have the same model name. Two model versions cannot be the same.
```

1. On the ModelArts console, choose **Development Workspace** > **Workflow** from the navigation pane.
2. Configure the information after the service deployment phase is started. After the configuration, click **Next**.

   If you want to start the deployment phase automatically without manual configuration, configure the **ServiceInputConfig** or **ServiceConfig** parameter in the code in advance, for example, **service_config=wf.steps.ServiceConfig(cluster_id="XX")**. If all required parameters are specified, the service deployment phase automatically starts.



Configuring Information for Deploying an Asynchronous Service

1. On the ModelArts console, choose **Workflow** from the navigation pane.
2. Configure the information after the service deployment phase is started. Select an asynchronous inference model and a version, and configure service startup parameters. After the configuration, click **Next**.

NOTE

After you select the required model and version, the system automatically matches the service startup parameters.

2.3.5 Creating a Multi-Branch Workflow

2.3.5.1 Multi-Branch Workflow

You can implement multi-branch in two ways. However, the condition phase is limited to two branches. **Configuring Phase Parameters to Control Branch Execution** allows you to replace the ConditionStep capability without adding new phases, offering more flexibility.

Creating a Condition Phase to Control Branch Execution is used for conditional branching in the execution of phases based on condition value comparison or metrics output by the preceding phase.

Configuring Phase Parameters to Control Branch Execution is used for complex scenarios that involve multiple branches. When each execution starts, the workflow decides which branches to run and which ones to skip based on the relevant configuration information. This way, only some branches are executed. This function has a similar use case as ConditionStep, but it is more powerful.

2.3.5.2 Creating a Condition Phase to Control Branch Execution

Description

This phase is used for conditional branching in the execution of phases based on condition value comparison or metrics output by the preceding phase. The application scenarios are as follows:

You need to determine the subsequent process based on different input values. If you need to determine whether to retrain or register a model based on the model precision output by the training phase, you can use this phase to control the process.

Parameter Overview

You can use ConditionStep to create a condition phase. The following is an example of defining a ConditionStep.

Table 2-63 ConditionStep

Parameter	Description	Mandatory	Data Type
name	Name of a condition phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.	Yes	str
conditions	List of conditions. The AND operation is used for multiple conditions.	Yes	Condition or condition list
if_then_steps	Steps to be executed if the calculation result of the condition expression is **True**.	No	str or str list
else_then_steps	Steps to be executed if the calculation result of the condition expression is **False**.	No	str or str list
title	Title for frontend-phase display.	No	str
description	Description of a condition phase.	No	str
depend_steps	Dependent phases.	No	Step or step list

Table 2-64 Condition

Parameter	Description	Mandatory	Data Type
condition_type	Condition type. The "==", ">", ">=", "in", "<", "<=", "!=", and "or" operators are supported.	Yes	ConditionTypeEnum
left	Left value of a condition expression.	Yes	int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo

Parameter	Description	Mandatory	Data Type
right	Right value of a condition expression	Yes	int, float, str, bool, Placeholder, Sequence, Condition, MetricInfo

Table 2-65 MetricInfo

Parameter	Description	Mandatory	Data Type
input_data	Metric input. Currently, only the output of JobStep is supported.	Yes	JobStep output
json_key	Key value corresponding to the metric information to be obtained	Yes	str

Description of the structure:

- **Condition object, which consists of the condition type, left value, and right value**
  - The condition type is obtained from ConditionTypeEnum. The "==", ">", ">=", "in", "<", "<=", "!=", and "or" operators are supported. The following table describes the mapping.

Enumeration	Operator
ConditionTypeEnum.EQ	==
ConditionTypeEnum.GT	>
ConditionTypeEnum.GTE	>=
ConditionTypeEnum.IN	in
ConditionTypeEnum.LT	<
ConditionTypeEnum.LTE	<=
ConditionTypeEnum.NOT	!=
ConditionTypeEnum.OR	or

  - The left and right values support the following types: integer, float, string, bool, placeholder, sequence, condition, and MetricInfo.
  - A condition phase supports a list of condition objects. The && operation is performed between multiple conditions.

- **if_then_steps and else_then_steps**
  - if_then_steps indicates a list of phases that are ready for execution if conditions evaluate to **true**. In this case, steps in else_then_steps are skipped.
  - else_then_steps indicates a list of phases that are ready for execution if conditions evaluate to **false**. In this case, steps in if_then_steps are skipped.

Examples

Refer to simple or advanced examples as needed.

Simple Examples

- Implemented using parameter configurations

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL,
default=True)

# Condition object
condition = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value,
right=True) # Condition object, including the type, left value, and right value.

# Condition phase
condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    conditions=condition, # Condition objects. The relationship between the conditions is &&.
    if_then_steps="job_step_1", # If conditions evaluate to true, job_step_1 is ready for execution, and
job_step_2 is skipped.
    else_then_steps="job_step_2" # If conditions evaluate to false, job_step_2 is ready for execution,
and job_step_1 is skipped.
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
model_step_1 = wf.steps.ModelStep(
    name="model_step_1",
    depend_steps=job_step_1
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
model_step_2 = wf.steps.ModelStep(
    name="model_step_2",
    depend_steps=job_step_2
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
```



```
    steps=[condition_step, job_step_1, job_step_2, model_step_1, model_step_2]
)
```

NOTE

Scenario description: **job_step_1** and **job_step_2** indicate two training phases that depend on **condition_step**. condition_step parameters determine the subsequent phase execution.

Execution analysis:

- If the default value of **left_value** is **True**, the calculation result of the condition logical expression is **True**. Then, **job_step_1** is executed, **job_step_2** is skipped, and all phases contained in the branches that use **job_step_2** as the unique root node are skipped. That is, **model_step_2** is skipped. Therefore, **condition_step**, **job_step_1**, and **model_step_1** are executed.
- If **left_value** is set to **False**, the calculation result of the condition logical expression is **False**. Then, **job_step_2** is executed, **job_step_1** is skipped, and all phases contained in the branches that use **job_step_1** as the unique root node are skipped. That is, **model_step_1** is skipped, and **condition_step**, **job_step_2**, and **model_step_2** are executed.

- Implemented by obtaining the metric information output by JobStep

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True,
create_dir=True, description="description_info")  # The name field is mandatory, and the title and
description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter
and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version
number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If
the value of an algorithm hyperparameter does not need to be changed, you do not need to
configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("dir
ectory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json",
create_dir=False))) # Metric output path. Metric information is automatically output by the job script
based on the specified data format. (In the example, the metric information needs to be output to the
metrics.json file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON,
description="Training flavor")
        )
     )  # Training flavors
)
```



```
# Define a condition object.
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    conditions=condition_lt, # Condition objects. The relationship between the conditions is &&.
    if_then_steps="training_job_retrain", # If conditions evaluate to true, training_job_retrain is ready for execution, and model_registration is skipped.
    else_then_steps="model_registration", # If conditions evaluate to false, model_registration is ready for execution, and training_job_retrain is skipped.
    depend_steps=job_step
)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_retrain = wf.steps.JobStep(
    name="training_job_retrain",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification retraining",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_retrain"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path_retrain/metrics.json", create_dir=False))) # Metric output path. Metric information is automatically output by the job script based on the specified data format. (In the example, the metric information needs to be output to the metrics.json file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor_retrain",
placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")
        )
     ),  # Training flavors
    depend_steps=condition_step
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # job_step output is used as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep outputs
    depend_steps=condition_step,
```



```
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[job_step, condition_step, job_step_retrain, model_step],
    storages=storage
)
```

In this example, ConditionStep obtains the accuracy output by **job_step** and compares it with the preset value to determine whether to retrain or register the model. When the accuracy output by **job_step** is less than the threshold 0.5, the calculation result of **condition_lt** is **True**. In this case, **job_step_retrain** runs and **model_step** skips. Otherwise, **job_step_retrain** skips and **model_step** runs.

NOTE

For details about the format requirements of the metric file generated by **job_step**, see **Creating a Training Job Phase**. In the condition phase, only the metric data whose type is float can be used as the input.

The following is an example of the **metrics.json** file:

```
[
  {
    "key": "loss",
    "title": "loss",
    "type": "float",
    "data": {
      "value": 1.2
    }
  },
  {
    "key": "accuracy",
    "title": "accuracy",
    "type": "float",
    "data": {
      "value": 0.8
    }
  }
]
```

Advanced Examples

```
import modelarts.workflow as wf

left_value = wf.Placeholder(name="left_value", placeholder_type=wf.PlaceholderType.BOOL, default=True)
condition1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=left_value, right=True)

internal_condition_1 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.GT, left=10, right=9)
internal_condition_2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.LT, left=10, right=9)

# The result of condition2 is internal_condition_1 || internal_condition_2.
condition2 = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.OR, left=internal_condition_1,
right=internal_condition_2)

condition_step = wf.steps.ConditionStep(
    name="condition_step_test", # Name of the condition phase. The name contains a maximum of 64
characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and
must be unique in a workflow.
    conditions=[condition1, condition2], # Condition objects. The relationship between the conditions is &&.
    if_then_steps=["job_step_1"], # If conditions evaluate to true, job_step_1 is ready for execution, and
job_step_2 is skipped.
    else_then_steps=["job_step_2"] # If conditions evaluate to false, job_step_2 is ready for execution, and
job_step_1 is skipped.
)
```



```
# This phase is used only as an example. You need to supplement other fields as required.
job_step_1 = wf.steps.JobStep(
    name="job_step_1",
    depend_steps=condition_step
)

# This phase is used only as an example. You need to supplement other fields as required.
job_step_2 = wf.steps.JobStep(
    name="job_step_2",
    depend_steps=condition_step
)

workflow = wf.Workflow(
    name="condition-demo",
    desc="this is a demo workflow",
    steps=[condition_step, job_step_1, job_step_2],
)
```

ConditionStep supports nested condition phases. You can flexibly design tit based on different scenarios.

NOTE

The condition phase can only support two branches, which is very limiting. You can use the new branch function to replace the ConditionStep capability without creating new phases. For details, see **Configuring Phase Parameters to Control Branch Execution**.

2.3.5.3 Configuring Phase Parameters to Control Branch Execution

Function

You can use parameters or metrics from training output to decide whether to run a phase. This way, you can control the process.

Application Scenarios

This function is used for complex scenarios that involve multiple branches. When each execution starts, the workflow decides which branches to run and which ones to skip based on the relevant configuration information. This way, only some branches are executed. This function has a similar use case as ConditionStep, but it is more powerful. This function applies to the dataset creation phase, labeling phase, dataset import phase, dataset release phase, job phase, model registration phase, and service deployment phase.

Controlling the Execution of a Single Phase

- Implemented using parameter configurations

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

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

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



```
# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

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

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

        )
    ), # Training flavors
    policy=wf.steps.StepPolicy(
        skip_conditions=[condition_equal] # Determines whether to skip job_step based on the calculation result of skip_conditions.
    )
)

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

In this example, **job_step** has a skip policy that is controlled by a bool parameter. If the placeholder parameter named **is_skip** is set to **True**, then **job_step** is skipped when **condition_equal** evaluates to **True**. Otherwise, **job_step** is run. For more details about the condition object, see **Creating a Condition Phase to Control Branch Execution**.

- Implemented by obtaining the metric information output by JobStep

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True, create_dir=True, description="description_info")  # The name field is mandatory, and the title and description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step = wf.steps.JobStep(
    name="training_job", # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
```



```
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[

wf.steps.JobOutput(name="train_url",obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path"))),
        wf.steps.JobOutput(name="metrics",
metrics_config=wf.data.MetricsConfig(metric_files=storage.join("directory_path/metrics.json", create_dir=False))) # Metric output path. Metric information is automatically output by the job script based on the specified data format. (In the example, the metric information needs to be output to the metrics.json file in the training output directory.)
    ],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")
        )
     )  # Training flavors
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

# Define a condition object.
condition_lt = wf.steps.Condition(
    condition_type=wf.steps.ConditionTypeEnum.LT,
    left=wf.steps.MetricInfo(job_step.outputs["metrics"].as_input(), "accuracy"),
    right=0.5
)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=job_step.outputs["train_url"].as_input()),  # job_step output is used as the input.
    outputs=wf.steps.ModelOutput(name='model_output',
model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep outputs
    depend_steps=job_step # Preceding job phase
    policy=wf.steps.StepPolicy(skip_conditions=condition_lt) # Determines whether to skip model_step based on the calculation result of skip_conditions.
)

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

In this example, **model_step** has a skip policy. The model registration depends on whether the accuracy output by **job_step** meets the preset value. When the accuracy output by **job_step** is less than the threshold 0.5, the calculation result of **condition_lt** is **True**. In this case, **model_step** skips. Otherwise, **model_step** runs.

NOTE

For details about the format requirements of the metric file generated by **job_step**, see **Creating a Training Job Phase**. In the condition phase, only the metric data whose type is float can be used as the input.

The following is an example of the **metrics.json** file:

```
[
    {
        "key": "loss", // Metric data name. The value contains a maximum of 64 characters and cannot contain special characters.
        "title": "loss", // Metric data title. The value contains a maximum of 64 characters.
        "type": "float", // Metric data type. Floating point, line chart, histogram, table, and one-dimensional table data types are supported.
        "data": {
            "value": 1.2 // Metric data value. For details about the usage examples of different types, see Creating a Training Job Phase.
        }
    },
    {
        "key": "accuracy",
        "title": "accuracy",
        "type": "float",
        "data": {
            "value": 0.8
        }
    }
]
```

Controlling Partial Execution of Multiple Branches

```
from modelarts import workflow as wf

# Create an OutputStorage object to centrally manage training output directories.
storage = wf.data.Storage(name="storage_name", title="title_info", with_execution_id=True, create_dir=True, description="description_info")  # The name field is mandatory, and the title and description fields are optional.

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

condition_equal_a = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=wf.Placeholder(name="job_step_a_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_a = wf.steps.JobStep(
    name="training_job_a",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_a")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    ),  # Training flavors
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_a)
)

condition_equal_b = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ, left=wf.Placeholder(name="job_step_b_is_skip", placeholder_type=wf.PlaceholderType.BOOL), right=True)
```



```
# Use JobStep to define a training phase, and use OBS to store the output.
job_step_b = wf.steps.JobStep(
    name="training_job_b",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

    ), # Algorithm used for training. An algorithm subscribed to in AI Gallery is used in this example. If the value of an algorithm hyperparameter does not need to be changed, you do not need to configure the hyperparameter in parameters. Hyperparameter values will be automatically filled.
    inputs=wf.steps.JobInput(name="data_url", data=obs_data),
    outputs=[wf.steps.JobOutput(name="train_url", obs_config=wf.data.OBSOutputConfig(obs_path=storage.join("directory_path_b")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(name="train_flavor", placeholder_type=wf.PlaceholderType.JSON, description="Training flavor")

        )
    ),  # Training flavors
    policy=wf.steps.StepPolicy(skip_conditions=condition_equal_b)
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=wf.data.DataConsumptionSelector(data_list=[job_step_a.outputs["train_url"].as_input(), job_step_b.outputs["train_url"].as_input()])),  # Select the output of job_step_a or job_step_b as the input.
    outputs=wf.steps.ModelOutput(name='model_output', model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")),  # ModelStep outputs
    depend_steps=[job_step_a, job_step_b],  # Preceding job phase
)

workflow = wf.Workflow(
    name="new-condition-demo",
    desc="this is a demo workflow",
    steps=[job_step_a, job_step_b, model_step],
    storages=storage
)
```

In this example, both **job_step_a** and **job_step_b** have a skip policy that is controlled by parameters. When the parameter values are different, the execution of **model_step** can be divided into the following cases (**model_step** has no skip policy configured, so it follows the default rule).

job_step_a_is_skip	job_step_b_is_skip	Whether to Execute model_step
True	True	No
	False	Yes
False	True	Yes

job_step_a_is_skip	job_step_b_is_skip	Whether to Execute model_step
	False	Yes

CAUTION

Default rule: A phase is automatically skipped if all the phases it depends on are skipped. Otherwise, the phase is run. This logic can apply to any phase.

Based on the previous example, if you want to override the default rule and make **model_step** run when **job_step_a** and **job_step_b** are skipped, you only need to configure a skip policy in **model_step**. The skip policy takes precedence over the default rule.

2.3.5.4 Configuring Multi-Branch Phase Data

Function

This function is only for the scenario where multiple branches are run. When you create a workflow phase, the data input source of the phase is uncertain. The data input source could be the output of any of the phases it depends on. Only after all dependency phases are run, the valid output is automatically selected as the input based on the actual execution situation.

Examples

```
from modelarts import workflow as wf

condition_equal = wf.steps.Condition(condition_type=wf.steps.ConditionTypeEnum.EQ,
left=wf.Placeholder(name="is_true", placeholder_type=wf.PlaceholderType.BOOL), right=True)
condition_step = wf.steps.ConditionStep(
    name="condition_step",
    conditions=[condition_equal],
    if_then_steps=["training_job_1"],
    else_then_steps=["training_job_2"],
)

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

# Define the input OBS object.
obs_data = wf.data.OBSPlaceholder(name="obs_placeholder_name", object_type="directory")

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_1 = wf.steps.JobStep(
    name="training_job_1",  # Name of a training phase. The name contains a maximum of 64 characters,
including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be
unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number
instead.
        parameters=[]
```



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

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

        )
     ),  # Training flavors
    depend_steps=[condition_step]
)

# Use JobStep to define a training phase, and use OBS to store the output.
job_step_2 = wf.steps.JobStep(
    name="training_job_2",  # Name of a training phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Image classification training",  # Title, which defaults to the value of name.
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="subscription_id",  # Subscription ID of the subscribed algorithm
        item_version_id="item_version_id",  # Algorithm version ID. You can also enter the version number instead.
        parameters=[]

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

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

        )
     ),  # Training flavors
    depend_steps=[condition_step]
)

# Define model name parameters.
model_name = wf.Placeholder(name="placeholder_name", placeholder_type=wf.PlaceholderType.STR)

model_step = wf.steps.ModelStep(
    name="model_registration", # Name of the model registration phase. The name contains a maximum of 64 characters, including only letters, digits, underscores (_), and hyphens (-). It must start with a letter and must be unique in a workflow.
    title="Model Registration",  # Title
    inputs=wf.steps.ModelInput(name='model_input', data=wf.data.DataConsumptionSelector(data_list=[job_step_1.outputs["train_url"].as_input(), job_step_2.outputs["train_url"].as_input()])),  # Select the output of job_step_1 or job_step_2 as the input.
    outputs=wf.steps.ModelOutput(name='model_output', model_config=wf.steps.ModelConfig(model_name=model_name, model_type="TensorFlow")), # ModelStep outputs
    depend_steps=[job_step_1, job_step_2] # Preceding job phase
)# job_step is an instance object of wf.steps.JobStep and train_url is the value of the name field of wf.steps.JobOutput.
```



```
workflow = wf.Workflow(name="data-select-demo",
                       desc="this is a test workflow",
                       steps=[condition_step, job_step_1, job_step_2, model_step],
                       storages=storage
                       )
```

NOTE

The workflow in this example has two parallel branches, but only one branch runs at a time, depending on the configuration of **condition_step**. The input source of **model_step** is either **job_step_1** or **job_step_2**'s output. If **job_step_1** runs and **job_step_2** is skipped, **model_step** uses **job_step_1**'s output as input, and vice versa.

2.3.6 Creating a Workflow

To create a workflow, define each phase by referring to **Creating Workflow Phases**. Follow these steps:

1. Sort out scenarios, understand preset steps' functions, and determine the DAG structure.
2. Debug single-phase functions like training or inference on ModelArts.
3. Choose the code template that matches the phase function and fill in the details.
4. Arrange phases according to the DAG structure to create a workflow.

Importing the Workflow Data Package

When creating a workflow, required objects are imported through workflow packages. The details are as follows:

```
from modelarts import workflow as wf
```

Import the data package:

```
wf.data.DatasetTypeEnum
wf.data.Dataset
wf.data.DatasetVersionConfig
wf.data.DatasetPlaceholder
wf.data.ServiceInputPlaceholder
wf.data.ServiceData
wf.data.ServiceUpdatePlaceholder
wf.data.DataTypeEnum
wf.data.ModelData
wf.data.GalleryModel
wf.data.OBSPath
wf.data.OBSOutputConfig
wf.data.OBSPlaceholder
wf.data.SWRImage
wf.data.SWRImagePlaceholder
wf.data.Storage
wf.data.InputStorage
wf.data.OutputStorage
wf.data.LabelTask
wf.data.LabelTaskPlaceholder
wf.data.LabelTaskConfig
wf.data.LabelTaskTypeEnum
wf.data.MetricsConfig
wf.data.TripartiteServiceConfig
wf.data.DataConsumptionSelector
```

Import the policy package:


```
wf.policy.Policy
wf.policy.Scene
```

Import the steps package:

```
wf.steps.MetricInfo
wf.steps.Condition
wf.steps.ConditionTypeEnum
wf.steps.ConditionStep
wf.steps.LabelingStep
wf.steps.LabelingInput
wf.steps.LabelingOutput
wf.steps.LabelTaskProperties
wf.steps.ImportDataInfo
wf.steps.DataOriginTypeEnum
wf.steps.DatasetImportStep
wf.steps.DatasetImportInput
wf.steps.DatasetImportOutput
wf.steps.AnnotationFormatConfig
wf.steps.AnnotationFormatParameters
wf.steps.AnnotationFormatEnum
wf.steps.Label
wf.steps.ImportTypeEnum
wf.steps.LabelFormat
wf.steps.LabelTypeEnum
wf.steps.ReleaseDatasetStep
wf.steps.ReleaseDatasetInput
wf.steps.ReleaseDatasetOutput
wf.steps.CreateDatasetStep
wf.steps.CreateDatasetInput
wf.steps.CreateDatasetOutput
wf.steps.DatasetProperties
wf.steps.SchemaField
wf.steps.ImportConfig
wf.steps.JobStep
wf.steps.JobMetadata
wf.steps.JobSpec
wf.steps.JobResource
wf.steps.JobTypeEnum
wf.steps.JobEngine
wf.steps.JobInput
wf.steps.JobOutput
wf.steps.LogExportPath
wf.steps.MrsJobStep
wf.steps.MrsJobInput
wf.steps.MrsJobOutput
wf.steps.MrsJobAlgorithm
wf.steps.ModelStep
wf.steps.ModelInput
wf.steps.ModelOutput
wf.steps.ModelConfig
wf.steps.Template
wf.steps.TemplateInputs
wf.steps.ServiceStep
wf.steps.ServiceInput
wf.steps.ServiceOutput
wf.steps.ServiceConfig
wf.steps.StepPolicy
```

Import the workflow package:

```
wf.workflow
wf.Subgraph
wf.Placeholder
wf.PlaceholderType
wf.AlgorithmParameters
wf.BaseAlgorithm
wf.Algorithm
wf.AIGalleryAlgorithm
wf.resource
```



```
wf.SystemEnv
wf.add_whitelist_users
wf.delete_whitelist_users
```

2.3.7 Publishing a Workflow

2.3.7.1 Publishing a Workflow to ModelArts

You can publish a workflow to ModelArts in two ways: **Publishing to the Running State** and **Publishing and Executing the Workflow**. Publishing to the running state requires configuring input and output parameters on the workflow page. Publishing and executing the workflow allows you to modify the code and run it directly through the SDK, eliminating the need to configure and run it on the console.

Publishing to the Running State

After creating a workflow, you can use the **release()** method to publish the workflow to the running state for configuration and execution (on the workflow page of the management console).

Run the following command:

```
workflow.release()
```

After the preceding command is executed, if the log indicates that the workflow is published, you can go to the ModelArts workflow page to view the workflow. On the workflow details page, click **Configure** to configure parameters.

The **release_and_run()** method is based on the **release()** method and allows you to publish and run workflows in the development state, without the need to configure and execute workflows on the console.

CAUTION

Note the following when using this method:

- For all configuration objects related to placeholders in the workflow, you need to either set default values or use fixed data objects directly.
- The method executes differently depending on the workflow object's name. It creates and runs a new workflow if the name does not exist. It updates and runs the existing workflow if the name already exists, using the new workflow structure for the new execution.
  ```
  workflow.release_and_run()
  ```

Publishing and Executing the Workflow

With this method, you can publish and run workflows on the SDK without using the console. You need to modify the workflow code as follows:

```
from modelarts import workflow as wf

# Define a unified output storage path.
output_storage = wf.data.OutputStorage(name="output_storage", description="Unified configuration of output storage", default="**")
```



```
# Dataset object
dataset = wf.data.DatasetPlaceholder(name="input_data", default=wf.data.Dataset(dataset_name="**",
version_name="**"))

# Create a training job.
job_step = wf.steps.JobStep(
    name="training_job",
    title="Image Classification Training",
    algorithm=wf.AIGalleryAlgorithm(
        subscription_id="**", # Subscription ID of the image classification algorithm. Obtain the subscription
ID on the algorithm management page. This parameter is optional.
        item_version_id="10.0.0", # Version number of the subscribed algorithm. This parameter is optional.
        parameters=[
            wf.AlgorithmParameters(name="task_type", value="image_classification_v2"),
            wf.AlgorithmParameters(name="model_name", value="resnet_v1_50"),
            wf.AlgorithmParameters(name="do_train", value="True"),
            wf.AlgorithmParameters(name="do_eval_along_train", value="True"),
            wf.AlgorithmParameters(name="variable_update", value="horovod"),
            wf.AlgorithmParameters(name="learning_rate_strategy",
value=wf.Placeholder(name="learning_rate_strategy", placeholder_type=wf.PlaceholderType.STR,
default="0.002", description="Learning rate for training. 10:0.001,20:0.0001 indicates that the learning rate
of the first 10 epochs is 0.001 and that of the next 10 epochs is 0.0001. If the epoch is not specified, the
learning rate will be adjusted based on the validation precision. The training will be stopped if the precision
is not significantly improved anymore.")),
            wf.AlgorithmParameters(name="batch_size", value=wf.Placeholder(name="batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="Number of images trained in each step
(on a single card)")),
            wf.AlgorithmParameters(name="eval_batch_size", value=wf.Placeholder(name="eval_batch_size",
placeholder_type=wf.PlaceholderType.INT, default=64, description="Number of images validated in each
step (on a single card)")),
            wf.AlgorithmParameters(name="evaluate_every_n_epochs",
value=wf.Placeholder(name="evaluate_every_n_epochs", placeholder_type=wf.PlaceholderType.FLOAT,
default=1.0, description="Validation is performed every n epochs.")),
            wf.AlgorithmParameters(name="save_model_secs", value=wf.Placeholder(name="save_model_secs",
placeholder_type=wf.PlaceholderType.INT, default=60, description="Model saving frequency (unit: s)")),
            wf.AlgorithmParameters(name="save_summary_steps",
value=wf.Placeholder(name="save_summary_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="Summary saving frequency (unit: step)")),
            wf.AlgorithmParameters(name="log_every_n_steps",
value=wf.Placeholder(name="log_every_n_steps", placeholder_type=wf.PlaceholderType.INT, default=10,
description="Log printing frequency (unit: step)")),
            wf.AlgorithmParameters(name="do_data_cleaning",
value=wf.Placeholder(name="do_data_cleaning", placeholder_type=wf.PlaceholderType.STR, default="True",
description="Whether to clean data. If the data format is abnormal, the training fails. You are advised to
enable this function to ensure training stability. If the data volume is too large, data cleaning may take a
long time. You can clean data offline. (Formats including BMP, JPEG, PNG, and RGB three-channel are
supported.) You are advised to use JPEG.")),
            wf.AlgorithmParameters(name="use_fp16", value=wf.Placeholder(name="use_fp16",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use mixed precision.
Mixed precision accelerates training but causes precision loss. Enable this parameter unless precision is
strictly required.")),
            wf.AlgorithmParameters(name="xla_compile", value=wf.Placeholder(name="xla_compile",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use XLA for accelerated
training. This function is enabled by default.")),
            wf.AlgorithmParameters(name="data_format", value=wf.Placeholder(name="data_format",
placeholder_type=wf.PlaceholderType.ENUM, default="NCHW", enum_list=["NCHW", "NHWC"],
description="Input data format. NHWC indicates channel last, and NCHW indicates channel first. This
parameter defaults to NCHW (faster).")),
            wf.AlgorithmParameters(name="best_model", value=wf.Placeholder(name="best_model",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to save and use the model
with the highest precision instead of the latest model during training. The default value is True, indicating
that the optimal model is saved. Within a certain error range, the latest high precision model is saved as
the optimal model.")),
            wf.AlgorithmParameters(name="jpeg_preprocess", value=wf.Placeholder(name="jpeg_preprocess",
placeholder_type=wf.PlaceholderType.STR, default="True", description="Whether to use the JPEG
preprocessing acceleration operator (only JPEG data is supported) to accelerate data reading and improve
performance. This function is enabled by default. If the data format is not JPEG, enable data cleaning to use
the function."))
```



```
        ]
    ),
    inputs=[wf.steps.JobInput(name="data_url", data=dataset)],
    outputs=[wf.steps.JobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=output_storage.join("/train_output/")))],
    spec=wf.steps.JobSpec(
        resource=wf.steps.JobResource(
            flavor=wf.Placeholder(
                name="training_flavor",
                placeholder_type=wf.PlaceholderType.JSON,
                description="Training flavor",
                default={"flavor_id": "**"}
            )
        )
    )
)

# Create a workflow.
workflow = wf.Workflow(
    name="image-classification-ResNeSt",
    desc="this is an image classification workflow",
    steps=[job_step],
    storages=[output_storage]
)
```

1. Fill in the actual values for all ** in the code above. The configuration mainly involves these three items:
   - Unified storage: default value of **output_storage**. Enter an existing OBS path in the format of */OBS bucket name/Folder path/*.
   - Dataset object: Enter the dataset name and version number.
   - Training flavor: Configure GPU resources since the algorithm in this example can run only on GPUs. You can use free flavor **modelarts.p3.large.public.free**.
2. After the configuration, run this code:
   ```
   workflow.release_and_run()
   ```
3. After the execution, go to the ModelArts console. In the navigation pane, choose **Workflow** to view the workflow status.

2.3.7.2 Publishing a Workflow to AI Gallery

You can publish workflows to AI Gallery and share them with other users. To do so, run this code:

```
workflow.release_to_gallery()
```

Once the release is done, you can visit AI Gallery to see the asset details. The asset permission is set to private by default, but you can change it if you want.

1. Go to AI Gallery.
2. Choose **My Gallery** > **My Assets** > **Workflow**.
3. In the **My Publishes** tab, view the workflow published to AI Gallery.
4. Click the workflow name to view the workflow details.

The **release_to_gallery()** method contains the following input parameters.

Parameter	Description	Mandatory	Type
content_id	Workflow asset ID	No	str

Parameter	Description	Mandatory	Type
version	Version number of the workflow asset. The format is *x.x.x*.	No	str
desc	Description of the workflow asset version	No	str
title	Workflow asset name. If this parameter is not specified, the workflow name is used by default.	No	str
visibility	Visibility of the workflow asset. The value can be **public**, **group** (whitelist), and **private** (visible only to you). The default value is **private**.	No	str
group_users	Whitelist. This parameter is mandatory only when visibility is set to **group**. You can only enter **domain_id**.	No	list[str]

You can use the method in two ways based on the input parameters:

- workflow.release_to_gallery(title="*Asset name*") publishes a new workflow asset with version 1.0.0. If the workflow includes an algorithm that is not from AI Gallery, it also publishes the algorithm to AI Gallery with version 1.0.0.
- workflow.release_to_gallery(content_id="****", title="*Asset name*") publishes a new version based on the specified workflow asset. The version number increases automatically. If the workflow includes an AI Gallery algorithm, it updates the algorithm asset with a higher version number.

Workflow asset whitelist setting:

You can specify the **visibility** and **group_users** fields of the **release_to_gallery** method when you publish an asset for the first time. To change the whitelist for accessing a certain asset, use this command:

```
from modelarts import workflow as wf

# Add specified whitelist users.
wf.add_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])

# Delete specified whitelist users.
wf.delete_whitelist_users(content_id="**", version_num="*.*.*", user_groups=["**", "**"])
```


NOTE

When you modify the whitelist for a workflow asset, the system automatically obtains the information of the algorithm asset that the workflow version depends on and sets the whitelist for the algorithm asset as well.

2.3.8 Advanced Workflow Capabilities

2.3.8.1 Using Big Data Capabilities (MRS) in a Workflow

Function

This phase calls MRS for big data cluster computing. It is used for batch data processing and model training.

Application Scenarios

You can use MRS Spark for big data computing in this phase.

Examples

On the Huawei Cloud MRS console, check available MRS clusters of your account. If no MRS cluster is available, create one with Spark selected.

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

- **Specifying a boot script and cluster**

```
from modelarts import workflow as wf
# Define a MrsJobStep using MrsStep.

algorithm = wf.steps.MrsJobAlgorithm(
    boot_file="obs://spark-sql/wordcount.py",    # OBS path to the boot script
    parameters=[wf.AlgorithmParameters(name="run_args", value="--master,yarn-cluster")]
)
inputs = wf.steps.MrsJobInput(name="mrs_input", data=wf.data.OBSPath(obs_path="/spark-sql/
mrs_input/"))  # OBS path to the input data
outputs = wf.steps.MrsJobOutput(name="mrs_output",
obs_config=wf.data.OBSOutputConfig(obs_path="/spark-sql/mrs_output")) # OBS path to the output
data
step = wf.steps.MrsJobStep(
        name="mrs_test",          # Step name, which can be customized
        mrs_algorithm=algorithm,
        inputs=inputs,
        outputs=outputs,
        cluster_id="cluster_id_xxx"   # MRS cluster ID
)
```

- **Configuring a cluster and boot script**

```
from modelarts import workflow as wf
# Define a phase using MrsJobStep.
run_arg_description = "Program execution parameter, which is used as the program running
environment parameter. The default value is (--master,yarn-cluster)".
app_arg_description = "Program execution parameter, which is used as the input parameter of the
boot script, for example, (--param_a=3,--param_b=4). This parameter is optional and left blank by
default."
mrs_outputs_description = "Data output path, which can be obtained from train_url in the parameter
list."
cluster_id_description = "cluster id of MapReduce Service"
```

```
algorithm = wf.steps.MrsJobAlgorithm(
    boot_file=wf.Placeholder(name="boot_file",
                             description="Program boot script",
                             placeholder_type=wf.PlaceholderType.STR,
                             placeholder_format="obs"),
    parameters=[wf.AlgorithmParameters(name="run_args",
                                       value=wf.Placeholder(name="run_args",
                                                            description=run_arg_description,
                                                            default="--master,yarn-cluster",
                                                            placeholder_type=wf.PlaceholderType.STR),
                                       ),
                wf.AlgorithmParameters(name="app_args",
                                       value=wf.Placeholder(name="app_args",
                                                            description=app_arg_description,
                                                            default="",
                                                            placeholder_type=wf.PlaceholderType.STR)
                                       )
                ]
    )

inputs = wf.steps.MrsJobInput(name="data_url",
                              data=wf.data.OBSPlaceholder(name="data_url",object_type="directory"))

outputs = wf.steps.MrsJobOutput(name="train_url",
obs_config=wf.data.OBSOutputConfig(obs_path=wf.Placeholder(name="train_url",
placeholder_type=wf.PlaceholderType.STR,
placeholder_format="obs",description=mrs_outputs_description)))

mrs_job_step = wf.steps.MrsJobStep(
    name="mrs_job_test",
    mrs_algorithm=algorithm,
    inputs=inputs,
    outputs=outputs,
    cluster_id=wf.Placeholder(name="cluster_id", placeholder_type=wf.PlaceholderType.STR,
description=cluster_id_description, placeholder_format="cluster")
)
```

- **Using an MRS phase on the console**

  After a workflow is published, configure phase parameters such as the data input, data output, boot script, and cluster ID on the workflow configuration page.

2.3.8.2 Specifying Certain Phases to Run in a Workflow

Workflows support predefined scenarios to enable partial execution. You can split the DAG into different branches based on the scenarios during workflow development. Then, you can run each branch independently as a separate workflow. The sample code is as follows:

```
workflow =wf.Workflow(
    name="image_cls",
    desc="this is a demo workflow",
    steps=[label_step, release_data_step, training_step, model_step, service_step],
    policy=wf.policy.Policy(
        scenes=[
            wf.policy.Scene(
                scene_name="Model training",
                scene_steps=[label_step, release_data_step, training_step]
            ),
            wf.policy.Scene(
                scene_name="Service deployment",
                scene_steps=[model_step, service_step]
            ),
        ]
    )
)
```


This example shows a workflow with five phases. The policy defines two preset scenarios: model training and service deployment. When the workflow is published to the running state, partial execution is disabled by default and all phases run. You can specify certain scenarios to run on the global configuration page.

NOTE

You can define the same phase in different running scenarios using partial execution. However, you must ensure that the data dependency between phases is correct. Partial execution can only be configured and used in the running state and cannot be debugged in the development state.

