CodeArts Pipeline

User Guide

Issue	01
Date	2025-09-11

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 CodeArts Pipeline Usage

CodeArts Pipeline is a visual platform that facilitates automated task scheduling. It relies on the automated tasks in other CodeArts services such as **Build**, **Check**, **TestPlan**, and **Deploy**.

You can orchestrate these automated tasks based on your requirements, such as application deployment in the development, test, or production environment. A single configuration triggers executions to save manual operations.

CodeArts Pipeline is a service provided by the **CodeArts** solution. For details about its role in the solution, see **CodeArts Architecture**.

Operation Process

Figure 1-1 Pipeline operation process

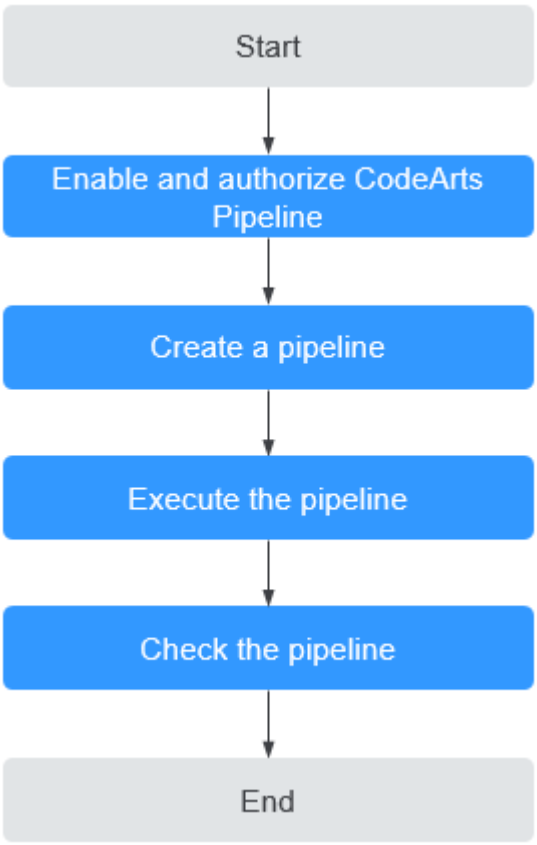


Table 1-1 Description of the operation process

Process	Description
Enable and authorize CodeArts Pipeline	**Enable** and **authorize CodeArts Pipeline**.
Create a pipeline	**Create a pipeline through either the GUI** or **YAML**. Configure **parameters**, **execution plans**, **roles and permissions**, and **event notifications** for the pipeline.
Execute the pipeline	**Execute a pipeline** manually or through schedulers.
Check the pipeline	**Check pipeline information and execution results**.

2 Enabling and Authorizing CodeArts Pipeline

Prerequisites

You have **signed up for a HUAWEI ID and enabled Huawei Cloud services**.

Enabling CodeArts Pipeline

You need to subscribe to a CodeArts package before using CodeArts Pipeline.

Step 1 Access the **CodeArts Pipeline console**.

Step 2 Click **Buy** to purchase a CodeArts package.

Step 3 Purchase a package as needed. For details, see **Purchasing CodeArts**.

----End

Authorizing CodeArts Pipeline

You can configure CodeArts Pipeline permissions at three levels to control user behaviors.

Table 2-1 Pipeline permissions

Level	Module	Description
Tenant-level permissions	Extension, tenant-level policy, tenant-level rule, and pipeline template	Permissions to manage module resources in a tenant. You can configure permissions in IAM. The configurations take effect for all projects of a tenant.

Level	Module	Description
Project-level permissions	Pipeline, policy (project-level), microservice, environment, and change	Permissions to manage module resources of a specific project. You can configure permissions in project settings. The configurations take effect for all resources of a project.
Resource-level permissions	Pipeline	Permissions to perform operations for a specific pipeline. You can configure permissions in a pipeline. The configuration takes effect for a specified pipeline.

- Tenant-level permissions

  IAM allows you to configure permissions for specified users regarding tenant-level rules, tenant-level policies, extensions, and pipeline templates.

  a. Log in to CodeArts using a tenant account or an authorized account.
  b. Click the username in the upper right corner and select **IAM**.
  c. In the navigation pane on the left, choose **User Groups**. On the displayed page, create a user group or select an existing user group, and click **Authorize**.

     Select the CodeArts Pipeline service to check related policies, as shown in the following table.

     Table 2-2 Pipeline policies

Policy Name	Description
CloudPipeline Tenant Rules FullAccess	Full permissions on tenant-level rules within CodeArts Pipeline. • Permissions on rules correspond to **cloudpipeline:rule:update** in IAM. An administrator can use the system-defined policy **CloudPipeline Tenant Rules FullAccess** or custom policies to authorize users. • Common users can check all tenant-level rules. Authorized users can check and manage all tenant-level rules.

Policy Name	Description
CloudPipeline Tenant Rule Templates FullAccess	Full permissions on tenant-level policies within CodeArts Pipeline. • Permissions on pipeline policies correspond to **cloudpipeline:ruletemplate:update** in IAM. An administrator can use the system-defined policy **CloudPipeline Tenant Rule Templates FullAccess** or custom policies to authorize users. • Common users can check all tenant-level policies. Authorized users can check and manage all tenant-level policies.
CloudPipeline Tenant Extensions FullAccess	Full permissions on extensions within CodeArts Pipeline. • Permissions on extensions correspond to **cloudpipeline:extensions:update** in IAM. An administrator can use the system-defined policy **CloudPipeline Tenant Extensions FullAccess** or custom policies to authorize users. • Common users can view all extensions. Authorized users can view and manage all extensions.
CloudPipeline Tenant Pipeline Templates FullAccess	Full permissions on pipeline templates within CodeArts Pipeline. • Permissions on pipeline templates correspond to **cloudpipeline:pipelinetemplate:update** in IAM. An administrator can use the system-defined policy **CloudPipeline Tenant Pipeline Templates FullAccess** or custom policies to authorize users. • Common users can create templates and view all templates. However, they can manage only the templates created by themselves. Authorized users can view and manage all templates.

d. Select the required policies, click **Next**, and set the minimum authorization scope for the user group.

e. Add the specified users to the user group to complete user authorization.

NOTE

In addition to system-defined policies, tenants can also **create custom policies** to grant permissions.

- Project-level permissions

  CodeArts allows you to configure permissions on pipeline resources for each role in a project.

  a. **Log in to the Huawei Cloud console**.

b. Click  in the upper left corner of the page and choose **Developer Services** > **CodeArts Pipeline** from the service list.

c. Click **Access Service** to access the CodeArts Pipeline homepage.

d. On the top navigation bar, click **Homepage** to access the CodeArts homepage.

e. Click a project name to access the project.

f. In the left navigation pane on the left, choose **Settings** > **Permissions**.

Pipeline-related resources are in CodeArts Pipeline. They are change, development environment, pipeline, policy (project-level), microservice, pre-production environment, production environment, test environment, parameter group, and job template.

NOTE

By default, a user with permissions to edit or execute pipelines can also view pipelines.

Pipeline permissions

The following table lists the pipeline permissions for each role in a project in the initial state.

Table 2-3 Project-level permissions

Role	View	Create	Execute	Edit	Delete	Group	Tag	Disable
Project admin	√	√	√	√	√	√	√	√
Project manager	√	√	√	√	√	√	√	√
Developer	√	√	√	×	×	×	×	×
Test manager	√	×	×	×	×	×	×	×
Tester	√	×	×	×	×	×	×	×
Participant	√	×	×	×	×	×	×	×
Viewer	√	×	×	×	×	×	×	×
Product manager	√	×	×	×	×	×	×	×

Role	View	Create	Execute	Edit	Delete	Group	Tag	Disable
System engineer	√	√	√	√	√	√	√	√
Committer	√	√	√	×	×	×	×	×

- To clone a pipeline, you must have the permission to create a pipeline and edit the source pipeline.
- By default, role permissions in a pipeline inherit and are associated with the role permissions in the project until role or user permissions are modified in the pipeline.
- By default, a pipeline creator has all permissions on the pipeline.

Policy permissions

The following table lists the project-level policy permissions for each role in a project in the initial state.

Table 2-4 Project-level policy permissions

Role	View	Create	Edit	Delete
Project admin	√	√	√	√
Project manager	√	√	√	√
Developer	√	√	√	√
Test manager	√	×	×	×
Tester	√	×	×	×
Participant	√	×	×	×
Viewer	√	×	×	×
Product manager	√	×	×	×
System engineer	√	√	√	√
Committer	√	√	√	√

To clone a policy, you must have the permission to create a policy and edit the source policy.

Microservice permissions

The following table lists the microservice permissions for each role in a project in the initial state.

Table 2-5 Project-level microservice permissions

Role	View	Create	Edit	Delete
Project admin	√	√	√	√
Project manager	√	√	√	√
Developer	√	×	×	×
Test manager	√	×	×	×
Tester	√	×	×	×
Participant	√	×	×	×
Viewer	√	×	×	×
Product manager	√	×	×	×
System engineer	√	√	√	√
Committer	√	×	×	×

Change permissions

The following table lists the change permissions for each role in a project in the initial state.

Table 2-6 Project-level change permissions

Role	View	Create	Edit	Execute
Project admin	√	√	√	√
Project manager	√	√	√	√
Developer	√	√	√	√
Test manager	√	×	×	×
Tester	√	×	×	×
Participant	√	×	×	×

Role	View	Create	Edit	Execute
Viewer	√	×	×	×
Product manager	√	×	×	×
System engineer	√	√	√	√
Committer	√	√	√	√

Environment permissions

The following table lists the release environment permissions for each role in a project in the initial state.

Table 2-7 Project-level development environment permissions

Role	View	Create	Edit	Delete	Execute	Roll Back
Project admin	√	√	√	√	√	√
Project manager	√	√	√	√	√	√
Developer	√	√	√	√	√	√
Test manager	√	×	×	×	×	×
Tester	√	×	×	×	×	×
Participant	√	×	×	×	×	×
Viewer	√	×	×	×	×	×
Product manager	√	√	√	√	√	√
System engineer	√	√	√	√	√	√
Committer	√	√	√	√	√	√

Table 2-8 Project-level test environment permissions

Role	View	Create	Edit	Delete	Execute	Roll Back
Project admin	√	√	√	√	√	√
Project manager	√	√	√	√	√	√
Developer	√	×	×	×	×	×
Test manager	√	√	√	√	√	√
Tester	√	√	√	√	√	×
Participant	√	×	×	×	×	×
Viewer	√	×	×	×	×	×
Product manager	√	×	×	×	×	×
System engineer	√	×	×	×	×	×
Committer	√	√	√	√	√	√

Table 2-9 Project-level pre-production environment permissions

Role	View	Create	Edit	Delete	Execute	Roll Back
Project admin	√	√	√	√	√	√
Project manager	√	√	√	√	√	√
Developer	√	×	×	×	×	×
Test manager	√	×	×	×	×	×
Tester	√	×	×	×	×	×
Participant	×	×	×	×	×	×
Viewer	×	×	×	×	×	×

Role	View	Create	Edit	Delete	Execute	Roll Back
Product manager	√	×	×	×	×	×
System engineer	√	×	×	×	×	×
Committer	√	√	√	√	√	√

Table 2-10 Project-level production permissions

Role	View	Create	Edit	Delete	Execute	Roll Back
Project admin	√	√	√	√	√	√
Project manager	√	√	√	√	√	√
Developer	×	×	×	×	×	×
Test manager	×	×	×	×	×	×
Tester	×	×	×	×	×	×
Participant	×	×	×	×	×	×
Viewer	×	×	×	×	×	×
Product manager	×	×	×	×	×	×
System engineer	√	×	×	×	×	×
Committer	√	√	√	√	√	√

Parameter group permissions

The following table lists the parameter group permissions for each role in a project in the initial state.

Table 2-11 Project-level parameter group permissions

Role	Create	Delete	Edit	Associate
Project admin	√	√	√	√
Project manager	√	√	√	√
Developer	√	√	√	√
Test manager	×	×	×	×
Tester	×	×	×	×
Participant	×	×	×	×
Viewer	×	×	×	×
Product manager	×	×	×	×
System engineer	√	√	√	√
Committer	√	√	√	√

- Resource-level permissions

  You can configure permissions for a single pipeline by role or user. For details, see **Configuring Pipeline Permissions**.

  Role permissions

  – The project admin, pipeline creator, and project manager can change pipeline role permissions.
  – By default, role permissions for a pipeline are the same as the role permissions at the project level. If role permissions at the project level are changed, role permissions in a pipeline will be changed accordingly.
  – If you change the role permissions for a pipeline, the changed permissions will take effect, because the resource-level permissions take precedence over the project-level permissions.

  User permissions

  – The project admin, pipeline creator, and project manager can change pipeline user permissions.
  – By default, user and role permissions are consistent. If pipeline role permissions are changed, pipeline user permissions will be changed accordingly.
  – If you change the pipeline user permissions, the changed permissions will take effect, because user permissions take precedence over role permissions.

  Use project-level permissions

  – If this function is enabled, the role permissions of a pipeline become the same as those in the project settings.

- If this function is disabled, you can customize the role permissions of a specific pipeline.
- User permissions take precedence over role permissions.

3 Accessing CodeArts Pipeline

This section describes how to access CodeArts Pipeline.

Prerequisites

You have **enabled and authorized CodeArts Pipeline**.

Accessing Through the Homepage

Step 1 **Log in to the Huawei Cloud console**.

Step 2 Click  in the upper left corner of the page and choose **Developer Services** > **CodeArts Pipeline** from the service list.

Step 3 Click **Access Service** to access the Pipeline homepage.

Click  in the upper left corner of the page and select a region.

----End

Accessing Through a Project

Step 1 **Log in to the Huawei Cloud console**.

Step 2 Click  in the upper left corner of the page and choose **Developer Services** > **CodeArts Pipeline** from the service list.

Step 3 Click **Access Service** to access the CodeArts Pipeline homepage.

Step 4 On the top navigation bar, click **Homepage** to access the CodeArts homepage.

Step 5 Click a project name to access the project.

Step 6 In the left navigation pane, choose **CICD** > **Pipeline** to access the pipeline list.

Click  in the upper left corner of the page and select a region.

----End

4 Creating a Pipeline

4.1 Creating a Pipeline

Using the GUI simplifies pipeline creation and configuration, making it beginner-friendly and efficient. Use YAML to create pipelines to manage their configurations by version. This allows easy rollback to earlier versions and access to historical records. The method also ensures reusable configuration, minimizing errors.

YAML pipelines are easy to iterate and optimize. They facilitate team collaboration and make complex projects easier to manage, thus boosting development and deployment efficiency.

Notes and Constraints

- In the initial state, you can create a pipeline if you are any of the following roles:
  - Project administrator
  - Project manager
  - Developer
  - System engineer
  - Committer
- Project administrators can modify the permissions of other roles. For details, see **Authorizing CodeArts Pipeline**.

4.2 Creating a Pipeline with the GUI

Preparations

- You have **enabled and authorized CodeArts Pipeline**.
- **Create a project**.
- If you use a CodeArts Repo repository, **create a code repository**.
- If you want to enhance permissions to do operations on CodeArts Repo or connect to a third-party repository, create a service endpoint.

Notes and Constraints

- GitLab pipeline source is available in **LA-Mexico City2**, **LA-Sao Paulo1**, and **AP-Singapore** regions.
- Create a custom pipeline template.
  - System templates are used to clone or generate pipelines. They cannot be edited or deleted.
  - Pipeline source is not required for a template.
  - Stage admission configuration is not supported for template orchestration.

Creating a Pipeline

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 Click **Create Pipeline**. Configure parameters by referring to **Table 4-1**.

Table 4-1 Basic information

Parameter	Description	Example Value
Name	Pipeline name. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.	Enter **pipeline01**.
Agency URN	Unique identifier of IAM agency. If set: uses this agency identity and permissions to access other cloud services. If not set: uses the operator's identity and permissions.	-
Project	Project that a pipeline belongs to. • If you access CodeArts Pipeline through the homepage, select a project as needed. • If you access CodeArts Pipeline through a project, the parameter cannot be changed.	The default project is **project01**.

Parameter	Description	Example Value
Pipeline Source	Code source associated with the pipeline: • CodeArts code source: Repo: provides comprehensive code hosting services for enterprises and Git-based online code hosting services for software developers. • Third-party code source – GitHub: After connecting to a GitHub account, you can obtain the repository and branch information of that account. – GitLab: After connecting to a GitLab account, you can obtain the repository and branch information of that account. – Bitbucket: After connecting to a Bitbucket account, you can obtain the repository and branch information of that account. – Git: After connecting to a Git repository, you can obtain its branch information. – Repo (other projects): Code is pulled from CodeArts Repo repositories of other projects. Select a project, and then select a code repository and a default branch. • Artifact source Use SWR as the pipeline source to run the pipeline and generate system parameters such as the artifact name, download address, and version number for extensions. **NOTE** • If you do not need to associate the pipeline with a code repository, you can select **None**. In this case, executing a job that should be associated with a repository will result in an error. For details, see **FAQ**.	Select **Repo**.
Orchestration Method	Method of orchestrating a pipeline. If you select **Repo** as the code source, you can choose either of the following ways to orchestrate a pipeline. • Graphical: Uses the GUI to clearly display serial and parallel jobs. • YAML: Uses YAML (One YAML file can be used for multiple pipelines). Syntax auto-completion and validation available. For details, see **Creating a Pipeline with YAML**.	Select **Graphical**.
Service Endpoint	You need to use a service endpoint to connect to a third-party repository. Select an endpoint created in **Preparations** or click **Create One** to create an endpoint. For details, see **Creating Service Endpoints**.	-

Parameter	Description	Example Value
Workspace	Workspace of the Bitbucket account.	-
Repository	Repository associated with the pipeline. You can select an existing repository or click **Create Repository** to create a repository. For details, see **Creating a Repository Using a Template**.	Select **Repo01**.
Default Branch	Branch used when a pipeline is executed manually or at a specified time.	Select **master**.
Repo Endpoint	Optional. Configure an endpoint to enhance permissions for Repo. Endpoints are used for change-triggered pipelines and repository operation extensions. You can select an endpoint created in **Preparations** or click **Create One** to create an endpoint. For details, see **Creating Service Endpoints**.	-
Alias	Optional. After you set a repository alias, system parameters will be generated based on the alias. For example, *Alias*_**REPOSITORY_NAME** indicates the repository name. You can check the generated parameters on the **Parameter Configuration** page and reference them in a pipeline in the format of *${Parameter name}*. Enter a maximum of 128 characters, including only letters, digits, and underscores (_).	-
Description	Optional. Pipeline description. Enter a maximum of 1,024 characters.	-
Organization	If **Pipeline Source** is set to **SWR**, select the SWR organization. Organizations isolate and manage images within each company or department.	-
Image Name	If **Pipeline Source** is set to **SWR**, select an image in the organization.	-
Specified Version	Image version selected for the **SWR** pipeline source.	-

Parameter	Description	Example Value
Source Alias	If you select **SWR** for pipeline source, after you set an artifact source alias, system parameters will be generated based on the alias. For example, *Alias*_**ARTIFACT_NAME** indicates the artifact name. You can check the generated parameters on the **Parameter Configuration** page and reference them in a pipeline in the format of *${Parameter name}*. Enter a maximum of 128 characters, including only letters, digits, underscores (_), hyphens (-), and periods (.).	-

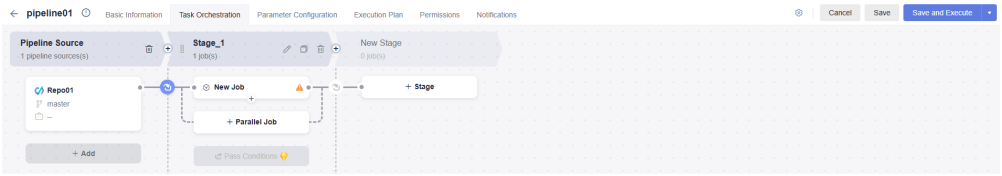
Step 3 Click **Next**. The **Select Template** page is displayed. The following uses the **Blank Template** as an example.

- System or custom templates: Jobs will be automatically generated based on the selected template. For more information, see **Managing Pipeline Templates**.
- **Blank Template**: Create a pipeline from scratch.

Step 4 Click **OK** to create a pipeline.

The **Task Orchestration** page is displayed. You can **configure the pipeline** or click **Cancel** to return to the pipeline list.

Figure 4-1 Task orchestration



----End

Managing Pipeline Templates

CodeArts Pipeline provides system templates and allows you to customize templates. You can use templates to quickly create continuous delivery pipelines and standardize the delivery process.

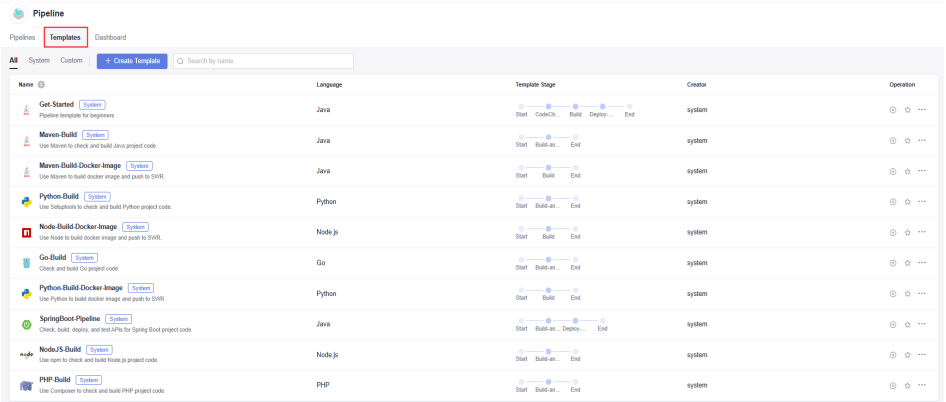
Table 4-2 System templates

Template	Description
Maven-Build	• Build template: Maven • Language: Java • Tools: Maven 3.5.3 and JDK 1.8

Template	Description
Python-Build	• Build template: Setuptools • Language: Python • Tool: Python 3.6
Go-Build	• Build template: Go • Language: Go • Tool: Go 1.10.3
SpringBoot-Pipeline	• Build template: Maven • Language: Java • Tools: Maven 3.5.3 and JDK 1.8
NodeJS-Build	• Build template: Node.js • Language: JavaScript • Tool: Node.js 12.7.0
PHP-Build	• Build template: PHP • Language: PHP • Tool: PHP 7.3.3

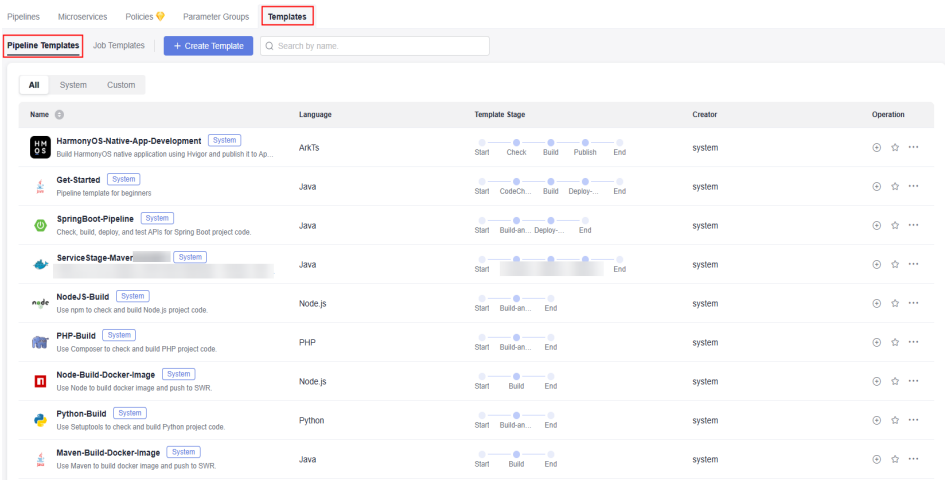
- Access pipeline templates
  - Homepage: Access the CodeArts Pipeline homepage, and switch to the **Templates** tab page.

    Figure 4-2 Pipeline templates

  - Project: Access the pipeline list in a project, and switch to the **Templates** tab.

Figure 4-3 Pipeline templates



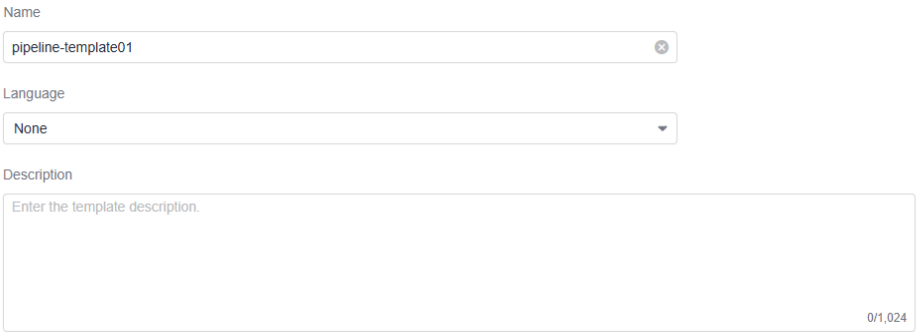
You can perform the following operations on pipeline templates.

Table 4-3 Operations on templates

Parameter	Description
⊕	Click this icon, you will be redirected to the page where you can quickly create a pipeline using a template.
☆	Click this icon to favorite a template. After a template is favorited, the icon changes to ★. You can click ★ to unfavorite the template.
•••	• Click this icon and select **Edit**. On the displayed **Task Orchestration** tab page, you can edit the template. • Click this icon and select **Clone**. On the displayed **Task Orchestration** tab page, you can clone the template. • Click this icon and select **Delete**. In the displayed window, delete the template as prompted.

- Customize a pipeline template
  a. Access pipeline templates.
  b. Click **Create Template**. The **Task Orchestration** page is displayed.
  c. Configure basic information, stages/jobs, and parameters.
     - **Basic Information**: Specify the template name, language (Java, Python, Node.js, Go, .NET, C++, PHP, ArkTS), and description (optional). Language is **None** by default.

Figure 4-4 Basic information



- **Task Orchestration**: You can add official, third-party, and custom extensions to a pipeline template. After tasks of build, code check, deployment, and API test are configured in a template, corresponding tasks will be created when you create a pipeline using this template.
- **Parameter Configuration**: Add parameters to the template. Pipeline template parameters include custom and predefined parameters. Custom parameters include string, auto-increment, and enumeration types. For details about how to configure parameters, see **Configuring Custom Parameters**.

d. Click **Save**. A message is displayed, indicating that the template is created successfully.

Tutorial Video

This video demonstrates how to create and execute a pipeline.

4.3 Creating a Pipeline with YAML

Preparations

- You have **enabled and authorized CodeArts Pipeline**.
- **Create a project**.
- **Create a code repository**.
- Prepare a YAML file.

  You can create a YAML file or orchestrate a YAML file in advance. A YAML file usually consists of the triggering mode **on**, parameter **env**, and job **jobs**. For details, see **YAML Syntax**.

Notes and Constraints

You can only use **Repo** as the pipeline source to create a YAML-based pipeline.

YAML File Example

The following YAML outlines a pipeline configuration. It consists of a build, a code check, and a deployment job in serial mode, and references pipeline parameters in the build job.

```
env:                                    # Define environment variables as key-value pairs. Environment variables can be referenced in any job within the pipeline.
  image_version: 1.0.0

jobs:                                   # Define jobs included in the pipeline.
  build:                                # Job ID, which defines the unique identifier of the job.
    name: maven build                       # Job name, which is displayed on the GUI.
    steps:                              # Define the steps within the job.
      - name: My build step                 # Step name, which is displayed on the GUI.
        uses: CodeArtsBuild                 # Extension used for this step.
        with:                           # Define the extension's runtime parameters as key-value pairs. Variables defined in "env" can be referenced.
          jobId: 878b4d13cb284d9e8f33f988a902f57c    # Job ID. On the job details page, copy the 32-bit string at the end of the browser URL to obtain the ID.
          artifactIdentifier: my_image
          version: ${{ env.image_version }}
  check:
    name: code check
    steps:
      - name: My check step
        uses: CodeArtsCheck
        with:
          jobId: 43885d46e13d4bf583d3a648e9b39d1e
          checkMode: full
  deploy:
    name: cce deploy
    needs:                              # Specify that this job should run only after the listed jobs have completed.
      - build
      - check
    if: ${{ completed() }}                  # Specify the condition under which this job should run.
    steps:
      - name: My deploy step
        uses: CodeArtsDeploy
        with:
          jobId: 9c5a5cda6ffa4ab583380f5a014b2b31
          version: ${{ env.image_version }}
```

Creating a YAML Pipeline

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 Click **Create Pipeline**. Configure parameters by referring to **Table 4-4**.

Table 4-4 Pipeline basic information

Parameter	Description	Example Value
Name	Pipeline name. Enter a maximum of 128 characters, including only letters, digits, underscores (_), and hyphens (-).	Enter **pipeline02**.
Agency URN	Unique identifier of IAM agency. If set: uses this agency identity and permissions to access other cloud services. If not set: uses the operator's identity and permissions.	-

Parameter	Description	Example Value
Project	Project that a pipeline belongs to. • If you access CodeArts Pipeline through the homepage, select a project as needed. • If you access CodeArts Pipeline through a project, the project displayed here defaults to the current project and cannot be changed.	The default project is **project01**.
Pipeline Source	Code source associated with the pipeline. Select **Repo** (CodeArts Repo). It provides comprehensive code hosting services for enterprises and Git-based online code hosting services for software developers.	Select **Repo**.
Orchestration Method	Method of orchestrating a pipeline. Select **YAML**: Use YAML to orchestrate a pipeline (one YAML file can be used for multiple pipelines). Syntax auto-completion and validation are available.	Select **YAML**.
Repository	Repository associated with the pipeline. You can select an existing repository or click **Create Repository** to create a repository. For details, see **Creating a Repository Using a Template**.	Select **Repo02**.
Default Branch	Branch used when a pipeline is executed manually or at a specified time.	Select **master**.
Configuration File	• **New**: Create a YAML file when orchestrating a pipeline. • **Existing**: Orchestrate a pipeline based on the existing YAML file. The orchestrated content will overwrite the original YAML file. For details about how to write a YAML file, see **YAML Syntax**.	Select **New**.
YAML File	This parameter is mandatory when **Configuration File** is set to **Existing**. Select a branch and enter the relative path of the YAML file.	-
Repo Endpoint	Optional. Configure an endpoint to enhance permissions for Repo. Endpoints are used for change-triggered pipelines and repository operation extensions. You can select an endpoint created in **Preparations** or click **Create One** to create an endpoint. For details, see **Creating Service Endpoints**.	-

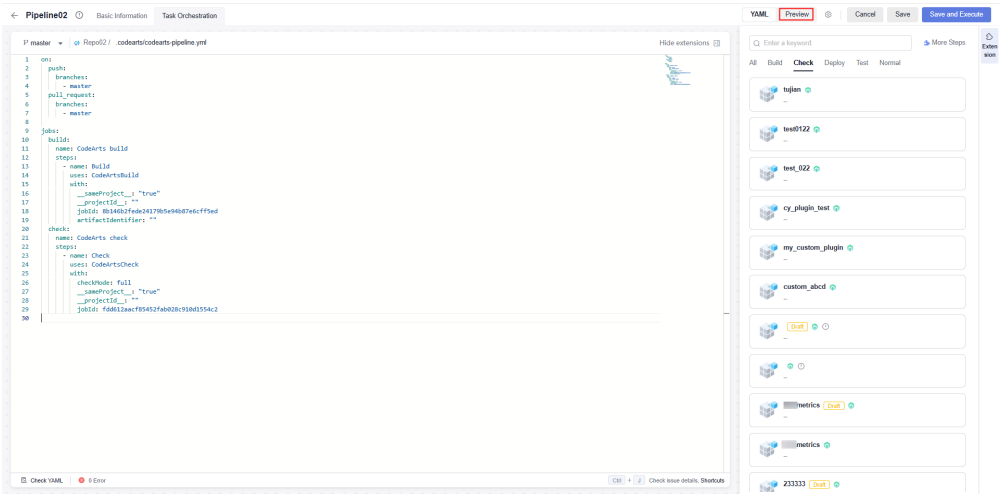
Parameter	Description	Example Value
Alias	Optional. After you set a repository alias, system parameters will be generated based on the alias. For example, *Alias*_**REPOSITORY_NAME** indicates the repository name. You can check the generated parameters on the **Parameter Configuration** page and reference them in a pipeline in the format of *${Parameter name}*. Enter a maximum of 128 characters, including only letters, digits, and underscores (_).	-
Description	Optional. Pipeline description. Enter a maximum of 1,024 characters.	-

Step 3 After configuring the basic information, click **OK**. The **Task Orchestration** page is displayed.

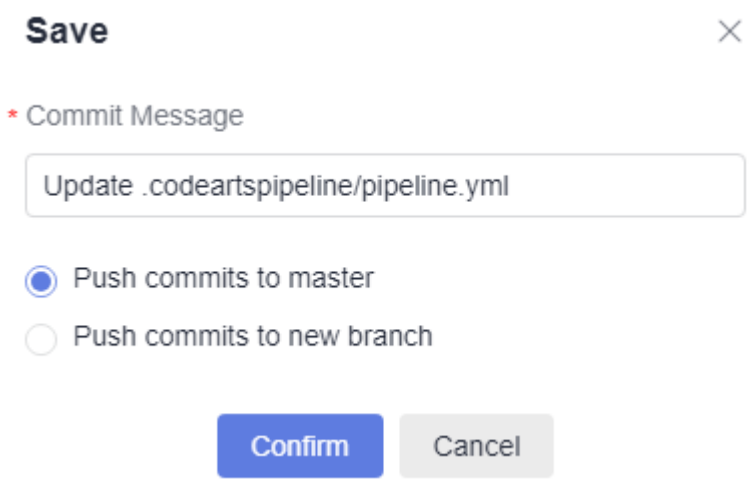
- You can edit the YAML file on the left. For details, see **YAML Syntax**. The following uses serial tasks of build and code check as an example.
- You can add extensions to the YAML file from the extension list displayed on the right.

You can verify YAML syntax during orchestration. Click **Preview** to switch to the graphical user interface.

Figure 4-5 Task orchestration



Step 4 After orchestration, click **Save**, enter the commits message, and push commits.



- Push commits to the existing branch: If you create the pipeline with a new YAML file, commits will be pushed to the default branch. If you create a pipeline with an existing YAML file, commits will be pushed to the branch where the YAML file resides.
- Push commits to a new branch: Commits will be pushed to a new branch. If you selected **Create merge request**, a merge request will be created for the new branch and the existing branch.

Step 5 Click **Confirm**. A message is displayed, indicating that the pipeline is created.

----End

Tutorial Video

This video demonstrates how to create and execute a pipeline with YAML.

5 Configuring a Pipeline

5.1 Orchestrating Pipeline Stages

Pipeline Stage

A stage is a basic part of a pipeline. Each stage organizes jobs, processes inputs using specific logic, and produces outputs. Stages run in a defined order to create clear workflows. Refer to this section to configure stages.

Notes and Constraints

- Configure pass conditions:
  - Only the **Pass-Conditions-of-Standard-Policies** type is available. You can select a project- or tenant-level policy.
  - Pass conditions can be set exclusively for each stage of a pipeline and is valid only for the corresponding stage.
  - You can set multiple pass conditions for one stage.

Billing

To use rules and policies, upgrade the CodeArts package to the basic edition. For details, see **CodeArts Billing Modes**.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Configuring Stages

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 Locate the target pipeline from the pipeline list, click ··· in the **Operation** column, and select **Edit**.

Step 3 On the **Task Orchestration** tab page, click or **Stage** to add a stage to the pipeline. After the stage is added, you can edit, clone, delete, or move it, or configure its entry type.

Figure 5-1 Task orchestration

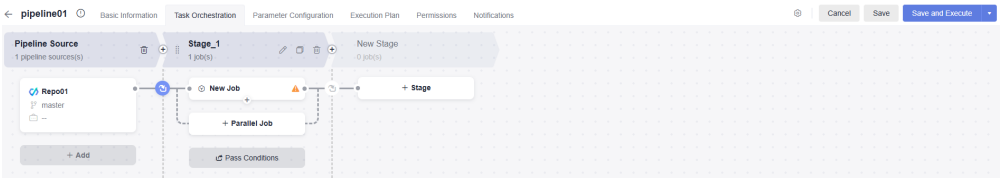
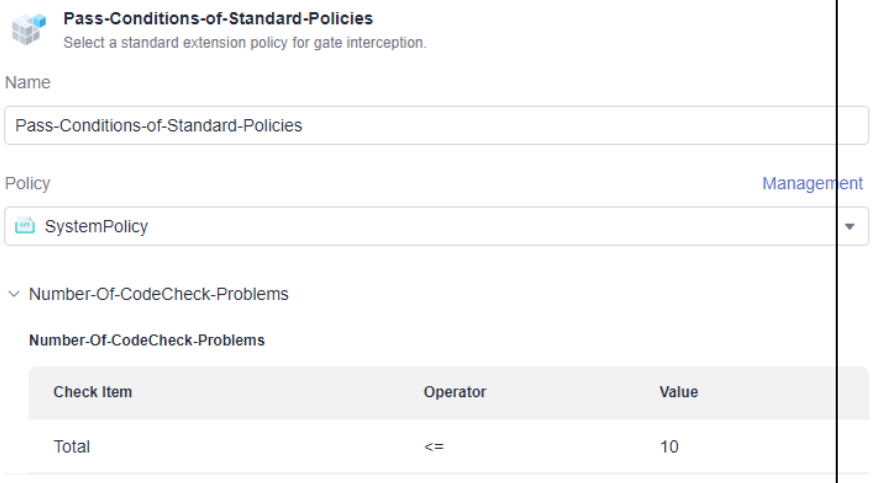
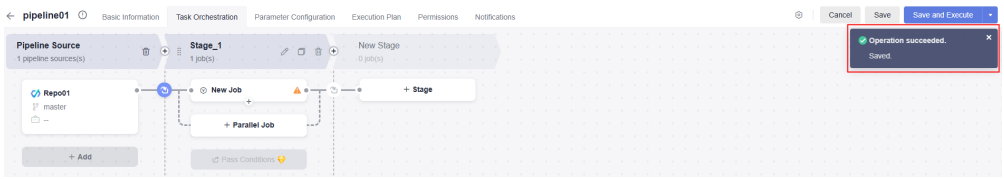


Table 5-1 Stage management

Operation	Description
Editing a stage	Click . In the displayed window, configure the stage name and whether to always run jobs in the stage. • If you select **Yes** for **Always Run**, jobs in this stage will be executed by default and cannot be canceled. • If you select **No** for **Always Run**, jobs in this stage will be selected by default for execution but can be canceled.
Cloning a stage	Click to clone a stage.
Deleting a stage	Click and confirm the deletion as prompted.
Sorting a stage	Click and drag to move a stage to adjust its sequence.
Setting stage admission	Specify in what conditions a pipeline can proceed to the next stage. Click . In the displayed window, configure the admission type. • **Automatic** (default): The pipeline automatically proceeds to the next stage. • **Manual**: The pipeline proceeds only after manual confirmation. • **Time window**: The pipeline proceeds to the next stage within a specified period.

Operation	Description
Pass conditions	Set rules and policies to decide in what conditions a pipeline can complete the current stage. • A rule defines different comparison operators based on extension thresholds to determine whether to apply the extension output to stage pass conditions. Multiple such rules constitute a policy. For details, see **Configuring a Rule**. • A policy is a set of rules defined as different condition templates for a step extension. Each policy can be applied to multiple pipelines. There are **tenant-level policies** and **project-level policies**. You can use them to control pipeline runs for high-quality delivery. 1. Click **Pass Conditions** under a stage. In the displayed window, move the cursor to the pass conditions card and click **Add**. 2. Enter a name and select a policy.  3. Click **OK**.

Step 4 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.



----End

Configuring a Rule

Rules are tenant-level resources and can be used in all tenant- or project-level policies of the current tenant.

Step 1 Access the CodeArts Pipeline homepage.

Step 2 Click the username in the upper right corner and click **All Account Settings**.

Step 3 In the navigation pane on the left, choose **Policy Management** > **Rules**.

Step 4 Click **Create Rule**. On the displayed page, configure parameters.

Figure 5-2 Creating a rule

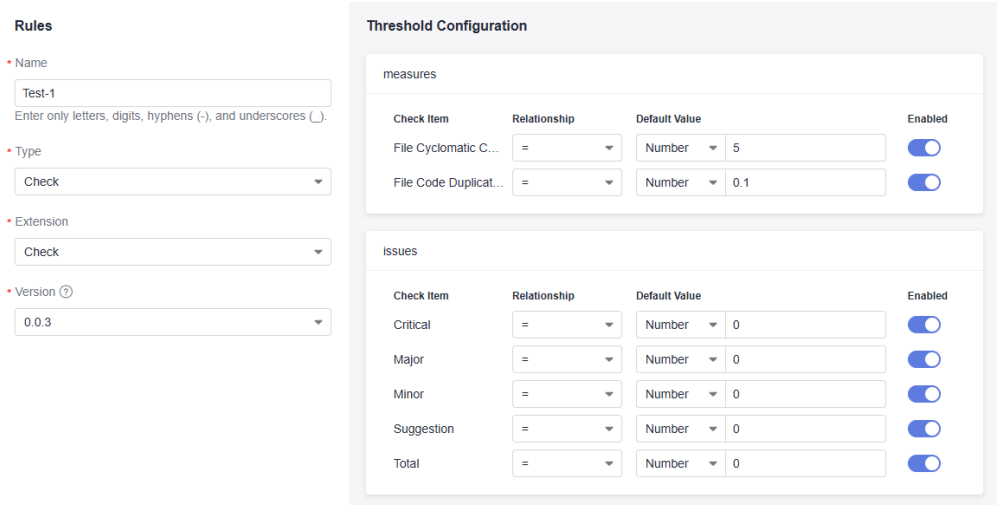
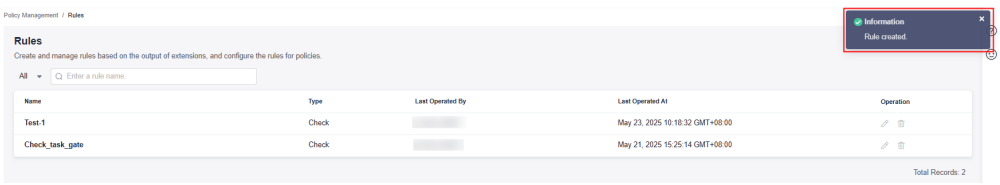


Table 5-2 Rule parameters

Parameter	Description	Example Value
Name	Rule name, which is generated based on the current time. Enter a maximum of 128 characters, including only letters, digits, underscores (_), and hyphens (-).	Enter **Test-1**.
Type	Rule type, which corresponds to the extension type. Supported extension types: **Build**, **Check**, and **Test**. • **Build**: extensions for code build. • **Check**: extensions for code check. • **Test**: extensions for API tests.	Select **Check**.

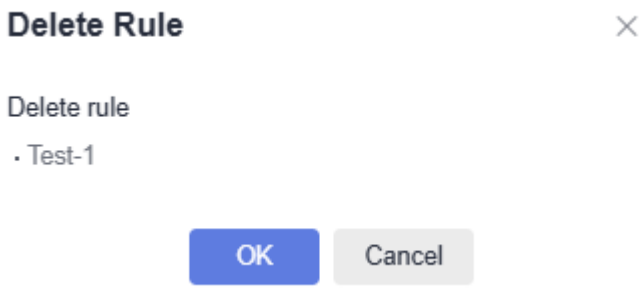
Parameter	Description	Example Value
Extension	Select from all extensions of the selected type. • Extensions of the **Build** type: Set build result thresholds. For example, you can select an official **Build** extension to set thresholds for Maven unit tests. • Extensions of the **Check** type: Set code check result thresholds. For example, you can select an official **Check** extension to set thresholds for code checks. • Extensions of the **Test** type: Set test result thresholds. For example, you can select an official **TestPlan** extension to set thresholds for determining the test case pass rate in test suites.	Select **Check**.
Version	Select from all extension versions with threshold settings.	Select **0.0.3**.
Threshold Configuration	Set relational operators based on thresholds automatically obtained from the selected extension version. Note that if you changed threshold settings, the relevant rule and policy would also be changed. If you set the relational operator to **Exclude** or **Include**, the threshold can only be the **Text** type. For the check item **Pass Ratio**, the value ranges from 0 to 1.	-
Code metrics	Reflect the quality of committed code. Metrics help you detect and fix issues in a timely manner and improve R&D efficiency. Code metrics cover the following two types of issues: • Cyclomatic complexity. • Duplication rate. CodeArts Check identifies duplicate lines, blocks, and rates. For details, see **Viewing Check Results**.	-

Step 5 Click **Confirm**. A message is displayed, indicating that the rule is created successfully. You can also perform the following operations in the rule list.



- On the rule list page, click in the **Operation** column to edit a rule.
  - The rule type cannot be edited.

  - After a rule is edited, all policies that reference the rule are automatically modified.
- On the rule list page, click in the **Operation** column to delete a rule. On the displayed dialog box, click **OK** to confirm the deletion.
  - After a rule is deleted, all policies that reference the rule automatically cancel the reference.



----End

Configuring a Pipeline Policy

There are tenant-level policies and project-level policies. Tenant-level policies are tenant-level resources and can be used in pass conditions for all pipelines of the current tenant. There is a system policy by default. You can check and use the policy, but cannot edit or delete it. Project-level policies are project-level resources and can be used in pass conditions for all pipelines of the current project. A maximum of 20 rules can be selected for a tenant-level or project-level policy.

Configuring the Tenant-level Policy

Step 1 Access the CodeArts Pipeline homepage.

Step 2 Click the username in the upper right corner and click **All Account Settings**.

Step 3 In the navigation pane on the left, choose **Policy Management** > **Policies**. The policy list page is displayed.

Step 4 Click **Create Policy**. On the displayed page, configure parameters.

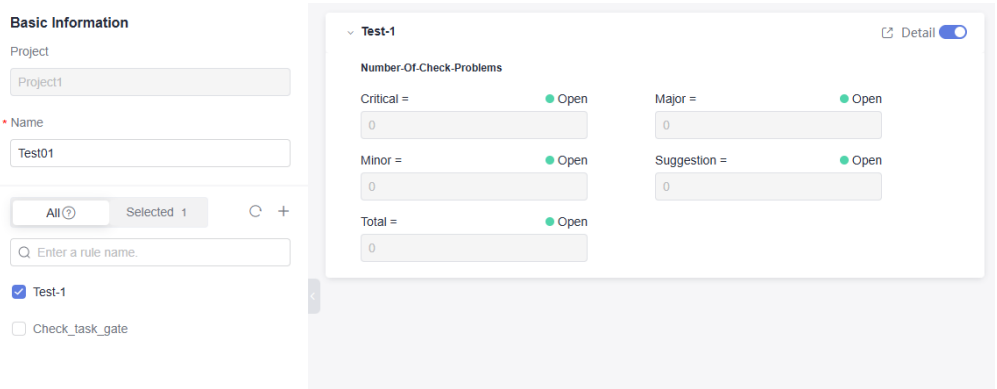
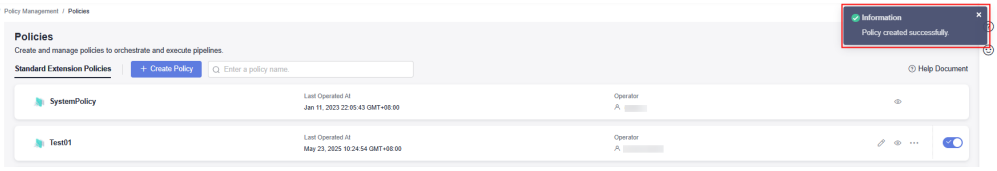


Table 5-3 Policy parameters

Parameter	Description	Example Value
Name	Policy name, which is generated based on the current time by default. Enter a maximum of 128 characters, including only letters, digits, underscores (_), and hyphens (-).	Enter **Test01**.
Rule	The selected rules will be displayed in the right part of the page. You can perform the following operations on each rule: • Edit: Click **Detail** in the upper right corner of the rule to check details. Click **Edit** in the upper right corner to edit the rule. • Enable/Disable: Click the toggle in the upper right corner to enable/disable the rule. After the rule is disabled, it will not take effect in the pass conditions where it is referenced.	Select **Test-1**.

Step 5 Click **Confirm**. A message is displayed, indicating that a policy is generated.



- On the policy list page, click to edit a policy. You can also perform the following operations in the policy list.
- Click to check a policy, and click **Edit** in the upper right corner to edit the policy.
- Click and select **Clone** to clone a policy.
- Click and select **Delete**. On the displayed dialog box, confirm the deletion. When you delete a policy, the system displays a message indicating the number of pipelines that use the policy. Once the policy is deleted, pipeline execution will fail.
- Click the toggle on the right to enable or disable a policy. If a policy is disabled, the system displays a message indicating the number of pipelines that use the policy. Once the policy is disabled, it will not take effect in pass conditions.

----End

Configuring Project-Level Policies

Step 1 Access the CodeArts Pipeline homepage through a project.

Step 2 Click the **Policies** tab. The policy list page is displayed.

Step 3 Click **Create Policy**. On the displayed page, configure parameters.

Figure 5-3 Policy parameters

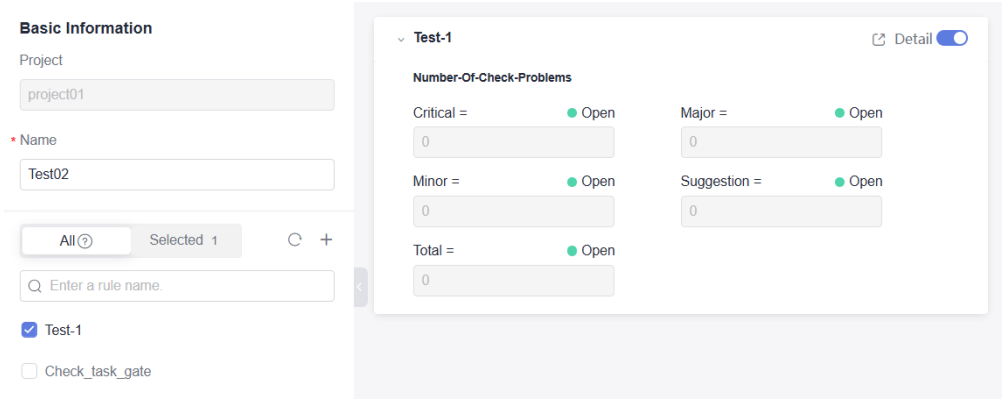
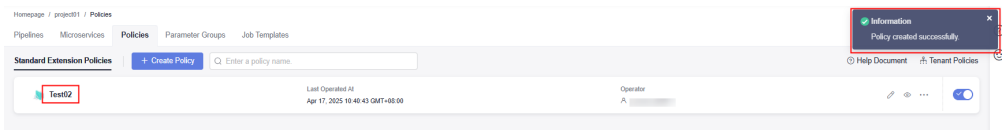


Table 5-4 Policy parameters

Parameter	Description	Example Value
Name	Policy name, which is generated based on the current time by default. Enter a maximum of 128 characters, including letters, digits, underscores (_), and hyphens (-).	Enter **Test02**.
Rule	The selected rules will be displayed in the right part of the page. You can perform the following operations on each rule: • Edit: Click **Detail** in the upper right corner of the rule to view the details. Click **Edit** in the upper right corner to edit the rule. • Enable/Disable: Click the toggle in the upper right corner to enable/disable the rule. After the rule is disabled, it will not take effect in the pass conditions.	Select **Test-1**.

Step 4 Click **Confirm**. A message is displayed, indicating that the policy is created successfully.



- On the policy list page, click  to edit a policy. You can also perform the following operations in the policy list.
- Click  to check a policy. Click **Edit** in the upper right corner to edit the policy.
- Click and select **Clone** to clone a policy.
- Click and select **Delete**. On the displayed dialog box, confirm the deletion.

When you delete a policy, the system displays a message indicating the number of pipelines that use the policy. Once the policy is deleted, pipeline execution will fail.

- Click the toggle on the right to enable or disable a policy.

  If a policy is disabled, the system displays a message indicating the number of pipelines that use the policy. Once the policy is disabled, it will not take effect in pass conditions.

- On the policy list page, click **Tenant Policies** in the upper right corner. In the displayed window, you can view, clone, or inherit a tenant-level policy.
  - View: Click in the **Operation** column to check the tenant-level policy. Click **Edit** in the upper right corner to edit the tenant-level policy.
  - Clone: Click in the **Operation** column to clone the selected tenant-level policy to create a project-level policy.
  - Inherit: Click in the **Operation** column to inherit a project-level policy from the tenant-level policy. The inherited rules are always consistent with rules of the tenant-level policy.

----End

5.2 Orchestrating Pipeline Jobs

Jobs in a Pipeline

A job is the minimum manageable execution unit in a pipeline. Jobs can be managed and orchestrated in serial and parallel modes, and executed based on a specific sequence and time in a stage. Refer to this section to configure jobs.

Notes and Constraints

You can **configure a pipeline** from scratch or from a template only when you have configured a job template. For details about how to configure a job template, see **Configuring a Job Template**.

Prerequisites

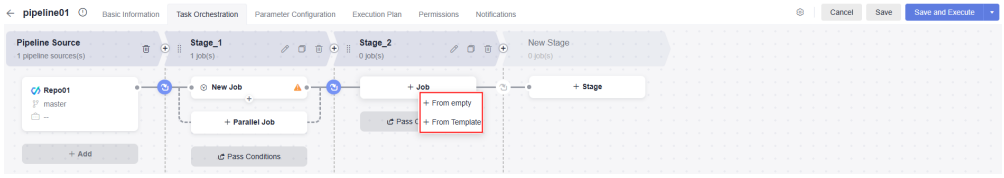
- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Configuring Pipeline Jobs

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the pipeline list page, search for the target pipeline, click ... in the **Operation** column, and click **Edit**.

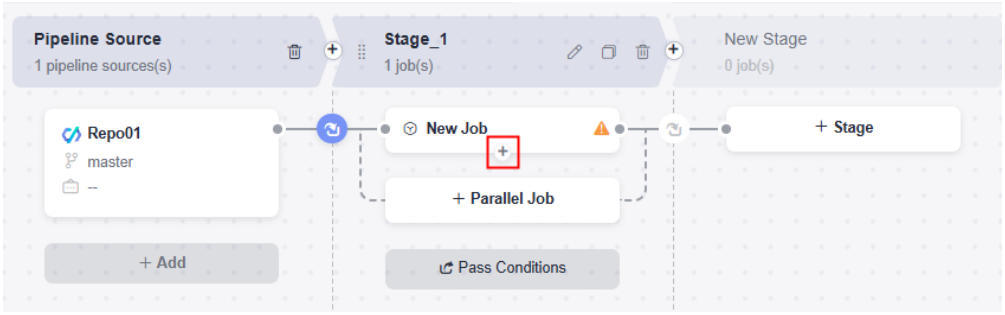
Step 3 On the **Task Orchestration** page, click **Job** under a stage, and select **From empty** or **From Template**. If you use the blank template to create a pipeline, **New Job** in **Stage_1** is empty by default and no job template can be selected.



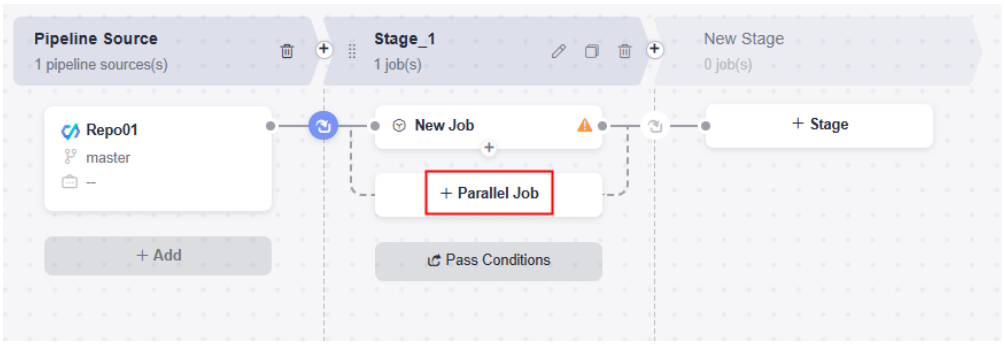
From empty: In the displayed window, configure jobs by referring to **Table 5-5**.

From Template: In the displayed window, select a desired template. For details about templates, see **Table 5-7**.

- Click + under a job to add a serial job. For example, a build job and a deployment job must be executed sequentially.



- Click **Parallel Job** to add a parallel job. For example, a code check job and a build job can be executed at the same time.



Step 4 Configure extensions and information for the job by referring to the following table.

Table 5-5 Job configuration

Operation	Description
Adding an extension	There are five types of extensions: build, code check, deployment, test, and normal extensions. You can filter or search for extensions by type. For more information, see **Managing Pipeline Extensions**. Move the cursor to an extension card and click **Add**. Then configure the following information: • Enter an extension name. • Select a task to be called. You can search for a task. If no proper task is available, create one as prompted. • If the called task has parameters, the parameters will be displayed. Configure parameters as needed. • You can add only one extension with flag *Job* to a single job. Extensions with flag *Draft* indicate that they are draft extensions. • The extension for suspending a pipeline can only be added to stages that do not contain parallel jobs.
Deleting an extension	Move the cursor to an extension card, click ⋮, and select **Delete** to delete the extension.
Replacing an extension	Move the cursor to an extension card, click ⋮, and select **Replace** to replace the extension. Or, click **Replace Step** above the extension name to choose another extension.
Sorting extensions	Click, hold, and move an extension card to adjust the extension sequence.

Operation	Description
Configuring jobs	Configure the following information: • **ID**: The job ID must be unique. Enter a maximum of 128 characters, including letters, digits, hyphens (-), and underscores (_). • **Execution Host**: You can use the built-in executors or create custom ones. You only need to configure executors for non-job-level extensions. – **Built-in executor**: provided by CodeArts Pipeline with out-of-the-box availability. – **Custom executor**: allows you to configure tools and running environments as needed. Before using custom executors, add an agent pool. For details, see **Agent Pools**. • Resource pool advanced settings: A pipeline job can run in the following resource pools. You only need to configure resource pools for non-job-level extensions. – Default resource pool: isolates your environments from others by container. – Dedicated resource pool: grants you exclusive access to the private VPC network via the intranet. – Custom resource pool: manages your server resources that are connected via a network proxy. For details, see **Managing Agent Pools**. **NOTE** Resource pools are available only in AP-Singapore. Executors are available in all regions except AP-Singapore. • Select Job – **Always**: The job will always be selected for execution and cannot be canceled. – **Disabled**: The job cannot be selected for execution. – **Selected by default**: The job is selected for execution by default and can be modified. – **Not selected by default**: Job is not selected for execution by default. • **Execute**: Execution conditions are the triggers for executing jobs in a pipeline. – **Even when previous job is not selected**: The current job is executed regardless of whether the previous serial job is completed or not selected. – **When previous job succeeds**: The current job is executed only when the previous serial job is successfully executed. – **If previous job fails**: The current job is executed only when the previous serial job fails.

Operation	Description
	– **Always**: The current job is always executed regardless of the previous serial job's final state (failed, completed, canceled, or ignored). – **With expression**: When the previous job is **COMPLETED**, **FAILED**, **CANCELED**, or **IGNORED** and the expression result is **true**, the current job will be executed. ▪ **Tag**: Click the text box. In the displayed window, you can select desired contexts, functions, or operators. Or you can just manually enter an expression in the text box. For details about how to configure an expression, see **Configuring Expressions**. Example: To execute the current job regardless of whether the previous job (ID: **job_1**) succeeded or failed, the expression can be as follows: `jobs.job_1.status == 'COMPLETED' || jobs.job_1.status == 'FAILED'` ▪ **Code**: The expression displayed here is in the string format and configured in the **Tag** mode. Manual input is not supported.

Step 5 After configuring the job, click **OK**. After the job is added, you can edit, clone, delete, or move the job.

Table 5-6 Job management

Operation	Description
Editing a job	Click a job card to edit the job.
Cloning a job	Click on the job card to create a serial job.
Deleting a job	Click on the job card and confirm the deletion as prompted.
Sorting jobs	Click, hold, and move a job card to adjust the sequence. Job sequence cannot be adjusted when jobs are executed in parallel.

Step 6 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring an Execution Plan

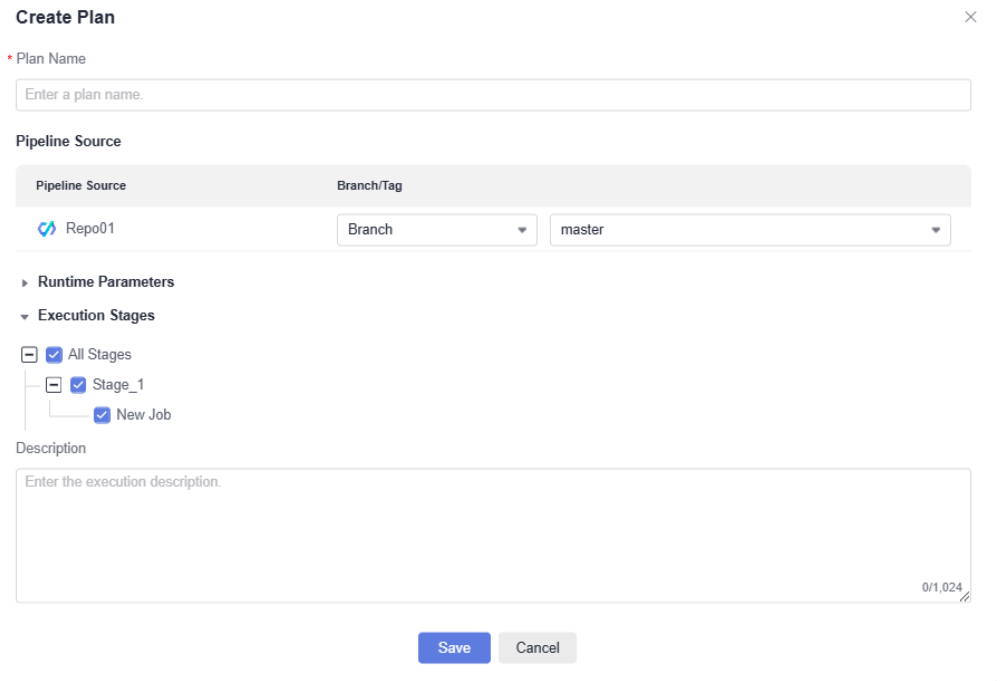
An execution plan outlines job execution steps, resource allocation, monitoring, and management. You can configure an execution plan in advance and select the plan when executing a pipeline manually or through schedulers.

Step 1 Access the CodeArts Pipeline homepage through a project.

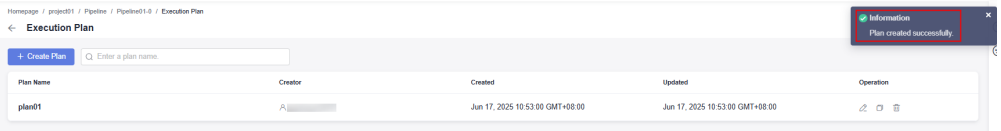
Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Execution Plan**.

Step 3 Click **Create Plan**. In the displayed window, configure the plan name, pipeline source information, runtime parameters, execution stages, and description. For details, see **Executing a Pipeline**.

Figure 5-4 Creating an execution plan



Step 4 Click **Save**. A message is displayed, indicating that an execution plan is created successfully. You can edit, clone, or delete the execution plan.



----End

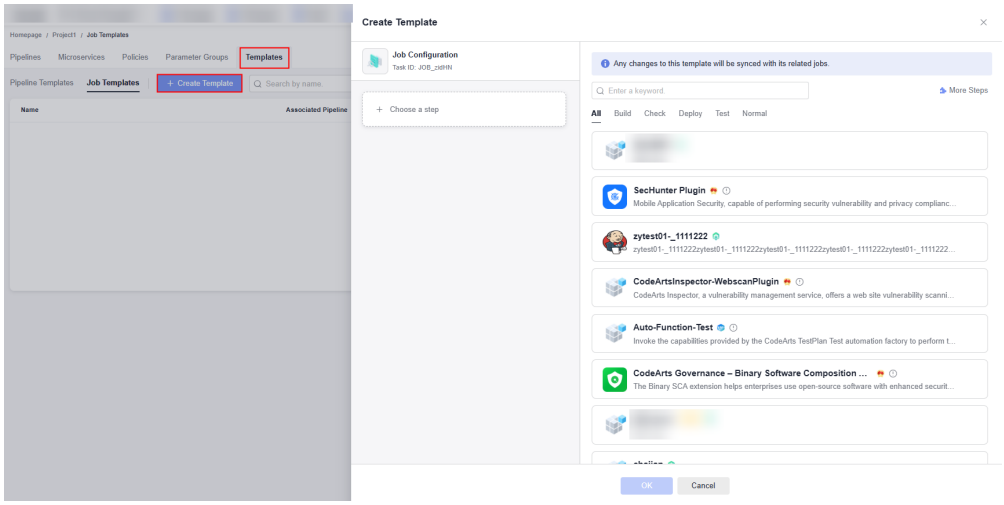
Configuring a Job Template

A pipeline job template is a ready-made model for creating and running specific jobs efficiently. It includes preset configurations and execution logic, saving time on repetitive work.

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 Click **Templates**. In the displayed page, click **Job Templates**.

Step 3 Click **Create Template**. In the displayed window, set parameters.



Step 4 Configure extensions and information for the template by referring to the following table.

Table 5-7 Configuring a job template

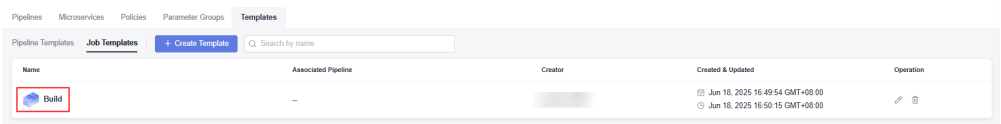
Operation	Description	Example Value
Adding an extension	There are five types of extensions: build, code check, deployment, test, and normal extensions. You can filter or search for extensions by type. For more information, see **Managing Pipeline Extensions**. Move the cursor to an extension card and click **Add**. Then configure the following information: • Enter an extension name. • Select a task to be called. You can search for a task. If no proper task is available, create one as prompted. • If the called task has parameters, the parameters will be displayed. Configure parameters as needed. • You can add only one extension with flag *Job* to a single job. Extensions with flag *Draft* indicate that they are draft extensions.	The following uses the **Build** extension as an example. You need to **create a build task build01** in advance. When adding the extension, set the parameters as follows: • **Name**: **Build** • **Project Config**: **This Project** • **Select Task**: **build01** • **Repository**: **Pipeline**

Operation	Description	Example Value
Deleting an extension	Move the cursor to an extension card, click  , and select **Delete** to delete the extension.	-
Replacing an extension	Move the cursor to an extension card, click  , and select **Replace** to replace the extension. Or, click **Replace Step** above the extension name to choose another extension.	-
Sorting extensions	Click, hold, and move an extension card to adjust the extension sequence.	-

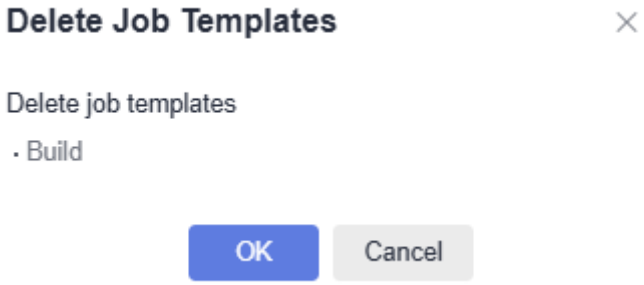
Operation	Description	Example Value
Configuring jobs	Configure the following information: • Basic information: – **Name**: name of the job template. – **Description**: description of the job template. A maximum of 1,024 characters are allowed. • **ID**: The job ID must be unique. Enter a maximum of 128 characters, including letters, digits, hyphens (-), and underscores (_). **NOTE** During pipeline orchestration, you can only edit the job ID in a job template. • Resource pool advanced settings: A pipeline job can run in the following resource pools. You only need to configure resource pools for non-job-level extensions. – Default resource pool: isolates your environments from others by container. – Dedicated resource pool: grants you exclusive access to the private VPC network via the intranet. – Custom resource pool: manages your server resources that are connected via a network proxy. For details, see **Managing Agent Pools**. **NOTE** Resource pools are available only in AP-Singapore. Executors are available in all regions except AP-Singapore. • Select Job – **Always**: The job will always be selected for execution and cannot be canceled. – **Disabled**: The job cannot be selected for execution. – **Selected by default**: The job is selected for execution by default and can be modified. – **Not selected by default**: Job is not selected for execution by default. • **Execute**: Execution conditions are the triggers for executing jobs in a pipeline. – **Even when previous job is not selected**: The current job is executed regardless of whether the previous serial job is completed or not selected.	The following uses the **Build Template** as an example. Set the parameters as follows: • **Name**: **Build Template** • **ID**: **JOB_KJbee** • **Select Job**: use the default value • **Execute**: **Even when previous job is not selected**

Operation	Description	Example Value
	– **When previous job succeeds**: The current job is executed only when the previous serial job is successfully executed. – **If previous job fails**: The current job is executed only when the previous serial job fails. – **Always**: The current job is always executed regardless of the previous serial job's final state (failed, completed, canceled, or ignored). – **With expression**: When the previous job is **COMPLETED**, **FAILED**, **CANCELED**, or **IGNORED** and the expression result is **true**, the current job will be executed. ▪ **Tag**: Click the text box. In the displayed window, you can select desired contexts, functions, or operators. Or you can just manually enter an expression in the text box. For details about how to configure an expression, see **Configuring Expressions**. Example: To execute the current job regardless of whether the previous job (ID: **job_1**) succeeded or failed, the expression can be as follows: `jobs.job_1.status == 'COMPLETED' || jobs.job_1.status == 'FAILED'` ▪ **Code**: The expression displayed here is in the string format and configured in the **Tag** mode. Manual input is not supported.	

Step 5 After the configuration is complete, click **OK**. The template page is displayed.



- Click  to edit the job template.
- Click  to delete the job template.



----End

5.3 Configuring Pipeline Extensions

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

5.3.1 Build

Call the **Build** extension to use CodeArts Build capabilities.

Adding the Extension with GUI

Add the **Build** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-8**.

Table 5-8 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Select Task	Select a build task whose code source is pipeline or the same repository as the pipeline you are configuring.
Repository	Select the corresponding code repository.
Artifact Identifier	Identifier of each build artifact. For example, if the ID of the build task is "build_job" and the artifact identifier is "demo", use "${{jobs.build_job.artifacts.demo}}" to obtain artifact information in subsequent tasks.

Adding the Extension with YAML

Add the **Build** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **Build**

  Call the **Build** extension to use CodeArts Build capabilities.

  ```
  uses: CodeArtsBuild
  with:
    jobId: 878b4d13cb284d9e8f33f988a902f57c
    artifactIdentifier: my_pkg
    customParam: value
  ```

  - **jobId**: ID of the build task. To obtain the ID, copy the 32 digits and letters at the end of the browser URL on the build task details page.
  - **artifactIdentifier**: Build artifact identifier.
  - **customParam**: Parameter value defined in the build task. There may be zero to multiple values.

5.3.2 Check

Call the **Check** extension to use CodeArts Check capabilities.

Adding the Extension with GUI

Add the **Check** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-9**.

Table 5-9 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Select Task	Select a code check task whose code source is the same repository as the pipeline you are configuring.
Repository	Select the corresponding code repository.
Check Mode	Code check mode. You can set check modes as needed to improve the check efficiency.

Adding the Extension with YAML

Add the **Check** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **Check**

  Call the **Check** extension to use CodeArts Check capabilities.

```
uses: CodeArtsCheck
with:
  jobId: 43885d46e13d4bf583d3a648e9b39d1e
  checkMode: full|push_inc_full||push_multi_inc_full
```

  - **jobId**: ID of a code check task.
  - **checkMode**: Check mode.
    - **full**: Checks all code.
    - **push_inc_full**: Checks all files changed in this code commit.
    - **push_multi_inc_full**: Checks all files changed between this code commit and the last successful code commit.

5.3.3 CloudNativeRelease

Orchestrate release policies for environments, such as rolling release and grayscale release.

Adding the Extension with GUI

Add the **CloudNativeRelease** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-10**.

Table 5-10 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Environment Level	Release environment type. Available environment types: development, test, pre-production, and production.
Environment	Environment to be released. For details, see **Creating a Release Environment**.
Artifact Path	Image path for deployment and release. Example: **swr.example.com/demo/springboot-helloworld:v1.1**. You can use ${} to reference pipeline parameters. Example: **swr.example.com/demo/springboot-helloworld:${version}**.

Adding the Extension with YAML

Add the **CloudNativeRelease** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- CloudNativeRelease

```
uses: official_microservice_release
with:
  categories: 8dc56cd6c2cf44029181f04025ed173a
  env_id: 39c94e52fb0e4c03af97e3252c68e2a3
  artifact: swr.example.com/demo/springboot-helloworld:v1.1
```

  - categories: release environment type.
  - env_id: environment to be released.
  - artifact: image path used during deployment.

5.3.4 Deploy

Call the **Deploy** extension to use CodeArts Deploy capabilities.

Adding the Extension with GUI

Add the **Deploy** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-11**.

Table 5-11 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Select Task	Select a deployment task whose code source is the same repository as the pipeline you are configuring.

Adding the Extension with YAML

Add the **Deploy** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **Deploy**

```
uses: CodeArtsDeploy
with:
  jobId: 9c5a5cda6ffa4ab583380f5a014b2b31
  customParam: value
```

  - **jobId**: ID of the deployment task.
  - **customParam**: Parameter value defined in the deployment task.

5.3.5 TestPlan

Call the **TestPlan** extension to use CodeArts TestPlan capabilities.

Adding the Extension with GUI

Add the **TestPlan** extension when you **orchestrate a pipeline**. Set parameter as shown in **Table 5-12**.

Table 5-12 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Select Task	Select an API automation test suite whose code source is the same repository as the pipeline you are configuring.
Parameters	• **Use global parameters of selected environment**: Different environments configured in multiple tasks can be selected and parameters in the corresponding environments can be used. • **Use new execution parameters**: You can directly enter the parameters to overwrite the global parameters used by the selected test task type.
Environment	When **Parameter** is set to **Use global parameters of selected environment**, select **Default Environment**.

Adding the Extension with YAML

Add the **TestPlan** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **TestPlan**

```
uses: CodeArtsTestPlan
with:
  jobId: vb180000vnrgoeib
  environmentModel: 1
  environmentId: 7c2eff2377584811b7981674900158e8
```

  - **jobId**: ID of the API test task.
  - **environmentModel**: parameter source.
    - **0**: Use new execution parameters.
    - **1**: Use the global parameters of the selected environment.
  - **environmentId**: environment ID when **environmentModel** is set to **1**.

5.3.6 Subpipeline

Call the **Subpipeline** extension to configure other pipelines in a project.

Adding the Extension with GUI

Add the **Subpipeline** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-13**.

Table 5-13 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Select Task	Select other pipelines in the current project to be called.
Branch/Tag	Set the branch or tag for the sub-pipeline. You can select **Use subpipeline default branch/tag** or **Use parent pipeline branch/tag**.

Adding the Extension with YAML

Add the **Subpipeline** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **Subpipeline**

```
uses: SubPipeline
with:
  pipelineId: 80ea2d9ffba94c20b9a0a0be47d3a0d8
  branch: master
```

  - **pipelineId**: ID of the called pipeline.
  - **branch**: (Optional) Branch used for running the sub-pipeline.
    - The default branch of the sub-pipeline is used if this parameter is not set.
    - You can reference a parameter or context to define **branch**. For example, if you want to run the parent pipeline source, and the code source alias is **my_repo**, the reference format is **${{sources.my_repo.target_branch}}**.

5.3.7 DelayedExecution

Pause a pipeline for a period of time or until a specified time. You can manually resume or stop a pipeline, or delay the execution for a maximum of three times.

Adding the Extension with GUI

Add the **DelayedExecution** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-14**.

Table 5-14 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Delay By	Delayed execution mode. You can set **Duration** or **Scheduled time**.
Duration	Duration of the delayed execution. You can configure a value.
Scheduled time	Time when a delayed pipeline restarts from. You can configure a value.
Time zone	Select a time zone.

Adding the Extension with YAML

Add the **DelayedExecution** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **DelayedExecution**

```
uses: Delay
with:
  timerType: delay|scheduled
  delayTime: 300
  scheduledTime: '00:00'
  timeZone: China Standard Time
```

  - **timerType**: Delay type. **delay** indicates pausing a pipeline for a period of time. **scheduled** indicates pausing a pipeline until a specified time.
  - **delayTime**: Time duration (in seconds) when **timerType** is set to **delay**.
  - **scheduledTime**: Exact time when **timerType** is set to **scheduled**.
  - **timeZone**: Time zone.

5.3.8 ManualReview

Create manual review tasks by assigning one person or one group. An email notification can be sent to reviewers.

Adding the Extension with GUI

Add the **ManualReview** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-15**.

Table 5-15 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Review Source	Source of the reviewer. Select **Project Members** or **Project Roles**.
Reviewer	Configure reviewers.
Review Role	When **Review Source** is set to **Project Roles**, select review roles. In an IPD project, the project creator does not participate in the review as a project manager.
Review Mode	Select **Review by all** or **Review by any person**.
Timeout Processing	Configure the processing method when the review times out.
Review Duration	How long the review lasts, which cannot be longer than three days.
Description	Description of the review task.

Adding the Extension with YAML

Add the **ManualReview** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **ManualReview**

```
uses: Checkpoint
with:
  mode: members|roles
  approvers: 05d8ca972f114765a8984795a8aa4d41
  roles: PROJECT_MANAGER
  checkStrategy: all|any
  timeout: 300
  timeoutStrategy: reject|pass
  comment: comment
```

  - **mode**: Review mode.
    - **members**: Review by member.
    - **roles**: Review by role.
  - **approvers**: User IDs of the approvers when **mode** is set to **members**.
  - **roles**: Role when the mode is set to **roles**.

  - **checkStrategy**: Strategy used when the **mode** is set to **members**.
    - **all**: The review task needs to be reviewed by all members.
    - **any**: The review task can be reviewed by any member.
  - **timeout**: Review timeout.
  - **timeoutStrategy**: Strategy used when the review times out.
    - **reject**: The review is rejected.
    - **pass**: The review is approved.
  - **comment**: Review description.

5.3.9 CreateTag

Create and push tags for code repositories.

Adding the Extension with GUI

Add the **CreateTag** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-16**.

Table 5-16 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Tag Name	Tag name.
Repository	Select the corresponding code repository.

- Click the toggle  to enable **Continue If Tag Exists**. If the created tag exists in the code repository, no error is reported and the pipeline continues to run. Otherwise, the execution fails.

Adding the Extension with YAML

Add the **CreateTag** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **CreateTag**


```
uses: CreateTag
with:
  tagName: v1
```

- **tagName**: tag name.

5.3.10 JenkinsTask

Call Jenkins tasks.

Notes and Constraints

Currently, this function is available in **LA-Mexico City2**, **LA-Sao Paulo1**, and **AP-Singapore**.

Adding the Extension with GUI

Add the **JenkinsTask** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-17**.

Table 5-17 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Jenkins Endpoint	Create a Jenkins service endpoint by referring to **Jenkins**.
jobName	Jenkins job name.
params	Parameters (in JSON format) transferred for starting the job.
Asynchronous	Whether to execute the job asynchronously.
Description	Description for connecting to the endpoint.

Adding the Extension with YAML

Add the **JenkinsTask** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **JenkinsTask**

```
uses: Jenkins
with:
  endpoint: eac965b206e74e2b898a24a4375b6df6
  jobName: job
  params: '{ \"key\":\"value\" }'
```



```
async: true|false
description: description
```

- **endpoint**: ID of the Jenkins endpoint.
- **jobName**: Jenkins job name.
- **params**: Parameters (in JSON format) transferred for starting the job.
- **async**: Whether to execute the job asynchronously.
- **description**: Execution description.

5.3.11 PipelineSuspension

Call the **PipelineSuspension** extension to suspend the current pipeline.

Adding the Extension with GUI

Add the **PipelineSuspension** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-18**.

Table 5-18 Parameter description

Parameter	Description
Name	Extension name, which can be customized. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.

Adding the Extension with YAML

Add the **PipelineSuspension** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **PipelineSuspension**

```
uses: SuspendPipeline
```

5.3.12 ExecuteImage

Call the **ExecuteImage** extension to download public images from SWR to a custom executor and start the images.

Adding the Extension with GUI

Add the **ExecuteImage** extension when you **orchestrate a pipeline**. Set parameters as shown in **Table 5-19**.

Table 5-19 Parameter description

Parameter	Description
Name	Extension name. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Image Attribute	Only public images are supported.
SWR Image Address	Address of the SWR images to be downloaded. To obtain the address: 1. Log in to **SWR**. 2. In the navigation pane, click **My Images**, click the image name to go to the image details page. 3. Click to copy the image download command. The part following **docker pull** is the image path.
Startup Command	Container startup command. Enter Docker commands to run specific applications or scripts in the container.

Adding the Extension with YAML

Add the **ExecuteImage** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **ExecuteImage**

```
uses: official_docker_executor
  with:
    attribute: public
    image: swr.example.com/demo/springboot-helloworld:v1.1
    command: sh start.sh
```

  - **attribute**: Image attribute.
  - **image**: SWR image address.
  - **command**: Container startup command.

5.3.13 ThirdPartyAPI-v2

Call the **ThirdPartyAPI-v2** extension to configure web APIs and call third-party APIs.

Notes and Constraints

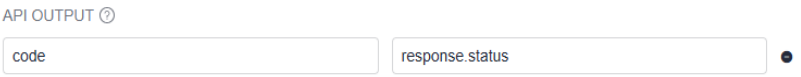
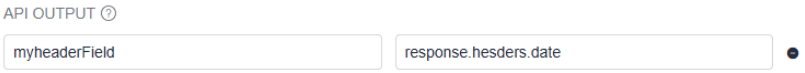
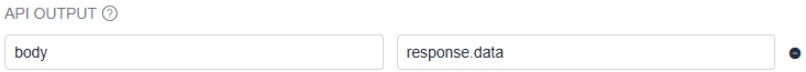
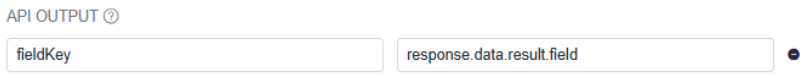
- The **ThirdPartyAPI-v2** extension requires certificate verification. Custom certificate is not supported.
- If the API return code is not **2***XX*, the operation fails.

Adding the Extension with GUI

Add the **ThirdPartyAPI-v2** extension when you **orchestrate a pipeline**. Set parameters as show in **Table 5-20**.

Table 5-20 Parameter description

Parameter	Description
Name	Extension name. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter at least one character.
API Method	Select **GET**, **POST**, **PATCH**, **PUT**, or **DELETE**.
API URL	API that can be called by the public network. Certificate verification is required.
API Header	Configure key-value pairs.
API Parameter	Configure key-value pairs.
API Body	If the request method is **POST**, **PATCH**, or **PUT**, enter the body in JSON format.

Parameter	Description
API Output	The root object name is **response**, which includes multiple outputs. • If the output key is **code**, refer to the following figure to obtain the specific request return code.  • If the returned request header contains the **date** field, refer to the following figure to obtain the **date** value.  • To output the entire body, refer to the following figure.  • If the returned JSON body is as follows and you need to obtain the value of **field**, refer to the following figure. `{ "result": { "field": "value", "field1": "value1" } }` 

Adding the Extension with YAML

Add the **ThirdPartyAPI-v2** extension when you **orchestrate a pipeline**. Configure the YAML syntax as follows:

- **ThirdPartyAPI-v2**

```
uses: official_third_part_invoke
  with:
    API_METHOD: GET
    API_URL: htps://demo
    API_HEADER: '[{"key":"Content-Type","value":"application/json"}]'
    API_PARAM: '[{"key":"name","value":"value"}]'
    API_BODY: "{}"
    API_OUTPUT: '[{"key":"code","value":"response.status"}]'
```

5.4 Configuring Pipeline Parameters

Pipeline Parameter

Parameters are configurable values used during pipeline running and control pipeline process and output. These parameters can be transferred among tasks. By configuring pipeline parameters, you can streamline data of build, deployment, and API test tasks. Parameters include:

- **Predefined Parameters**: They are preconfigured by the system and cannot be configured, deleted, or edited.
- **Custom Parameters**: You can add parameters of string, enumeration, or auto-increment type. You can create a maximum of 100 custom parameters.
- **Parameter Groups**: You can associate all pipelines in the same project with a parameter group. You can create a maximum of 5 parameter groups and add up to 20 custom parameters to each group.

Notes and Constraints

- If a code source alias is set, repository-related system parameters will be generated based on the alias. If no alias is set, the repository name is used as the alias to generate system parameters, for example, *Alias*_**TAG** indicates the repository tag name.
- If a parameter with the same name exists, the parameter priority is as follows: predefined parameters > custom parameters > parameter groups.
- If a pipeline is associated with multiple parameter groups that contain parameters with the same name, the value of the parameter in the last associated parameter group will be used.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Pipeline Parameters

CodeArts Pipeline includes predefined parameters, custom parameters, and parameter groups.

Predefined Parameters

Table 5-21 lists the predefined parameters. The parameter reference format is *${Parameter name}*. Enter **$** in the text box and the predefined parameter list will be displayed.

Table 5-21 Predefined Parameters

Parameter	Description
PROJECT_ID	ID of the project to which the current pipeline belongs.
PIPELINE_ID	ID of the current pipeline.
PIPELINE_NUMBER	Pipeline execution number.

Parameter	Description
COMMIT_ID	The last commit ID before execution.
COMMIT_ID_SHORT	The last short commit ID before execution.
TIMESTAMP	Pipeline execution timestamp. For example, **20211222124301**.
PIPELINE_TRIGGER_TYPE	Pipeline trigger type, which includes Manual, Scheduler, Rollback, and Webhook (CreateTag, Note, Issue, MR, and Push).
PIPELINE_NAME	Pipeline name.
REPO_URL	Code repository address (HTTPS).
EXECUTE_USER	The user who executes the pipeline.
EXECUTE_USER_ID	Executor ID.
EXECUTE_USER_NAME	Executor name.
EXECUTE_USER_NICKNAME	Executor alias.
PASS_CONDITIONS_LINK	Pipeline execution details link.
PIPELINE_RUN_ID	Pipeline execution ID.
MERGE_ID	Merge request ID.
WEBHOOK_PAYLOAD	Webhook request payload information.

Parameter	Description
COMPONENT_CHANGE_IDS	Change ID list. For example, <ID1>, <ID2>,...<IDn>. If a pipeline is triggered based on specific changes, this parameter contains the change IDs that initiated the run. If a pipeline is triggered based on the code repository, it contains the change IDs associated with the microservice's repository.
Repo01_REPOSITORY_NAME	Repository name.
Repo01_SOURCE_BRANCH	The name of a merge request's source branch in the repository used to trigger the pipeline.
Repo01_TARGET_BRANCH	Name of the target branch for repository operations.
Repo01_TAG	Repository tag name.
Repo01_COMMIT_ID	The last commit ID before the repository execution.
Repo01_COMMIT_ID_SHORT	The last short commit ID before execution.
Repo01_REPO_URL	Code repository address (HTTPS).

Configuring Custom Parameters

You can create and configure pipeline parameters. The parameter reference format is *${Parameter name}*. Enter **$** in the text box and the custom parameter list will be displayed.

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the pipeline list page, search for the target pipeline, click ··· in the **Operation** column, and click **Edit**.

Step 3 Switch to the **Parameter Configuration** tab page.

Step 4 On the displayed page, click **Create Now** to configure parameters. If there are custom parameters, click **Create Parameter** to add a new one.

Figure 5-5 Creating a custom parameter

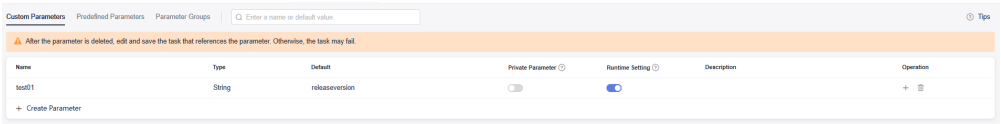
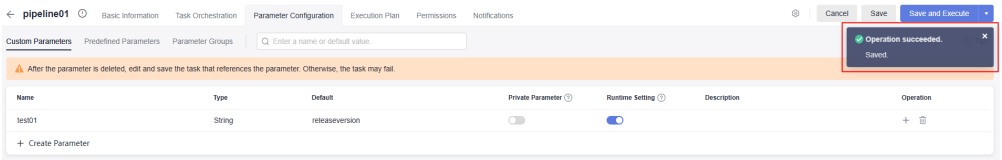


Table 5-22 Custom parameters

Parameter	Description	Example Value
Name	The specified name cannot be the same as that of a predefined parameter. Enter a maximum of 128 characters, including letters, digits, and underscores (_).	Enter **test01**.
Type	Parameter types, including **String** (default), **Auto Increment**, and **Enumeration**.	Select **String**.
Default	Default value of the parameter. • **String**: The value can contain no more than 8,192 characters. It can be left blank. • **Auto-Increment**: The value can contain no more than 8,192 characters. The value cannot be empty. If an auto-increment parameter is referenced in a pipeline, its value (which ends with a digit) is incremented by 1 each time the pipeline runs. • **Enumeration**: Enter a maximum of 8,192 characters, including only letters, digits, hyphens (-), underscores (_), commas (,), periods (.), and slashes (/). After you select **Enumeration**, the **Enumeration** dialog box is displayed for you to set optional values. The value cannot be empty. After that, click the **Default** drop-down list box, select or search for a value.	Enter **releaseversion**.
Private Parameter	Parameters are encrypted for storage and decrypted at runtime. Private parameters are not displayed in run logs and will not be cloned synchronously during pipeline cloning. Only parameters of string can be set to **Private Parameter**.	-
Runtime Setting	If **Runtime Setting** is enabled, you can change the value of the parameter during execution configuration.	Click to enable **Runtime Setting**.
Description	Description of the custom parameter. Enter a maximum of 512 characters.	-

Parameter	Description	Example Value
Operation	Click + in the **Operation** column to add a parameter. Click 🗑 in the **Operation** column to delete a parameter.	-

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.



----End

Configuring a Parameter Group

Step 1 Access the CodeArts Pipeline hompage through a project.

Step 2 Click the **Parameter Groups** tab and then click **Create Group**.

Step 3 On the displayed page, set parameters by referring to **Table 5-23**.

Figure 5-6 Creating a parameter group

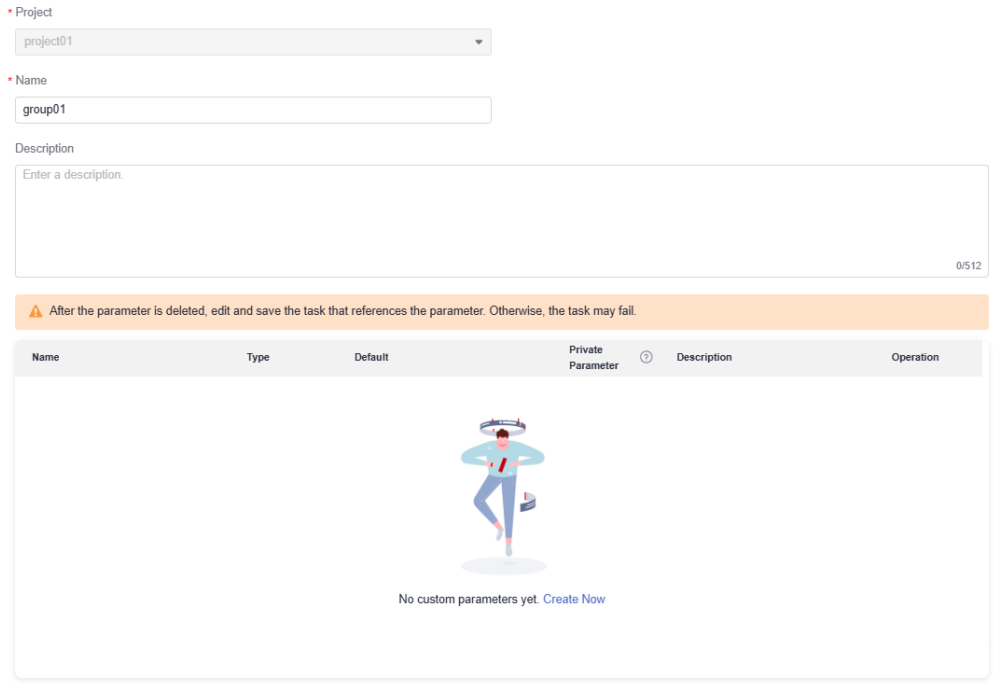


Table 5-23 Parameter group description

Basic Information	Description	Example Value
Project	Project to which the parameter group belongs. The project cannot be changed.	The default value is **project01**.
Name	Enter a maximum of 128 characters, including only letters, digits, and underscores (_).	Enter **group01**.
Description	Description of the parameter group. Enter a maximum of 512 characters.	-
Custom parameter list	Click **Create Now** to create custom parameters. For details, see **Configuring Custom Parameters**.	The following figure shows a new parameter.

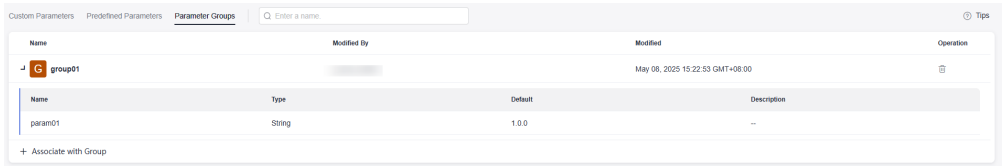
Step 4 Click **OK**.

Step 5 Locate the pipeline to be associated with the parameter group, click ••• in the **Operation** column, and select **Edit**. Click the **Parameter Configuration** tab and click **Parameter Groups**.

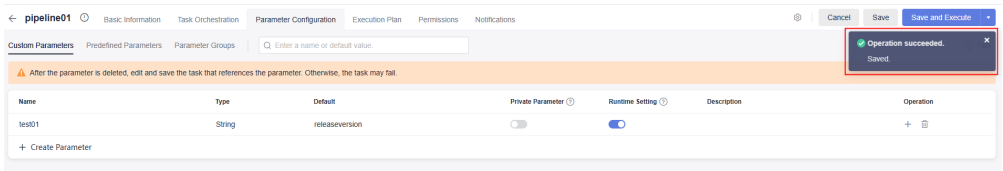
Step 6 Click **Associate Now**, select the desired parameter group, and click **Confirm** to associate the pipeline with the parameter group.

- Expand the group to check parameter details.
- Click in the **Operation** column to diassociate from the parameter group.

Figure 5-7 Associating with a parameter group



Step 7 After the configuration, click **Save**. A message is displayed, indicating that the operation is successful.



----End

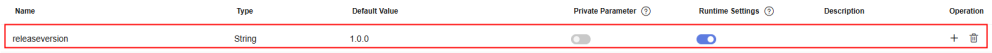
Using a Parameter in a Pipeline

This section describes how to configure the **releaseversion** parameter in a pipeline and transfer the parameter to a build job.

Step 1 Create a build task.

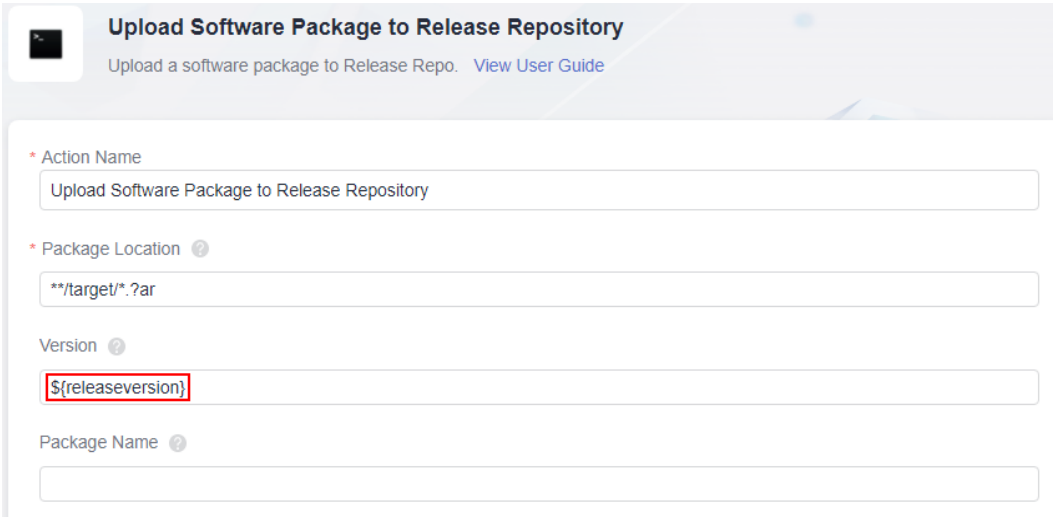
Step 2 Click the **Parameters** tab, add the **releaseversion** parameter, set the default value, and enable **Runtime Settings**.

Figure 5-8 Creating a build task parameter



Step 3 Click the **Build Actions** tab. On the displayed page, click **Add Action**, move the cursor to the **Upload to Release Repo** card, click **Add**, and set **Version** to a reference parameter. After you enter **$** in the text box, a parameter list is displayed. Select the **releaseversion** parameter created in the previous step. **$** will not trigger the display of parameter groups.

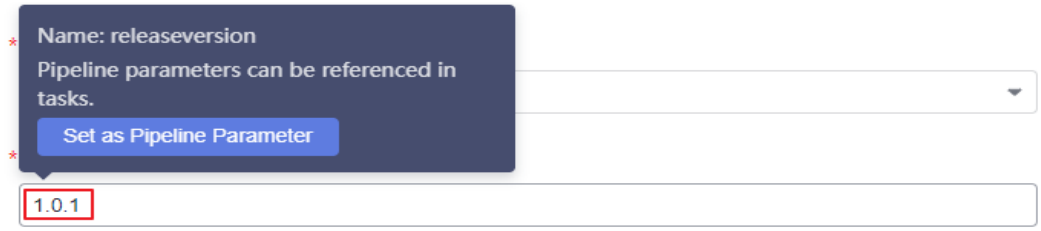
Figure 5-9 Referencing a build task parameter



Step 4 After the configuration, click **Save**.

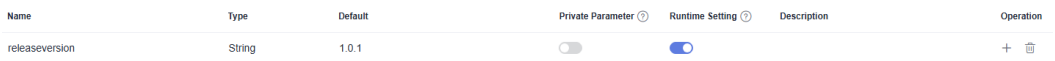
Step 5 **Create a pipeline** using the blank template, add the **Build** extension and select the created build task. The parameter **releaseversion** is displayed.

Figure 5-10 Configuring a build task parameter



Step 6 Move the cursor to the **releaseversion** parameter and click **Set as Pipeline Parameter**. Alternatively, click **OK**, switch to the **Parameter Configuration** tab page, create the pipeline parameter **releaseversion**, set **Type** to **Auto-increment** or **String**, set a default value, and enable **Runtime Setting**.

Figure 5-11 Creating a pipeline parameter



Step 7 Switch back to the **Task Orchestration** tab page, and edit the added build task. Use **$** to reference the **releaseversion** parameter in the build job.

Figure 5-12 Referencing a pipeline parameter

NOTE

- Only text parameters for which **Runtime Settings** is enabled will be displayed.
- You can move the cursor to a parameter name to quickly set the parameter as a pipeline parameter.

Step 8 Save the information and click **Save and Execute**. In the displayed window, check the parameter information.

The parameter value is the default value specified when you added the parameter. You can change the value if needed. If you change it, the new value will be used in the build job.

Step 9 Click **Execute** to execute the pipeline.

----End

5.5 Configuring Pipeline Execution Plans

Pipeline Execution Plan

You can configure event triggers, scheduled tasks, webhooks, parallel execution policies, and preemption policies for pipelines. These policies automate executions, trigger executions through third parties, allocate granular resources, and forcibly stop all running instances triggered by the same event.

Notes and Constraints

- For an event trigger:
  - The branch is matched first, and then the path is matched. If both branch and path matching is successful, the pipeline will be triggered.
  - Path exclusion takes precedence over path inclusion. If any changed files are not excluded, and the included path is not configured, the pipeline will be triggered; if the included path is configured and any of the changed files are included, the pipeline will be triggered.
  - Tag exclusion takes precedence over tag inclusion. If a tag is included and excluded at the same time, the pipeline will not be triggered.
  - Path matching covers the first 300 changed files per commit. To match more than 300 files, split them.
- A maximum of 10 scheduled tasks can be configured for a pipeline.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Configuring Event Triggers

Event triggers include code commit, merge request, and tag creation.

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Execution Plan** page, and then configure event triggers.

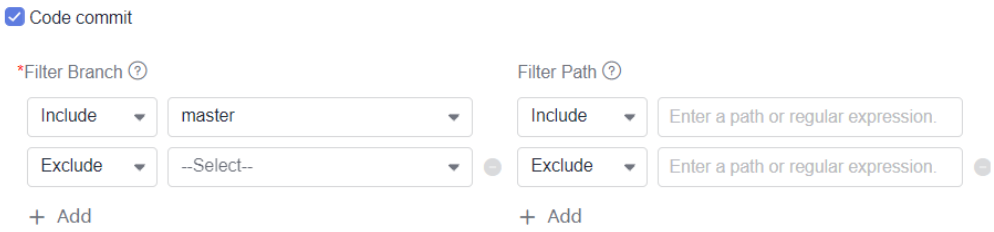
- Triggered upon code commits (in CodeArts Repo)

  You can filter branches and paths by including or excluding specific ones. Target branches and paths will be monitored for code commits.

  – Branch filter: allows you to include or exclude branches.

  – Path filter: allows you to include or exclude paths where changed files locate.

  Figure 5-13 Configuring code commit trigger

- Triggered upon merge requests (in CodeArts Repo)

  You can filter branches and paths by including or excluding specific ones. Target branches and paths will be monitored for merge request events.

  Events:

  – **Create**: triggered upon MR creation.

  – **Merge**: triggered when an MR is merged. A code commit event will also be triggered.

  – **Reopen**: triggered upon MR reopening.

  – **Update**: triggered upon MR content, setting, or source code update. If you also enable **Code update**, the pipeline will be triggered only upon source code update.

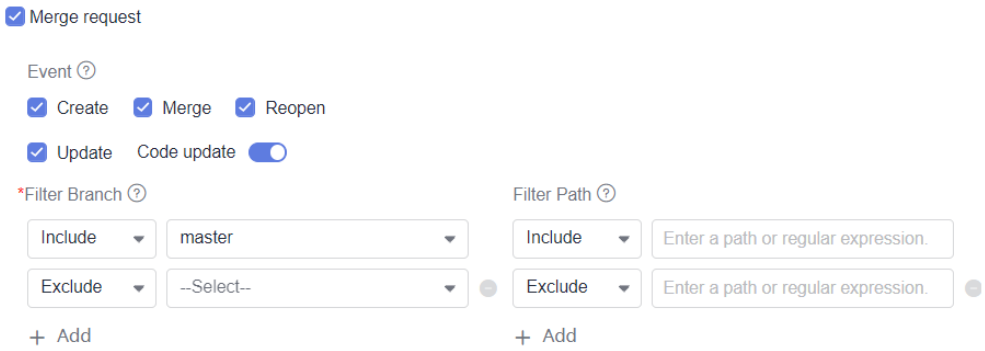
  – **Code update**: triggered upon source code update.

  Branch description:

  – Branch filter: allows you to include or exclude branches.

  – Path filter: allows you to include or exclude paths where changed files locate.

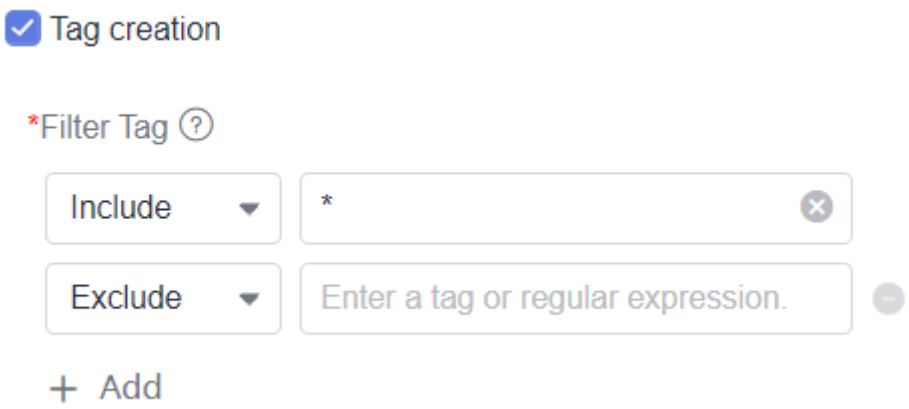
Figure 5-14 Configuring merge request trigger



- Triggered upon tag creation (in CodeArts Repo)

  You can filter tags by including or excluding specific ones. The associated code repository will be monitored for tag creation.

Figure 5-15 Configuring tag creation trigger



Step 4 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Scheduled Tasks

Set scheduled tasks to execute a pipeline at a specified time.

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Execution Plan** page.

Step 4 Click **Create Now** to create a scheduled task. Turn on the **Enable** toggle, and set the execution time.

Figure 5-16 Configuring a scheduled task

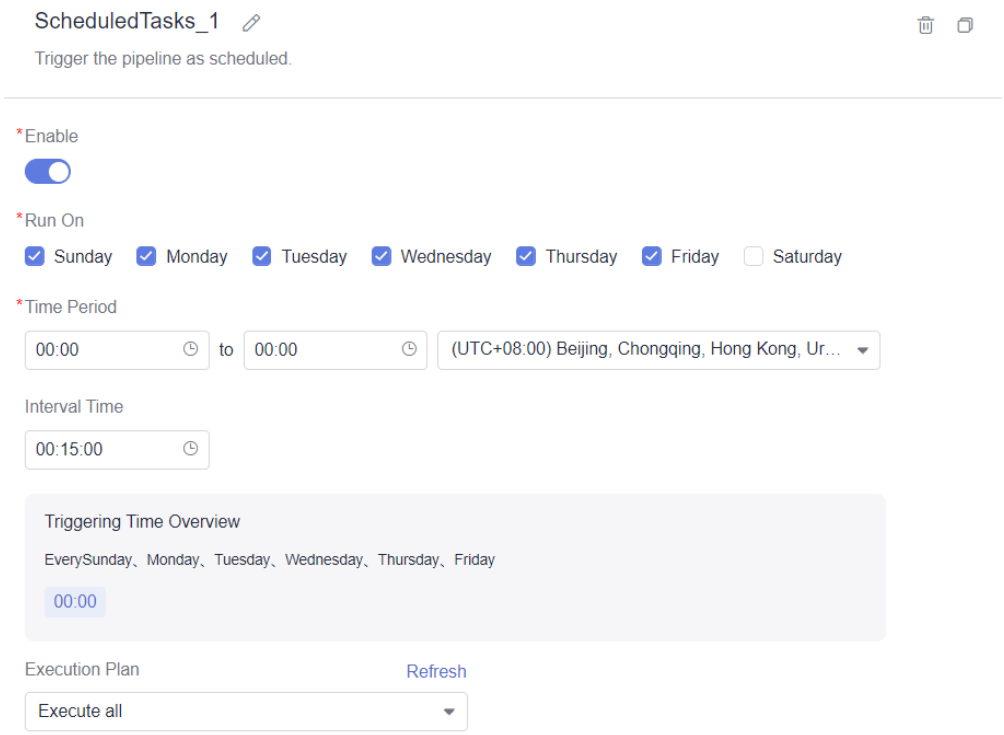


Table 5-24 Scheduled task

Parameter	Description
Enable	Whether to enable the scheduled task.
Run On	Select the execution date.
Time Period	Select the execution period and time zone.
Time Interval	Set the interval for triggering the pipeline.
Execution Plan	Select a plan to be executed at a scheduled time. You can also create an execution plan. For details, see **Configuring an Execution Plan**.

NOTE

To delete a scheduled task, click in the upper right corner. To clone a scheduled task, click in the upper right corner.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Webhooks

You can configure webhooks to automatically trigger a pipeline through a third-party system.

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Execution Plan** page.

Step 4 Enable **Webhook** and set parameters as shown in **Table 5-25**.

Figure 5-17 Configuring a webhook trigger

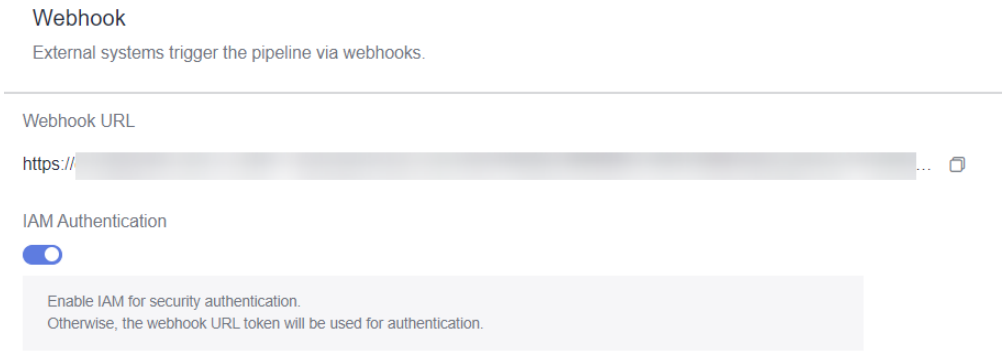


Table 5-25 Webhook parameters

Parameter	Description
Webhook URL	Copy the address to the third-party system trigger and use the POST method to call the address to run the pipeline.
IAM Authentication	• If you need to enable IAM authentication, add the user IAM token to the API request header. The following is a calling example: curl --header "Content-Type: application/json" --header 'x-auth-token: XXXX (IAM Token)' --request POST --data "{}" *Webhook trigger source* • If you do not need to enable IAM authentication. The following is a calling example: curl --header "Content-Type: application/json" --request POST --data "{}" *Webhook trigger source*

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Parallel Execution

By default, five parallel executions are allowed in a pipeline. Excess instances will not be executed. Alternatively, you can change the maximum number of parallel instances (running and paused).

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Execution Plan** page.

Step 4 Enable **Parallel Execution**, set the max parallel instances and execution policy for extras.

Figure 5-18 Configuring parallel execution

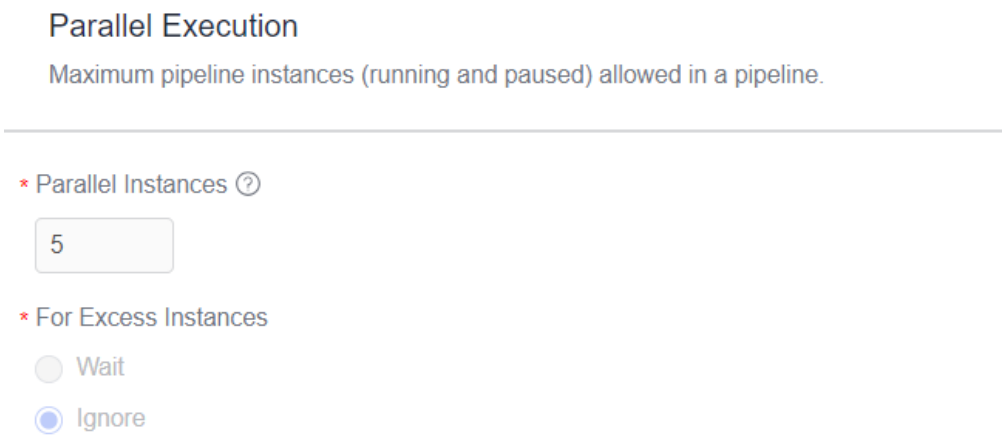


Table 5-26 Parallel execution parameters

Parameter	Description
Parallel Instances	Maximum parallel instances, which vary by your purchases and packages. For details, see CodeArts Pipeline specifications.
For Excess Instances	You can choose: • **Wait**: Excess instances will wait for execution. You can check the queuing instances on the pipeline details page. – Max. 100 queuing instances per pipeline. – Instances will not be executed after 24 hours of waiting. – You can manually cancel the waiting. – Instance configurations are fixed once they enter the queue. • **Ignore**: Excess instances will not be executed.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Preemption

If there are several running instances triggered by the same event, the most recent instance will proceed and others will stop.

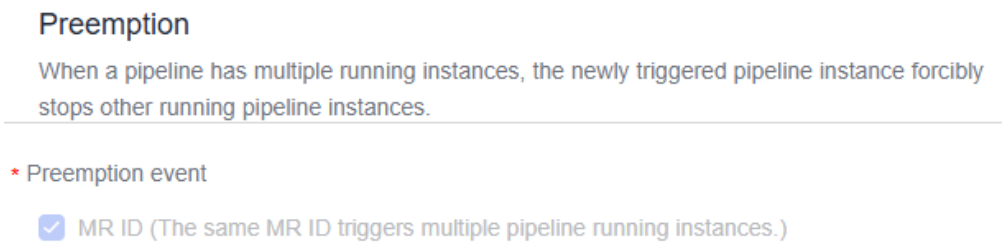
Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ··· in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Execution Plan** page.

Step 4 Enable **Preemption** and select a preemption event.

Figure 5-19 Configuring a preemption policy



- Currently, this feature only applies to CodeArts Repo repositories.
- Only the **MR ID** event is available, that is, running instances triggered by the same MR will be preempted.
- After **Preemption** is enabled, all running instances that meet the preemption condition will be stopped.
- If there are excess instances, and no instances meet the preemption condition, the excess instances will either wait or not be executed, depending on the policy.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

5.6 Configuring Pipeline Permissions

Pipeline Permissions

Pipeline permissions control what users or roles can do with pipelines and their resources. They help keep pipelines secure, stable, and compliant by managing who can perform specific actions.

This section describes how to configure permissions for a single pipeline by role or user.

Notes and Constraints

- By default, role permissions of a pipeline are the same as those of the project that the pipeline belongs to.
- The permissions of the project administrator and pipeline creator cannot be changed.
- By default, user permissions automatically synchronize with role permissions. If user permissions are changed, the new user permissions overwrite role permissions.
- By default, a user with permissions to edit or execute pipelines can also view pipelines.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Configuring Pipeline Permissions

Step 1 **Access the CodeArts Pipeline homepage**.

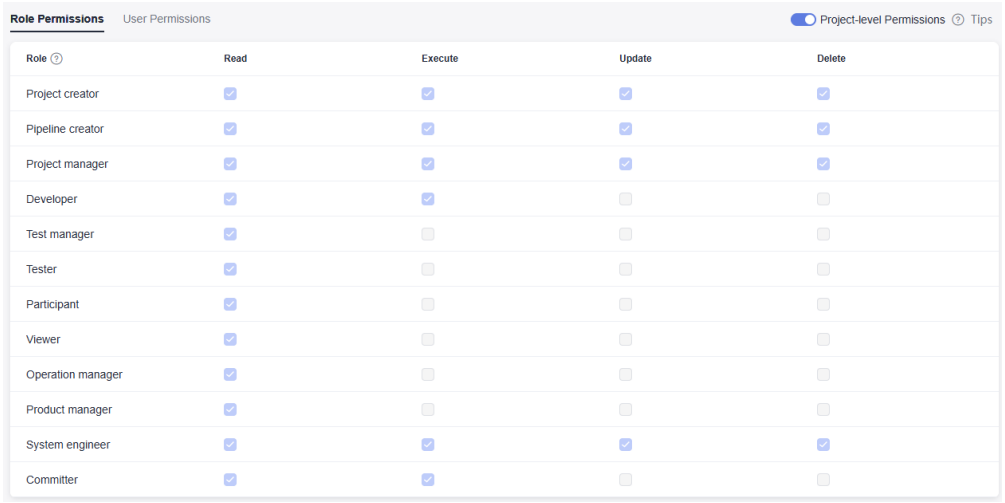
Step 2 On the pipeline list page, search for the target pipeline, click ··· in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Permissions** page, disable **Project-level Permissions**, and then configure role and user permissions for the pipeline. If **Project-level Permissions** is enabled, the pipeline inherits the project permissions.

- Configure role permissions

  You can select or deselect permissions to specify whether a role has permissions to view, execute, edit, and delete the pipeline.

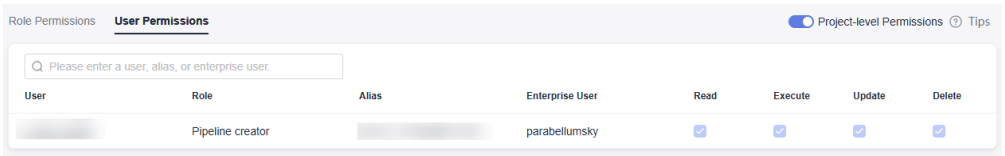
Figure 5-20 Configure role permissions



- Configure user permissions

  You can select or deselect permissions to specify whether a user has permissions to view, execute, edit, and delete the pipeline.

Figure 5-21 Configure user permissions



----End

5.7 Configuring Pipeline Notifications

Pipeline Notification

Pipeline notifications inform users or systems about pipeline activities. Subscribing to specific events helps you stay updated on pipeline statuses and execution results. You can configure event notifications (pop-ups, emails, Feishu, WeCom, and DingTalk notifications) for a pipeline.

Constraints

- By default, only pop-up notifications will be sent.
- WeCom, DingTalk, and Feishu notifications are available only in the **AP-Singapore** region.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Configuring Notifications from CodeArts Pipeline

You can configure pop-ups to inform creators, executors, and users who have favorited pipelines of pipeline events (deletion, failure, success, and updates).

Step 1 Access the CodeArts Pipeline homepage.

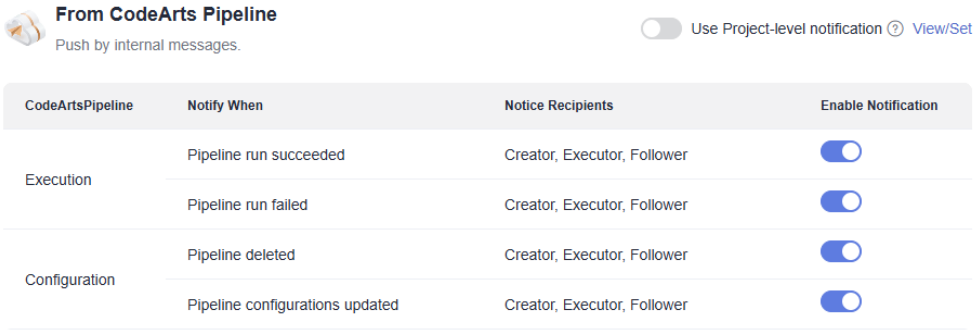
Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Notifications** page.

Step 4 Enable **From CodeArts Pipeline**, and enable or disable notifications as needed.

- You can click  in the upper right corner of the CodeArts Pipeline homepage and check the notification messages in the **Messages** window.

Figure 5-22 Configuring notifications from CodeArts Pipeline



Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Email Notifications

You can configure email notifications to inform creators, executors, and users who favorite pipelines of pipeline activities (deletion, failure, success, and updates).

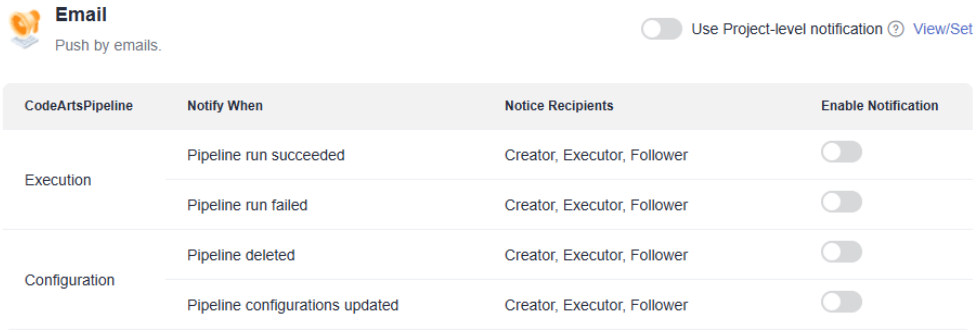
Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ••• in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Notifications** page.

Step 4 Enable **Email**, and enable or disable notifications as needed.

Figure 5-23 Configuring email notifications



Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring WeCom Notifications

Push pipeline activities to WeCom.

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the pipeline list page, search for the target pipeline, click ... in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Notifications** page.

Step 4 Enable **WeCom** and set parameters.

Figure 5-24 Configuring WeCom notifications

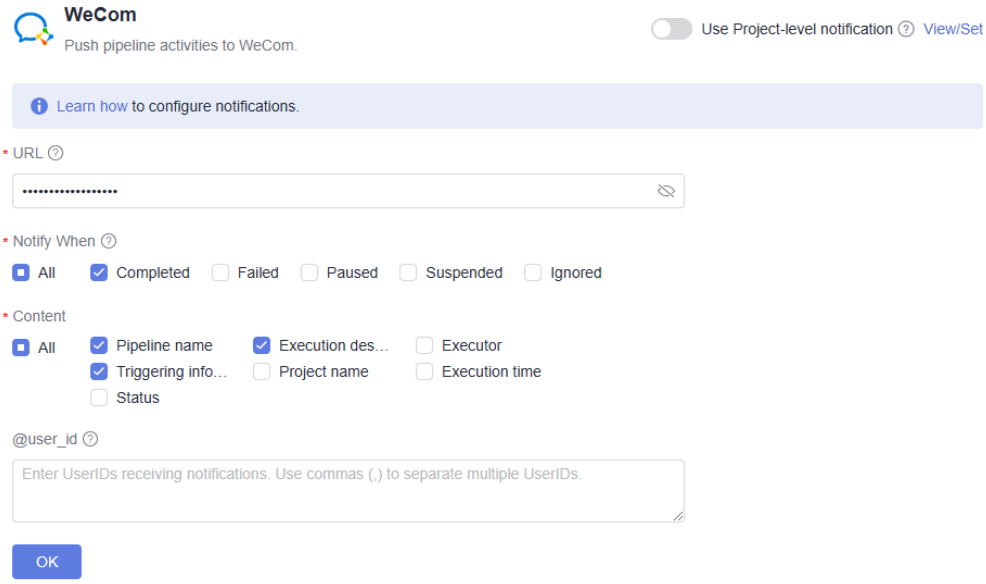


Table 5-27 Parameters of WeCom notifications

Parameter	Description
URL	Enter the URL of the WeCom robot notifications. The URL must start with **http://** or **https://**, for example, **https://***xxxxx*.
Notify When	Select the pipeline status to be notified: **completed**, **failed**, **paused**, **suspended**, and **ignored**.
Content	Select the content to be notified: **Pipeline name**, **Execution description**, **Executor**, **Triggering information**, **Project name**, **Execution time**, and **Status**.
@user_id	Enter the user IDs receiving notifications. Use commas (,) to separate multiple user IDs.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring DingTalk Notifications

Push pipeline activities to DingTalk.

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the pipeline list page, search for the target pipeline, click ... in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Notifications** page.

Step 4 Enable **DingTalk** and set parameters.

Figure 5-25 Configuring DingTalk notifications

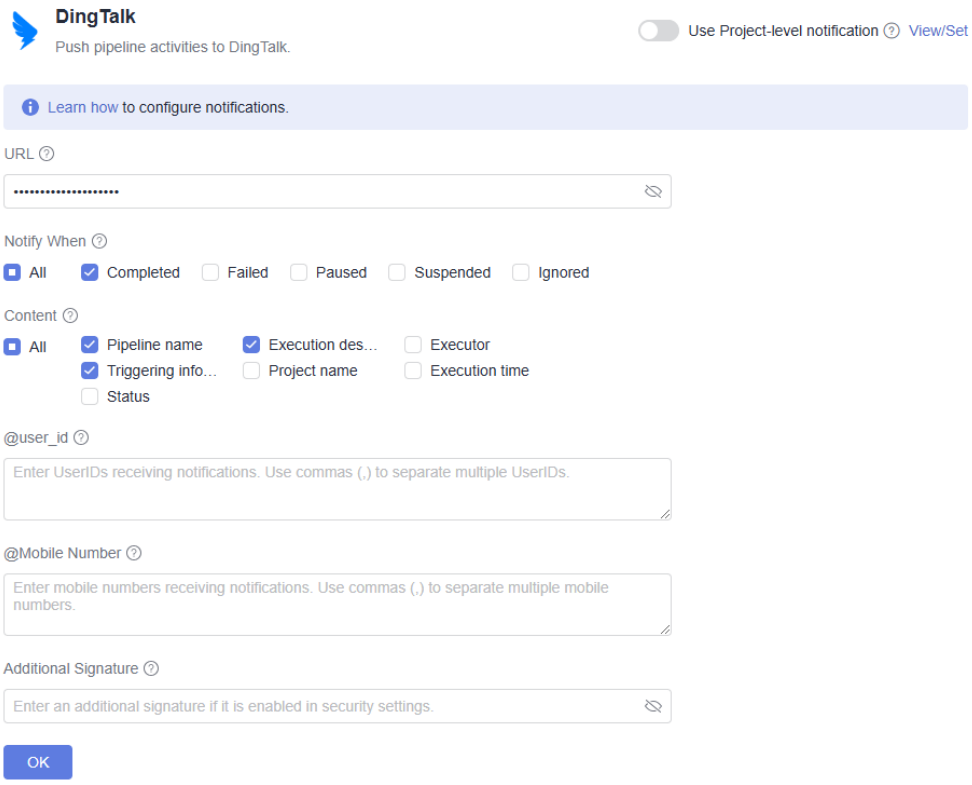


Table 5-28 Parameters of DingTalk notifications

Parameter	Description
URL	Enter the URL of the DingTalk robot notifications. The URL must start with **http://** or **https://**, for example, **https://***xxxxx*.
Notify When	Select the pipeline status to be notified: **completed**, **failed**, **paused**, **suspended**, and **ignored**.
Content	Select the content to be notified: **Pipeline name**, **Execution description**, **Executor**, **Triggering information**, **Project name**, **Execution time**, and **Status**.
@user_id	Enter the user IDs receiving notifications. Use commas (,) to separate multiple user IDs.
@Mobile Number	Enter the mobile numbers receiving notifications. Use commas (,) to separate multiple numbers.
Additional Signature	(Optional) Enter this parameter only when you have enabled security settings for the DingTalk custom robot.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Configuring Feishu Notifications

Push pipeline activities to Feishu.

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, search for the target pipeline, click ... in the **Operation** column, and select **Edit**.

Step 3 Switch to the **Notifications** page.

Step 4 Enable **Feishu** and set parameters.

Figure 5-26 Configuring Feishu notifications

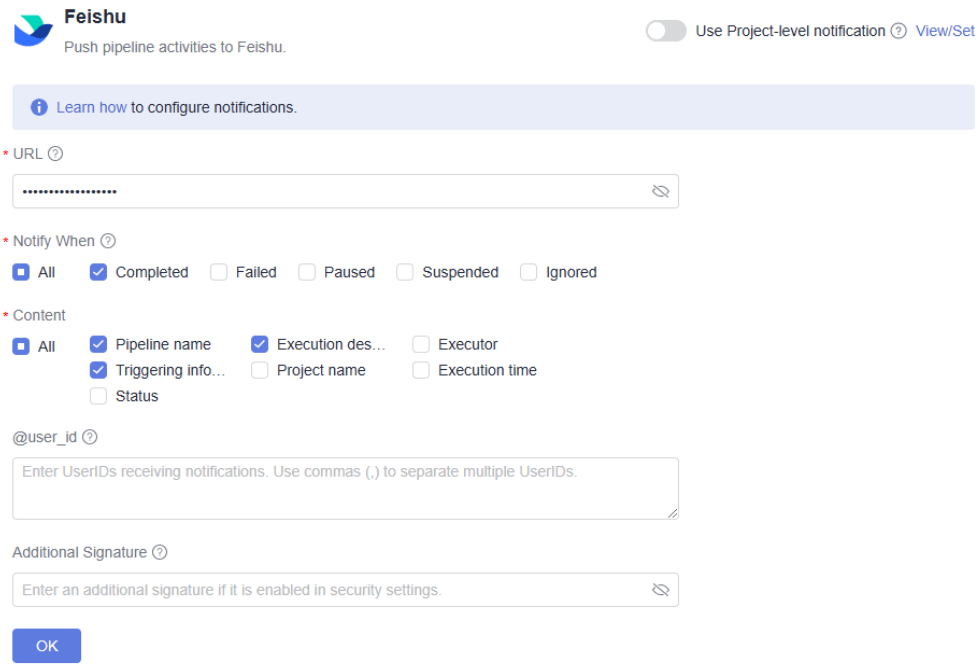


Table 5-29 Parameters of Feishu notifications

Parameter	Description
URL	Enter the URL of the Feishu robot notifications. The URL must start with **http://** or **https://**, for example, **https://***xxxxx*.
Notify When	Select the pipeline status to be notified: **completed**, **failed**, **paused**, **suspended**, and **ignored**.

Parameter	Description
Content	Select the content to be notified: **Pipeline name**, **Execution description**, **Executor**, **Triggering information**, **Project name**, **Execution time**, and **Status**.
@user_id	Enter the user IDs receiving notifications. Use commas (,) to separate multiple user IDs.
Additional Signature	(Optional) Enter this parameter only when you have enabled security settings for the Feishu custom robot.

Step 5 After the configuration, click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

6 Grouping Pipelines

Group Management

A project usually involves multiple pipelines. You can group them to improve efficiency, for example, by environment level (production and testing), or by R&D stage (scheduled build, development self-test, integration test, and production deployment).

Notes and Constraints

- On the **Permissions** tab page, only the project admin and project manager can modify pipeline permissions in batches.

Prerequisites

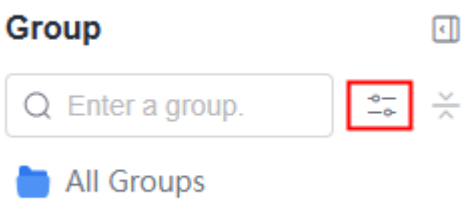
- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Grouping Pipelines

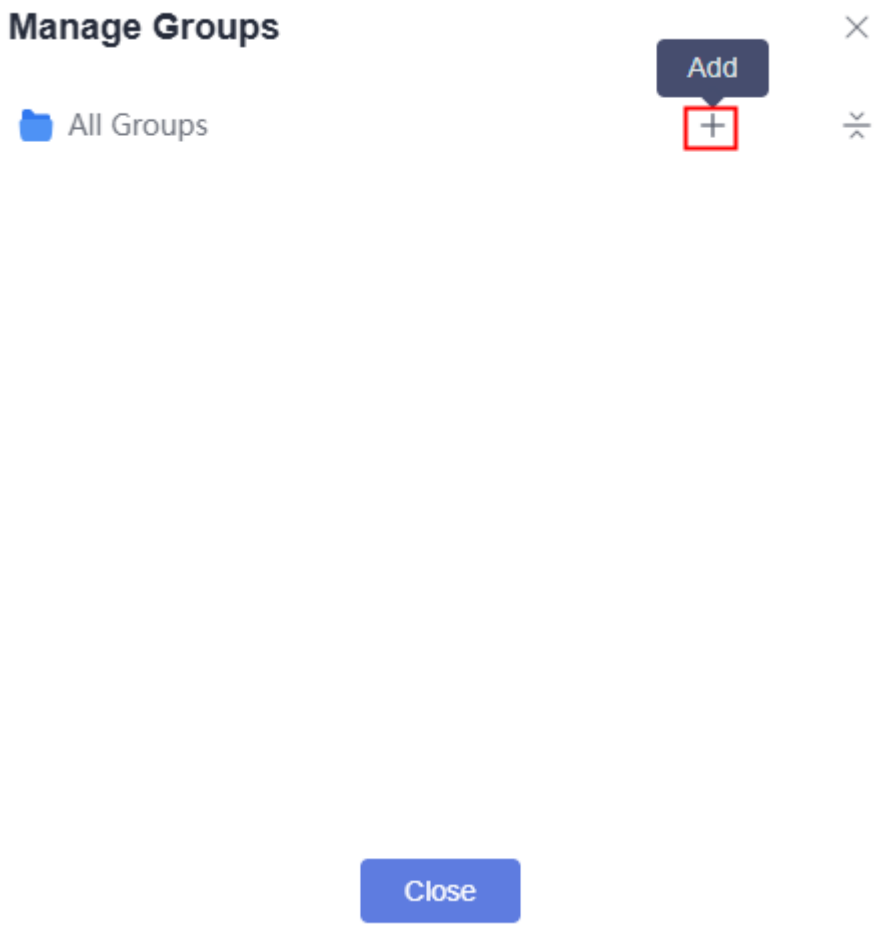
Step 1 **Access the CodeArts Pipeline homepage** through a project.

Step 2 Click **All Groups** to expand the pipeline group panel.

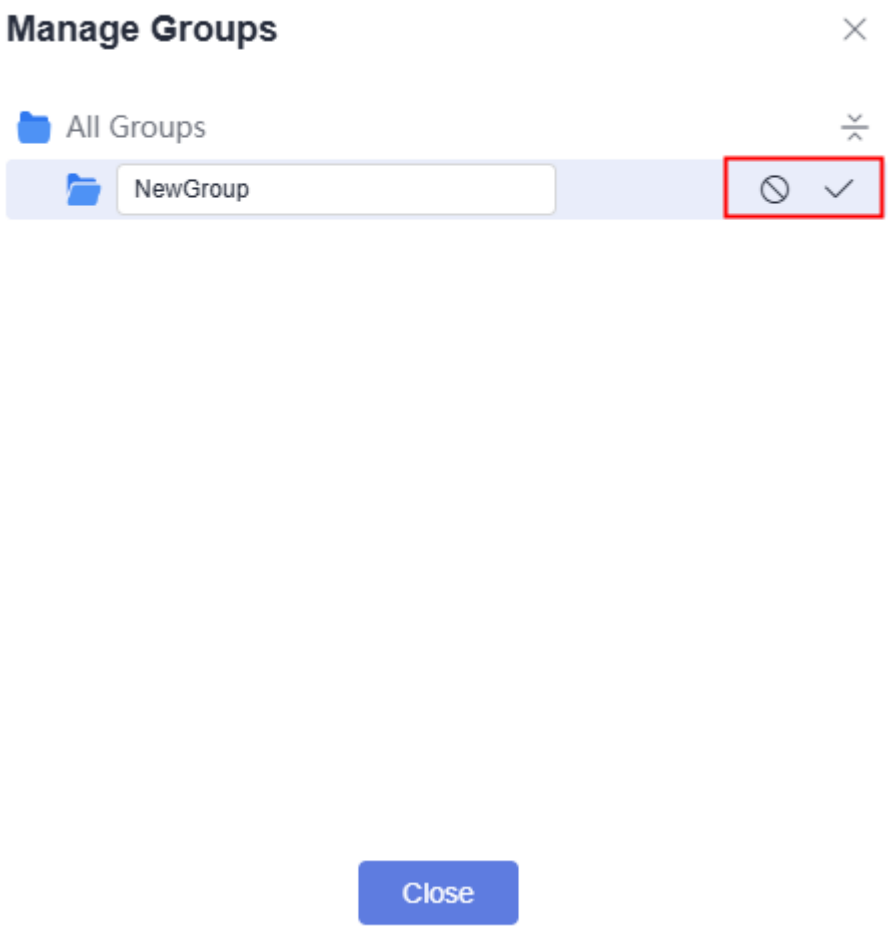
Step 3 Click , the **Manage Groups** window is displayed. Move the cursor to the row where **All Groups** is located and click to add a group.



Step 4 Move the cursor to the row where **All Groups** is located and click + to add a group.



Step 5 Specify a group name. Click ✓ to confirm group creation or click ⊘ to cancel group creation.



After a group is created, you can perform the following operations:

- Click + in the row where the group is located to create a subgroup. You can create a maximum of three levels of subgroups.
- Click in the row where the group is located to change the group name.
- Click in the row where the group is located to move or delete the group.

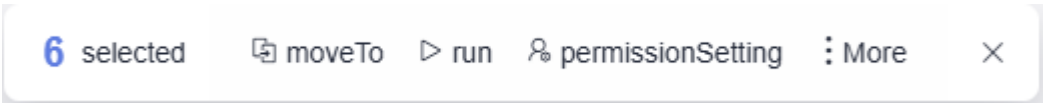
NOTE

After the first group is created, **Ungrouped** is automatically generated for ungrouped pipelines.

Step 6 Click **Close** to return to the pipeline list page after all groups are created.

Step 7 Select desired pipelines and perform the following batch operations.

Figure 6-1 Operations on multiple pipelines



- Click **Move To**. The **Move Group** window is displayed. Select a group and click **Confirm**.
- Click **Execute**. In the displayed window, click **OK** to execute pipelines.
- Click **Permissions**. In the displayed window, the project admin and project manager can configure permissions for selected pipelines. Enable **Project-level Permissions** to inherit the project-level permissions. Disable **Project-level Permissions**, and then configure role and user permissions for selected pipelines.
- Choose **More** > **Set Tag**. In the displayed window, set tags for selected pipelines.
- Choose **More** > **Delete**. In the displayed window, enter the prompt information and click **OK**. A maximum of 20 pipelines can be deleted at a time.

----End

7 Executing a Pipeline

You can check the pipeline execution progress, logs, and results in real time.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Executing a Pipeline

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 Search for the target pipeline, click ▷ in the **Operation** column, and select **Manual Execution**.

Step 3 In the displayed window, set the following parameters:

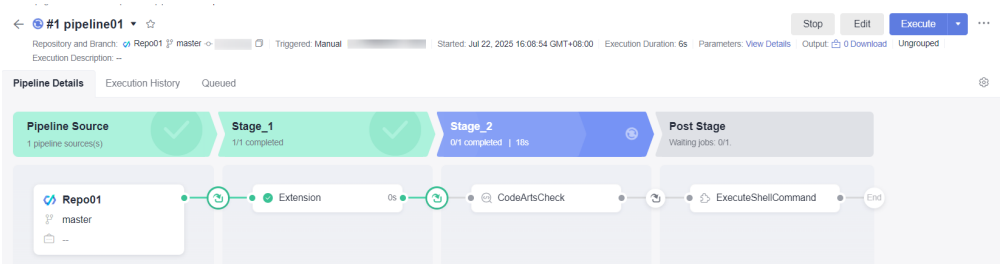
- **Pipeline Source**: Select the branch or label of the code source.
- **Runtime Parameters**: (Optional) Set runtime parameters. For details, see **Configuring Pipeline Parameters**.
- **Execution Stages**: Select one or more jobs to execute. By default, all jobs are executed.

  NOTE

  If **Always Run** is set to **Yes** for a stage, jobs in this stage will be selected for execution by default and cannot be canceled.
- **Description**: Describe the execution.

Step 4 Click **Execute**. On the pipeline details page, you can check the execution progress and job status in real time.

Figure 7-1 Executing a Pipeline



- Click **Stop** in the upper right corner to stop the execution.
- Click **Edit** to change the pipeline configurations.
- A pipeline can have multiple concurrent executions. You can click **Execute** to start a new execution of the pipeline. The maximum number of concurrent pipeline executions varies based on your purchase. For details, see **CodeArts Pipeline specifications**.

Step 5 After the execution is complete, you can check the execution result. If you encounter any problem during the execution, see **Troubleshooting**.

----End

8 Checking a Pipeline

You can check the pipeline list, pipeline execution history, execution details, and queuing status.

Notes and Constraints

- By default, only project managers, project creators, pipeline creators, and system engineers can delete pipelines. You can configure permissions for different roles.
- A tenant can create a maximum of 100 tags.
- Execution records are generated only after the first execution.
- A running pipeline cannot be disabled or deleted.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a project member. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Checking a Pipeline

Step 1 **Access the CodeArts Pipeline homepage**.

The pipeline list page displays all pipelines of the current user. The information is listed in the following table.

Table 8-1 Pipeline information

Parameter	Description
Name	Pipeline name and the project to which the pipeline belongs. **NOTE** If you access CodeArts Pipeline through a project, the project name will not be displayed here.
Last Executed	Information about the last execution of a pipeline, including the execution mode, branch, latest code commit ID, and executor.

Parameter	Description
Tag	Pipeline tag.
Workflow	Scheduling process and execution status (completed, failed, running, or stopped) of the pipeline.
Started & Lasted	Start time and duration of the last execution.
Execution Time	Start time of the last execution.
Execution Duration	Duration of the pipeline execution.
Operation	• Click ▷ to execute a pipeline. • Click ☆ to favorite a pipeline. After the pipeline is favorited, the icon changes to ★. You can click the icon again to unfavorite the pipeline. **NOTE** After you favorite a pipeline, the pipeline will be displayed on top of the pipeline list when you access the page again. Favorited pipelines are sorted in descending order based on their last execution time. If they have not been executed, they are sorted in descending order based on their creation time. • Click ••• and select **Edit** to edit a pipeline. • Click ••• and select **Clone** to quickly create a pipeline based on the current pipeline. • Click ••• and select **Preview** to preview a pipeline. • Click ••• and select **Operation History** to view historical operation records (creation, editing, and failure). • Click ••• and select **Set Tag**. In the displayed dialog box, perform the following operations. – Click + to add tags for the pipeline. Max. five tags for each pipeline. – Click **Manage Tags** to create, edit, or delete tags. • Click ••• and select **Disable** to disable a pipeline. • Click ••• and select **Delete** to delete a pipeline. • Click ••• and select **Execution Plan** to **configure a plan**.

- By default, all users can view the pipeline list.
- Click the drop-down list box of **All Pipelines** to filter pipelines by **All pipelines**, **My created pipelines**, or **My executed pipelines**.
- You can search for a pipeline by its name.
- Click **Set** in the upper right corner to customize the pipeline display information.

Step 2 Click a pipeline name. The **Execution History** page is displayed, showing the execution records.

Table 8-2 Execution history

Parameter	Description
Execution Message	The execution sequence number, execution branch, latest commit information, and latest commit ID of the branch.
Status	Pipeline execution status, including completed, running, failed, stopped, paused, suspended, and ignored.
Execution Type	Pipeline triggers, including manually, code commit, merge request, tag creation, scheduled task, webhook, and sub-pipeline.
Workflow	Scheduling process and execution status (completed, failed, running, or stopped) of the pipeline.
Execution Description	Description of the pipeline execution.
Job Details	Pipeline execution details.
Execution Time	Time when the pipeline starts to be executed.
Execution Duration	Duration of the pipeline execution.
Operation	Pipeline execution parameters. You can click  to view the details.

- You can click the time filter to filter execution records by time. By default, executions in the past 31 days are displayed. You can also check executions in the past 7 days, 14 days, or 90 days.
- Click **Set** in the upper right corner to customize the pipeline execution history information.

Step 3 Click the execution ID to go to the **Pipeline Details** page and check the execution details.

Table 8-3 Operations on the pipeline details page

Operation	Description
Retry	If the execution fails, you can click **Retry** in the upper right corner to resume the execution.
Edit	You can click **Edit** to orchestrate the pipeline.
Execute	You can click **Execute** to execute the pipeline with the latest configurations. An execution record will be generated.

Operation	Description
Download	You can click **Download** next to **Output** to download the build packages. ● Build packages are available only for build jobs. ● If there are multiple build packages, click **Download All**. ● Only the latest 10 build packages are displayed. To download other build packages, go to the **Release Repos** page.
View logs	Click a job card to check its logs and result. ● No log will be generated for **DelayedExecution** and **PipelineSuspension** jobs. ● You can click the failed job card to check the failure reason.
More operations	Click ... in the upper right corner of the page to clone, preview, disable, and delete the pipeline, and check the operation history and execution plan.

Step 4 Click the **Queued** tab.

This page displays the instances to be executed.

- Max. 100 queuing instances per pipeline.
- Instances will not be executed after 24 hours of waiting.
- Click in the **Operation** column to cancel the queuing.
- Instance configurations are fixed once they enter the queue.

----End

9 Checking the Dashboard

The pipeline statistics dashboard displays pipelines, their execution statuses, parallel executions in the system and in each project.

Prerequisites

You have **enabled and authorized CodeArts Pipeline**.

Checking the Dashboard

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 The dashboard displays pipeline statistics.

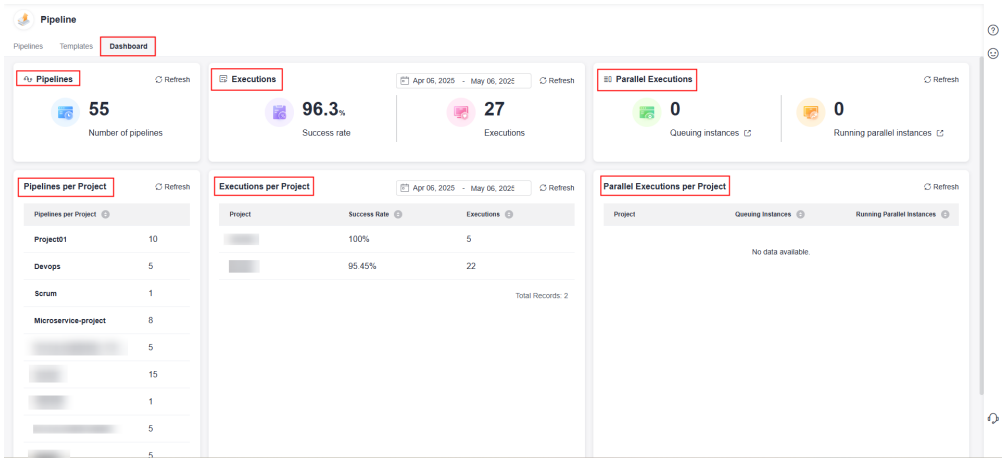


Table 9-1 Pipeline statistics

Item	Description
Pipelines	Total number of pipelines.
Executions	Execution success rate and execution times. You can check executions within 90 days.

Item	Description
Parallel Executions	Number of queuing and running parallel instances. • Click on the right of the queuing instances. The **Queuing Instances** window is displayed for you to check the queuing pipelines and triggers. • Click on the right of the running parallel instances. The **Running Instances** window is displayed for you to check the names, execution information, triggers, and execution time of running pipelines.
Pipelines per Project	Number of pipelines in each project. You can click a project name to access pipelines in this project.
Executions per Project	Execution success rate and execution times in each project. You can check executions during a specified period. You can also click a project name to access pipelines in this project.
Parallel Executions per Project	Number of queuing and running parallel instances in each project.

----End

10 Configuring a Change-triggered Pipeline

Microservice

Microservices are a software governance architecture. A complex software project consists of one or more microservices. Microservices in a system are loosely coupled. Each microservice is independently developed, verified, deployed, and released. Changes can be used to meet requirements and fix vulnerabilities. A change belongs to only one microservice. In microservices, you can create change-triggered pipelines to associate them with change resources and release changes for quick project delivery.

Microservices have the following benefits:

- Specialized: Each microservice focuses on a specific function. It is relatively easy to develop and maintain a single microservice.
- Independently deployable: A microservice is independently deployed and updated without affecting the whole system.
- Tech-diverse: For microservices architectures, different services communicate over RESTful APIs. You can choose the desired technology for each service.

A change-triggered pipeline has the following features:

- A microservice can have only one change-triggered pipeline.
- An integration branch is automatically created during the execution of the change-triggered pipeline. After successful execution, the branch content is merged to the master branch.
- After successful execution, the change status is automatically updated.
- Only one pipeline instance can run at one time.
- A change-triggered pipeline cannot be triggered by an event or at a specified time.

Notes and Constraints

- When you create a microservice, if you set **Pipeline Source** to **None**, after the microservice is created, you can click its name to associate it with a pipeline source on the **Overview** page.

- A repository can be associated with only one microservice.
- After creating a microservice, when you change its code repository, if there are unclosed changes or running pipelines in the microservice, the **Data Processing** window will be displayed. In that case, close all changes and stop all running pipelines.

Prerequisites

- You have **enabled and authorized CodeArts Pipeline**.
- Your role is a **project administrator** or **pipeline creator**. For details about how to configure permissions, see **Authorizing CodeArts Pipeline**.

Creating a Microservice

Step 1 **Access the CodeArts Pipeline homepage** through a project.

Step 2 Click the **Microservices** tab.

Step 3 Click **Create Microservice**. On the displayed page, configure parameters.

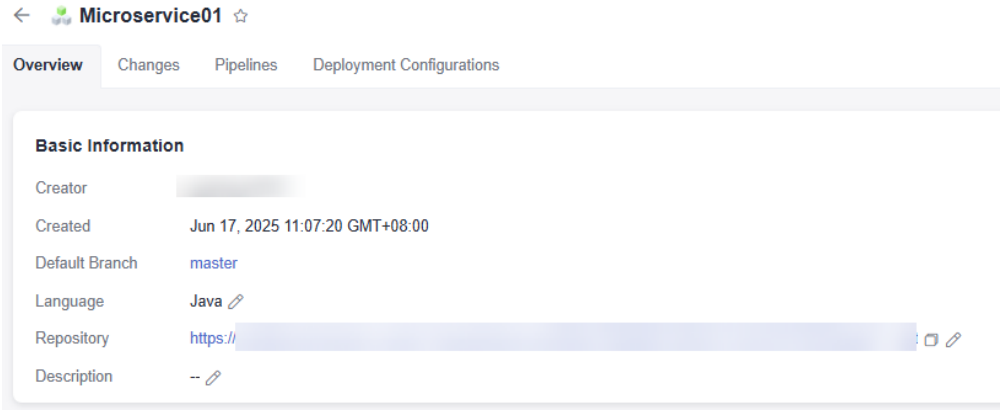
Table 10-1 Microservice parameters

Parameter	Description	Example Value
Project	Project to which the microservice belongs. The project cannot be changed.	The default value is **project01**.
Microservice Name	The name can contain a maximum of 128 characters, including letters, digits, and underscores (_).	Enter **Microservice01**.
Pipeline Source	Only **Repo** is supported.	Select **Repo**.
Repository	Code repository associated with the microservice. Select a created code repository.	Select **Repo01**.
Default Branch	Default branch associated with a microservice. This branch will be used when a change-triggered pipeline is executed. After the change-triggered pipeline is executed, all changed feature branches will be merged into the default branch.	Select **master**.
Language	The development language of the microservice. Available languages: **Java**, **Python**, **Node.js**, **Go**, **.Net**, **C++**, and **PHP**.	Select **Java**.
Description	Description of the microservice. Enter a maximum of 1,024 characters.	-

Step 4 Click **OK**. A message is displayed, indicating that the microservice is created successfully. The **Overview** page is displayed.

Information such as the creator, creation time, and code source is displayed. You can edit the language, repository, and description.

Figure 10-1 Microservice overview page



Step 5 Return to the microservice list to review the created microservice, as shown in the following table.

Figure 10-2 Microservice list

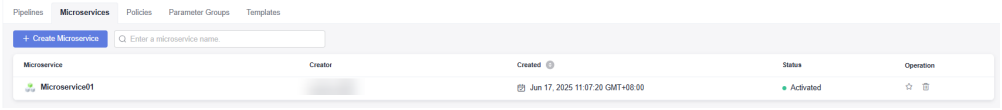


Table 10-2 Microservice list

Item	Description
Microservice	Microservice name.
Creator	Name of the user who created the microservice.
Created	Time when the microservice was created. You can move the cursor to the **Created** column and click to sort microservices by creation time.
Status	Status of a microservice. After a microservice was created, it is in an activated status.
Operation	Click to favorite a microservice. After the microservice is favorited, the icon changes to . You can click the icon again to unfavorite the microservice. Also, you can click to delete the microservice. If a microservice is deleted, all changes and pipelines in the microservice will be deleted.

- The microservice list displays all microservices of the project.
- You can enter a microservice name in the search box to search for it.

----End

Creating a Change

You can manage changes in the microservice.

Preparations

You have **created a work item**.

Procedure

Step 1 **Access the CodeArts Pipeline homepage** through a project.

Step 2 Click the **Microservices** tab.

Step 3 Click a microservice name. The **Overview** page is displayed.

Step 4 Click the **Changes** tab.

All changes are displayed. Click **All Changes** and select **My Changes** to filter changes created by the login user.

Step 5 Click **Create Change**. On the displayed page, set parameters.

Figure 10-3 Change parameters

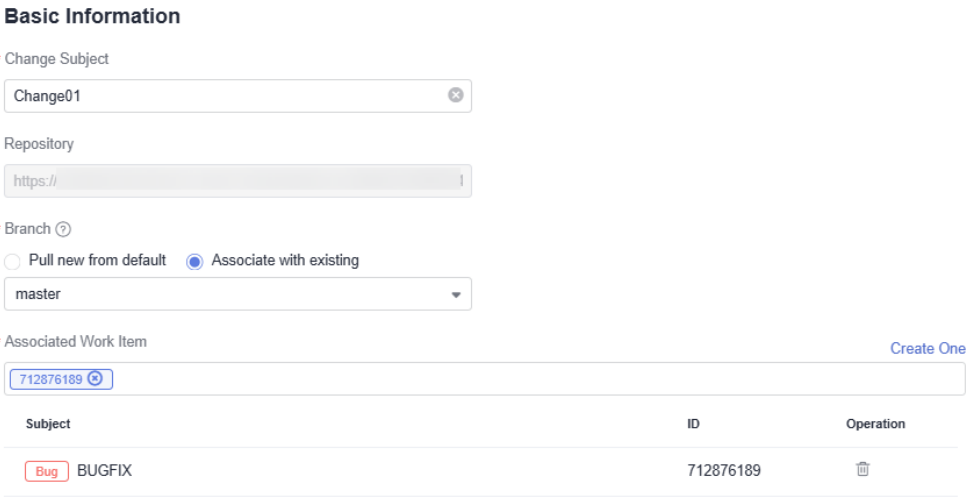


Table 10-3 Change parameters

Parameter	Description	Example Value
Change Subject	Name of the change. Enter a maximum of 256 characters.	Enter **Change01**.
Repository	Name of the repository associated with the microservice. The repository cannot be changed.	-

Parameter	Description	Example Value
Branch	You can pull a new branch from the default branch or associate the change with an existing branch. After the change is released through the change-triggered pipeline, the code branch will be automatically merged to the default branch of the microservice.	Select **Associate with an existing branch** and select **master** from the drop-down list.
Associated Work Item	Select started or ongoing work items in CodeArts Req.	Select the work item **BUGFIX** created in **Preparations**.

Step 6 Click **OK**. A message is displayed, indicating that the change is created successfully. The change details page is displayed.

The details page displays the change overview, associated work items, and operation history. You can submit the change for release, exit release, or cancel the change.

Figure 10-4 Change details

A change's lifecycle includes developing, to be released, releasing, and released.

For a change in the **Developing** status, click **Edit Work Item** to modify the associated work item.

The following describes how to submit a change for release, exit release, and cancel the change.

- Submit for release

  For a change in the **Developing** status, click **Submit for Release**. The **Submit for Release** dialog box is displayed.

  - If the microservice does not have a change-triggered pipeline, create one by referring to **Creating a Change-triggered Pipeline**.
  - If there is a change-triggered pipeline, click **OK** to submit the change.

  After the change is submitted, the change status changes from **Developing** to **To be released**.

- Exit release

  For a change in the **To be released** or **Releasing** status, click **Exit Release** to exit the release. The change status will change to **Developing**. For a change in the **Releasing** status, if the change-triggered pipeline is running, you cannot exit release.

- Cancel a change

  For a change in the **Developing** status, click **Cancel Change**.

  In the displayed dialog box, click **OK**. The change status changes to **Canceled** and the change will be deleted.

----End

Creating a Change-triggered Pipeline

Step 1 **Access the CodeArts Pipeline homepage** through a project.

Step 2 Click the **Microservices** tab.

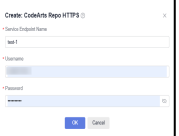
Step 3 Click a microservice name. The **Overview** page is displayed.

Step 4 Switch to the **Pipelines** tab.

Step 5 Click **Create Pipeline**. On the displayed page, configure parameters.

Table 10-4 Pipeline parameters

Parameter	Description	Example Value
Name	Pipeline name, which is generated based on the creation time by default. Enter a maximum of 128 characters, including letters, digits, underscores (_), and hyphens (-).	Enter **Pipeline-1**.
Agency URN	Unique identifier of IAM agency. If set: uses this agency identity and permissions to access other cloud services. If not set: uses the operator's identity and permissions.	-
Project	Project to which the microservice belongs.	The default value is **project01**.
Pipeline Source	Source of the code repository. Only **Repo** is supported.	Select **Repo**.

Parameter	Description	Example Value
Repository	Name of the repository associated with the microservice. If you change the code repository of a microservice, the repository for all pipelines of the microservice will also be changed.	Select **Repo01**.
Default Branch	The default branch associated with the microservice. If you change the default branch of a microservice, the default branch for all pipelines of the microservice will also be changed.	Select **master**.
Repo Endpoint	• Configure an endpoint to elevate permissions on repository operations. Endpoints are used for change-triggered pipelines and repository operation extensions. Click **Create One** to create a Repo endpoint. For details, see *Creating Service Endpoints*. • If you use an incorrect username or password when creating this endpoint, the pipeline will fail to run. For details, see **FAQs**.	Create an endpoint as shown in the following figure. 
Alias	Repository alias. Enter a maximum of 128 characters, including letters, digits, and underscores (_). After you set a repository alias, system parameters will be generated based on the alias. For example, *Alias*_**REPOSITORY_NAME** indicates the repository name. You can check the generated parameters on the **Parameter Configuration** page and reference them in a pipeline in the format of *${Parameter name}*.	-
Change-based Trigger	If **Change-based Trigger** is enabled for a pipeline, this pipeline is marked with  . A microservice can have only one change-triggered pipeline.	Click  to enable **Change-based Trigger**.
Description	Pipeline description. Enter a maximum of 1,024 characters.	-

Step 6 Click **Next**. On the displayed page, select a template or select **Blank Template**.

Step 7 Click **OK**. On the **Task Orchestration** tab page, **orchestrate a pipeline**.

Step 8 Click **Save**. A message is displayed, indicating that the pipeline is saved successfully.

----End

Executing a Change-triggered Pipeline

Step 1 Access the CodeArts Pipeline homepage through a project.

Step 2 Click the **Microservices** tab.

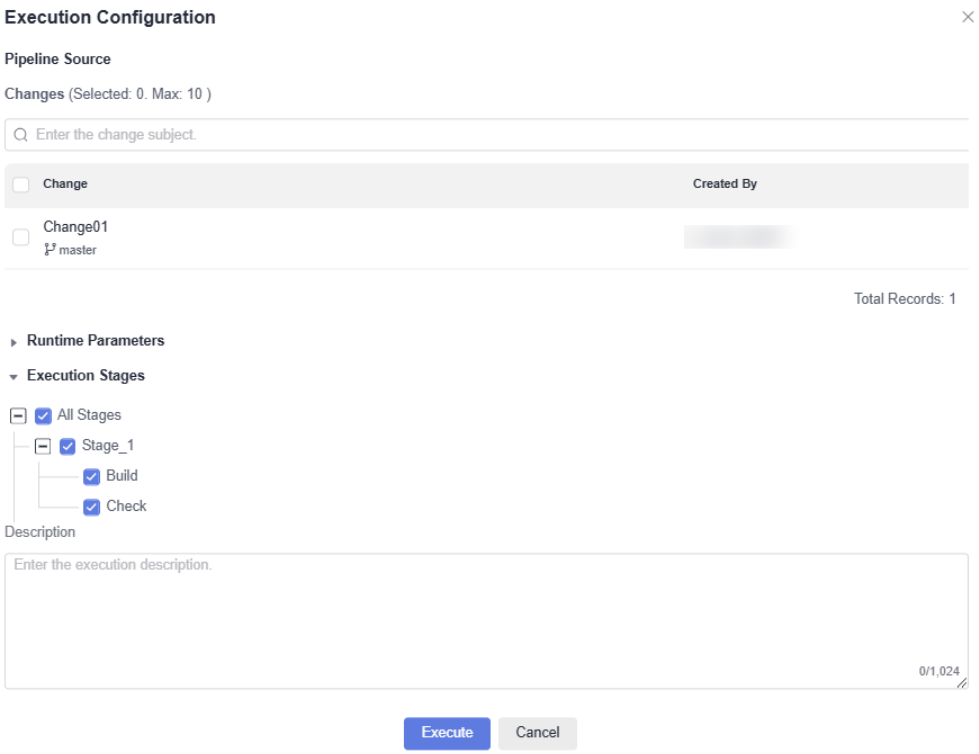
Step 3 Click a microservice name. The **Overview** page is displayed.

Step 4 Switch to the **Pipelines** tab.

Step 5 Click a pipeline name. The pipeline **Execution History** page is displayed.

Step 6 Click **Execute** in the upper right corner and configure the execution.

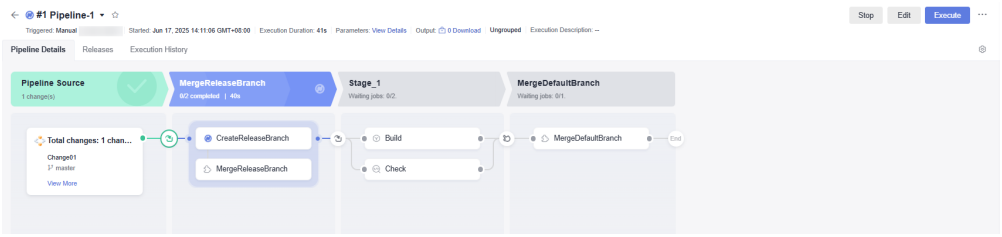
Figure 10-5 Execution configurations for a change-triggered pipeline



- **Changes**: Changes in **To be released** or **Releasing** status are displayed. Select one or more changes.
- **Runtime Parameters**: (Optional) Set runtime parameters and then save them. For details, see **Using a Parameter in a Pipeline**.
- **Execution Stages**: Select one or more jobs to execute. By default, all jobs are selected for execution.
- **Description**: Describe the debugging about the execution.

Step 7 After the configurations, click **Execute**. The pipeline details page is displayed.

Figure 10-6 Executing a change-triggered pipeline



When the change-triggered pipeline is running, there are **MergeReleaseBranch** and **MergeDefaultBranch** stages.

- **MergeReleaseBranch**: The change-triggered pipeline automatically pulls a new branch from the master branch and integrates all change feature branches into the new branch.
- **MergeDefaultBranch**: The new branch is merged to the master branch.

Step 8 After the execution is complete, check the pipeline execution result. For details about possible problems, see **FAQs**.

After the pipeline was successfully executed, the status of all selected changes changed to **Released**.

- Click the pipeline name to go to its details page.
  - Click **View More** on the pipeline source card. In the displayed dialog box, review the selected changes.
  - Click a change name to go to its details page.
- Click the **Releases** tab.
  - All changes in the **To be released** and **Releasing** statuses are displayed.
  - You can enter a keyword in the search box to search for a change.
  - Click in the **Operation** column. In the displayed dialog box, click **OK**. The change status will become **Developing**. For a change in the **Releasing** status, you can exit the release only after the change-triggered pipeline execution is complete or stopped.

----End

Tutorial Video

This video shows how to release a change through a change-triggered pipeline.

11 Managing Pipeline Extensions

11.1 Extensions Overview

CodeArts Pipeline has a collection of built-in extensions covering build, check, deployment, and test. You can use these extensions for pipeline orchestration. Enterprises can quickly connect existing tools to the Pipeline service or develop their own extensions through the extension platform. CodeArts Pipeline provides a visual, low-code, and open extension market to adapt to service requirements.

Accessing the Extension Platform

- Method 1
  a. **Access the CodeArts Pipeline homepage**.
  b. On the CodeArts Pipeline homepage, choose **Services** > **Extensions**.
- Method 2
  a. **Access the CodeArts Pipeline homepage**.
  b. Create or edit a pipeline.
  c. On the **Task Orchestration** page, add or edit a job. On the displayed window, click **More Steps** in the upper right corner.

The extension page displays all available extensions. You can click the card of an extension to check its details.

Scenarios

- You can use extensions provided by CodeArts Pipeline (such as **KubernetesRelease**) to connect to cloud services.
- You can use official tools to develop extensions. CodeArts Pipeline allows you to compile service scripts in mainstream languages, such as Shell, Node.js, Python, and Java. Some basic extensions can be used together with custom executors to provide more execution modes.
- You can also customize extensions to connect to third-party CI/CD tools.

11.2 Pipeline Official Extensions

CodeArts Pipeline provides official extensions as listed in **Table 11-1**.

Table 11-1 Pipeline Official Extensions

Type	Name	Description
Build	Build	Calls CodeArts Build capabilities. CodeArts Build provides an easy-to-use, cloud-based build platform that supports multiple programming languages, helping you achieve continuous delivery with shorter period and higher efficiency. With CodeArts Build, you can create, configure, and run build tasks with a few clicks. CodeArts Build also supports automated code retrieval, build, and packaging, as well as real-time status monitoring. **Learn more.**
	Build-Template	This extension can be configured only in a pipeline template. When a pipeline is generated based on the template, the extension automatically creates a build job and configures the job in the generated pipeline.
Test	TestPlan	Calls CodeArts TestPlan capabilities. CodeArts TestPlan is a one-stop cloud testing platform provided for software developers. It covers test management and API tests and integrates the DevOps agile testing concepts, helping you improve management efficiency and deliver high-quality products. **Learn more.**
	TestPlan-Template	This extension can be configured only in a pipeline template. When a pipeline is generated based on the template, the extension automatically creates an API test job and configures the job in the generated pipeline.
Deploy	Deploy	Calls CodeArts Deploy capabilities. CodeArts Deploy allows you to visually deploy applications in VMs or containers by using Tomcat, Spring Boot, and other templates. You can also flexibly orchestrate atomic actions for deployment. CodeArts Deploy standardizes your deployment environment and processes by integrating with CodeArts Pipeline. **Learn more.**
	Deploy-Template	This extension can be configured only in a pipeline template. When a pipeline is generated based on the template, the extension automatically creates a deployment job and configures the job in the generated pipeline.
	KubernetesRelease	Allows you to deploy container images to Cloud Container Engine (CCE) or native Kubernetes clusters. It supports rolling release and blue-green deployment.

Type	Name	Description
	CloudNative Release	Allows you to orchestrate release policies for environments, such as rolling release and grayscale release.
	DeveloperJointCommissioning	This extension is a Kubernetes-based cloud native R&D environment delivery platform. It provides developers with capabilities such as continuous R&D environment delivery and component joint debugging, and helps enterprises reduce R&D environment resource costs.
	CloudReleaseExecutor	This extension executes the deployment policy configured in the **CloudNativeRelease** extension and is called by the **CloudNativeRelease** extension in the background. Manual configuration and direct calls are not supported.
	DeployMicroserviceInstance	This extension updates the deployment configuration of a microservice instance in a specific environment.
Check	Check	Calls CodeArts Check capabilities. CodeArts Check is a cloud-based management service that checks code quality. Developers can easily perform static code and security checks in multiple languages and obtain comprehensive quality reports. CodeArts Check also provides bug fixing suggestions and trend analysis to control code quality and reduce costs. **Learn more.**
	Check-Template	This extension can be configured only in a pipeline template. When a pipeline is generated based on the template, the extension automatically creates a Check job and configures the job in the generated pipeline.
	BranchCheck	Specifies the target branch. If the current running branch lags behind the specified branch, the pipeline fails to run.
	Pass-Conditions-of-Standard-Policies	A standard extension policy for access control.
Normal	CreateTag	Creates and pushes tags for code repositories.
	Subpipeline	Configures and calls other pipeline tasks in a project.
	JenkinsTask	Calls Jenkins tasks. Currently, this function is available in **LA-Mexico City2**, **LA-Sao Paulo1**, and **AP-Singapore**.
	DelayedExecution	Pauses pipeline for a period of time or until a specified time. You can manually resume or stop a pipeline, or delay the execution for a maximum of three times.

Type	Name	Description
	ManualReview	Creates manual review tasks by assigning one person or one group.
	GitClone	Clones the code repositories configured in the pipeline source, which can be used together with shell commands and Maven build. Currently, this function is available in **LA-Mexico City2**, **LA-Sao Paulo1**, **AP-Singapore**, and **TR-Istanbul**.
	ExecuteShellCommand	Runs shell commands.
	upload-obs	Uploads files to Huawei Cloud OBS.
	download-obs	Downloads files from OBS to local.
	CreateReleaseBranch	Creates a release branch based on the default branch of a microservice. This extension is automatically configured by a change-triggered pipeline.
	MergeReleaseBranch	Merges a feature branch into a release branch. This extension is automatically configured by a change-triggered pipeline.
	MergeDefaultBranch	Merges a release branch into the default branch of a microservice. This extension is automatically configured by a change-triggered pipeline.
	Execute Docker Operations	Executes specified Docker operations. This extension can run only in a custom executor.

11.3 Customizing Extensions on the GUI

You can create, configure, and manage extensions on the GUI.

Notes and Constraints

- Version of the extension should be in *X.X.X* format. Each digit ranges from 0 to 99.
- Information on the **Version Info** page cannot be modified later.

Creating an Extension

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the CodeArts Pipeline homepage, choose **Services** > **Extensions**.

Step 3 Click + Standard Create.

Step 4 Set basic information. For details, see **Table 11-2**.

Table 11-2 Extension information

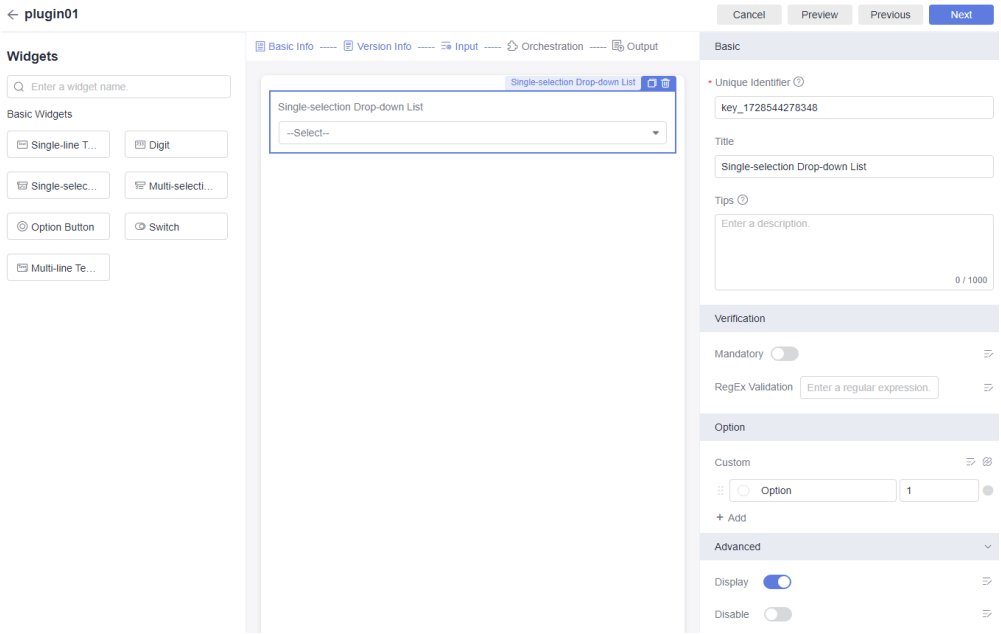
Parameter	Description
Icon	Icon of the extension. Upload an image in PNG, JPEG, or JPG format, with a file size no more than 512 KB (recommended: 128 x 128 pixels). If no image is uploaded, the system generates an icon.
Name	The extension name displayed in the extension platform. Enter only spaces, letters, digits, underscores (_), hyphens (-), and periods (.) with a maximum of 50 characters.
Unique Identifier	ID of the extension. Once set, this parameter cannot be changed. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 50 characters.
Type	Type of the extension, which can be **Build**, **Check**, **Test**, **Deploy**, or **Normal**. Once set, this parameter cannot be changed.
Description	Purposes and functions of the extension. The description can be edited. Enter no more than 1,000 characters.

Step 5 Click **Next**. On the **Version Information** page, set the version and description.

Step 6 Click **Next**. The **Input** page is displayed. Configure widgets as needed.

You can drag and drop widgets to generate visual forms and to streamline pipeline contexts. Multiple preset widgets are available: **Single-line Text Box**, **Digit**, **Single-selection Drop-down List**, **Multi-selection Drop-down List**, **Option Button**, **Switch**, **Multi-line Text Box**, and so on.

Figure 11-1 Orchestrating widgets



Drag required widgets to the middle area. Click a widget and configure its parameters on the right part of the page.

Table 11-3 Widget parameters

Category	Parameter	Description	Widget
Basic	Unique Identifier	Unique ID of the widget. The ID is used to obtain widget input. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 200 characters.	All
	Title	Name of the widget. The name will be displayed on the pipeline job orchestration page. Enter no more than 140 characters.	All
	Tips	Tooltip of the widget. Enter no more than 1,000 characters.	All
	Placeholder	Informative message displayed in the text box. For example, what value should be input.	Single-line Text Box
	Accuracy	Number of decimal places allowed in a widget value: **0** to **4**.	Digits
	Default Value	Default value of the widget.	Single-line Text Box, Digit, Switch, Multi-line Text Box, and Metrics
Verification	Mandatory	Whether the widget content is mandatory. Error messages can be set.	Single-line Text Box, Digit, Single-selection Drop-down List, Multi-selection Drop-down List, Option Button, and Multi-line Text Box
	RegEx Validation	Verifies the widget content. You can set error messages.	Single-line Text Box, Digit, Single-selection Drop-down List, Multi-selection Drop-down List, and Multi-line Text Box
	Word limit	Max. widget characters.	Multi-line Text Box

Category	Parameter	Description	Widget
Option	Custom	Options available for the widget. Click + Add to add an option. Click ⊖ to delete an option. • Option name: option displayed on the extension configuration page. • Value: value delivered when the extension is running. In addition to manual configuration, set options by: • APIs: Set options by configuring web APIs. Click ✎ on the right. On the displayed dialog box, you can configure parameters after enabling the function. For details, see **Table 11-4**. • Context: Configure data source to obtain the URL of the pipeline source or IDs of build jobs. Click 🔗 next to **Custom**. The context dialog box is displayed. You can configure parameters after enabling the function.	Single-selection Drop-down List, Multi-selection Drop-down List, and Option Button
Advanced	Display	Whether the widget is visible. You can click ✎ on the right to set display conditions.	All
	Disable	Whether the widget is disabled (not disabled by default). You can click ✎ on the right to set conditions.	All

Table 11-4 API parameters

Parameter	Description
Enabled	By enabling the function, you can set options by configuring APIs.
Linked Attribute	Associates the selected widgets with the API to transfer parameters. When a widget value is changed, the new value is used to call the API again.
URL	Only HTTPS protocol is supported.

Parameter	Description

Returned Data Path	The widget used must be list data. In the following response body example, the returned data path is **result.parameters**.
	{
	"result": {
	"total": 2,
	"parameters": [
	{
	"id": 3353753,
	"name": "parameters01"
	},
	{
	"id": 3353697,
	"name": "parameters02"
	}
	]
	},
	"status": "success"
	}

Option Value	Set this parameter to the value of the corresponding field in the returned data path. This parameter is delivered when the extension is running.
Option Name	Set this parameter to the value of the corresponding field in the returned data path. This parameter is displayed on the extension configuration page.
Params	Params parameters of the API request body.
Header	Header parameters of the API request body.
Remote Search	Enable this function to add a remote search field. For extension search, the entered value will be used as the value of the remote search field to call the API again. • Params parameter: The parameter type of the search field is the Params parameters of the API request body. • Body parameter: The parameter type of the search field is the Body parameters of the API request body.
Search Tip	Describes the search function and is displayed below the search bar. Max. 100 characters.

Step 7 Click **Next**. On the displayed **Orchestration** page, you can add the **ExecuteShellCommand** extension.

- **ExecuteShellCommand**: executes shell commands entered by users.

  Enter shell commands. Commands will be executed when a pipeline calls the extension. The commands indicate the actual service logic implementation process of an extension. For more input and output configurations, see **Customizing Shell Commands**.

Step 8 Click **Next**. On the **Output** page, click **Add Configuration** to configure output information, including **output**, **link**, **table**, and **metric**. After the pipeline is executed, go to the job result page to check the extension execution results. The result information is displayed by result type.

- **output**: displayed on the **Others** card. It outputs data together with shell commands.
- **metric**: displayed on the **Others** card. It outputs metric thresholds. The thresholds information can be referenced in an extension and finally applied to pipelines.
- **link**: displayed on the **Link** card. You can click the link to go to the corresponding page.
- **table**: displayed on the **Tabular Data** page. It is an object array and displays the array information in a table.

Step 9 After the configuration, click **Release** or **Release Draft**.

- Draft release

  Click **Release Draft** to release a test version.

  – You can configure a draft extension in a pipeline for debugging. After debugging, the draft extension can be officially released, so that other members of the current tenant can use the extension.

  – All draft versions are marked with **Draft**.

  – Only one draft is allowed. If there is already a draft, no more versions can be created until you officially release or delete the draft.

- Official release

  Click **Release** to release an official version. An official extension has a unique version number. All members of the current tenant can use this version in a pipeline.

----End

Customizing Shell Commands

When registering an extension or creating an extension version, you can use shell commands to implement service logic. The commands usually involve interaction with all kinds of data during pipeline execution. This section describes how to implement extension logic through data input and output.

- Data Input

  The obtained data consists of low-code GUI input, pipeline run parameters, and other information.

  – Low-code GUI input: Obtain the low-code user interface output using environment variables, for example, **echo $*{Widget ID}***.

  – Pipeline run parameters: Some pipeline run parameters will be delivered to environment variables, as shown in the following table.

Table 11-5 Pipeline environment variables

Variable	Description
STEP_NAME	Step name of the pipeline.
STEP_ID	Step ID of the pipeline.

Variable	Description
PLUGIN_VERSION	Version of the extension.
PIPELINE_ID	Pipeline ID.
PIPELINE_RUN_ID	Pipeline execution ID.
PLUGIN_NAME	Extension name.
PROJECT_ID	Project ID.
JOB_ID	Job ID of the pipeline.
RESULT_MSG_PATH	Directory for storing the file of extension execution result. The ${*STEP_ID*}_result.json and ${*STEP_ID*}_metrics.json files are written to this directory to report the execution result to the pipeline.

  - Other information: Obtain information by interacting with external data through Git, Wget, and Curl.

- Data Output

  Once executed, the custom extension can read file information in a specified path and obtain the metric data output.

  a. On the configuration page, configure the thresholds output of the extension.

  b. During development, the *${STEP_ID}*_**result.json** and *${STEP_ID}*_**metrics.json** files are stored in a specified path so that metric values can be parsed.

  Table 11-6 Output files

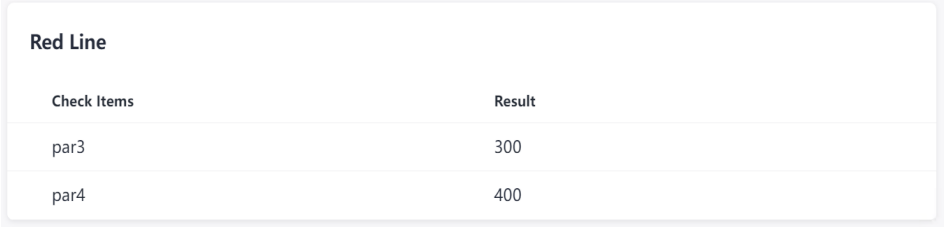
File	Description
${RESULT_MSG_PATH}/${STEP_ID}_result.json	The output is a text file in {"par1":123, "par2":456} format. After the pipeline is executed, the result will be displayed as the corresponding task result. Only extensions of the check type can display the result.

File	Description
${RESULT_MSG_PATH}/${STEP_ID}_metrics.json	The output is a text file in {"par1":123, "par2":456} format. The **Metrics** widget should be configured. After the extension is executed, the threshold configured for the **Metrics** widget and the content of *${STEP_ID}*_**metrics.json** are parsed for pipeline pass conditions. Notes: • During parsing, empty key values in the **Metrics** widget will be ignored. • If the key value configured for the **Metrics** widget cannot be found in the *${STEP_ID}*_**result.json** file, the specified threshold value will be used.

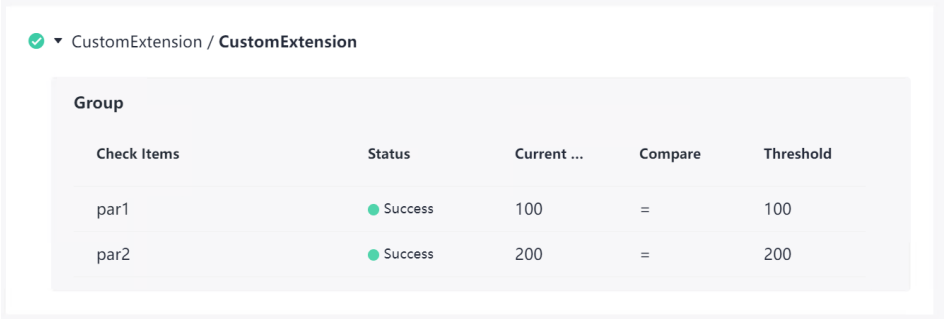
Example: **par1** and **par2** for task result display; **par3** and **par4** for pass conditions. The sample code is as follows:

```
# Optionally, construct the extension output.
echo '{"par1":100,"par2":200}' > ${RESULT_MSG_PATH}/${STEP_ID}_result.json
echo '{"par3":300,"par4":400}' > ${RESULT_MSG_PATH}/${STEP_ID}_metrics.json
```

c. After the extension run is complete, click the extension card to view the output.



If policies are configured for the current extension and applied to the pipeline pass conditions, click the pass conditions to check the check status.



11.4 Creating an Extension by Uploading an Extension Package

You can create and configure a pipeline extension by writing code for better flexibility and maintainability.

Preparing an Extension Package

Extension package

Notes

- The extension package must be in the ZIP format.
- The root directory of the package must contain a metadata file **codearts-extension.json**. For more information about the file, see **codearts-extension.json**.
- The **resources.json** file can be encoded only using UTF-8.

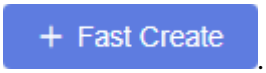
File structure

```
extension.zip                    # ZIP package of the extension
  | -- scripts                   # (Optional) Script folder for storing scripts that contain extension execution logic.
  |    | -- xxx                  # Script that contains extension execution logic
  | -- i18n                      # (Optional) Contents in multiple languages
  |    | -- zh-cn                # Contents in Chinese environment
  |         | -- resource.json   # Internationalization resources
  |    | -- en-us                # Contents in English environment
  |         | -- resources.json  # Internationalization resources
  | -- codearts-extension.json   # (Mandatory) Extension execution file (in JSON format), including basic information, inputs, and execution
```

Creating an Extension

Step 1 **Access the CodeArts Pipeline homepage**.

Step 2 On the CodeArts Pipeline homepage, choose **Services** > **Extensions**.

Step 3 Click 

.

Step 4 Set basic information. For details, see **Table 11-7**.

Table 11-7 Extension information

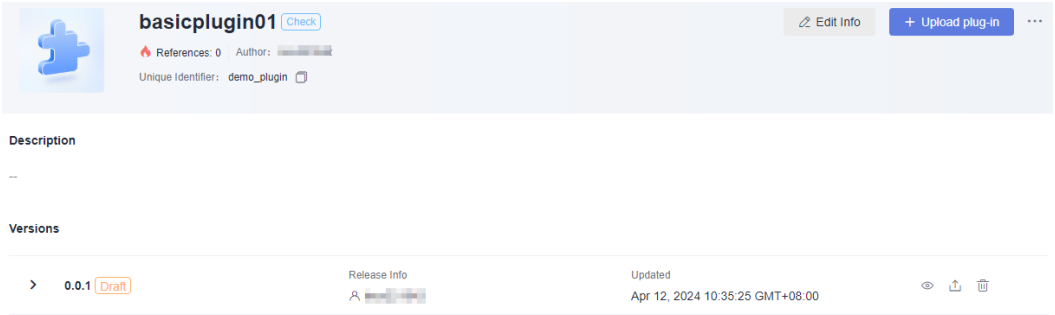
Parameter	Description
Icon	Icon of the extension. Upload an image in PNG, JPEG, or JPG format, with a file size no more than 512 KB (recommended: 128 x 128 pixels). If no image is uploaded, the system generates an icon.
Name	The extension name displayed in the extension platform. Enter only spaces, letters, digits, underscores (_), hyphens (-), and periods (.) with a maximum of 50 characters.

Parameter	Description
Unique Identifier	ID of the extension. This value should be consistent with the **name** field of the **codearts-extension.json** file. Once set, this parameter cannot be changed. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 50 characters. Mapping between the extension type and the value of **category**: • Build: **Build** • Check: **Gate** • Deploy: **Deploy** • Test: **Test** • Normal: **Normal**
Type	Type of the extension, which can be **Build**, **Check**, **Test**, **Deploy**, or **Normal**. Once set, this parameter cannot be changed.
Description	Purposes and functions of the extension. The description can be edited. Enter no more than 1,000 characters.

Step 5 Click **Next**. The version management page is displayed.

Step 6 On the displayed page, click  . In the displayed dialog box, select the desired extension (with input definition and execution script) and upload it. After the upload is successful, the version will be marked with **Draft**.

Figure 11-2 Uploading an extension



Step 7 Using an extension in a pipeline

Create a pipeline. On the **Task Orchestration** page, create a job, add the registered basic extension, and set parameters.

Step 8 Click **Save and Execute**. After the execution is complete, click the extension name to view the execution result.

Step 9 (Optional) After debugging, publish the extension as an official version.

1. Go to the extension page.
2. Click the registered basic extension.

3. On the displayed page, click on the right to publish the version as an official version.

The draft version can be overwritten for multiple times. However, the official version cannot be updated. You can click **Upload plug-in** in the upper right corner to upload a new version.

----End

codearts-extension.json

Example:

```
{
   "type": "Task",
   "name": "demo_plugin",
   "friendlyName": "Extension name",
   "description": "This is an extension.",
   "category": "Gate",
   "version": "0.0.2",
   "versionDescription": "Updated based on the initial version 0.0.1",
   "dataSourceBindings": [],
   "inputs": [
      {
   "name": "samplestring",                        # Use ${samplestring} in a script to obtain the value configured by an executor in a pipeline
   "type": "input",                               # Different types correspond to different functions
   "description": "Sample String",                # Description of input
   "defaultValue": "00",                          # Default value when the value of the required field is false
   "required": true,                              # Reset defaultValue if the required field is true, or the default value will be used
         "label": "Text box",                     # input information displayed on the pipeline editing page
         "validation": {
            "requiredMessage": "Enter a value",          # (Optional) The message displayed when the required field is left blank
            "regex": "^[a-zA-Z0-9-_\\u4e00-\\u9fa5]{1,32}$", #(Optional) RegEx validation
            "regexMessage": "Type error"                 # (Optional) The message displayed when RegEx validation failed
         }
      }
   ],
   "execution": {
      "type": "Shell",
      "target": "scripts/execution.sh"
   },
   "outputs": [{
         "name": "okey",                  # Output name.
         "type": "output",                # Output type: output or metrics.
         "description": "Description",
         "prop": {
            "defaultValue": "123"         # Default value
         }
      },
      {
         "name": "mkey",
         "type": "metrics",
         "description": "description",          # Description of output
         "prop": {
            "defaultValue": "213",
            "group": "213"                # Corresponding to the group name in pass conditions         }
      }
   ]
}
```

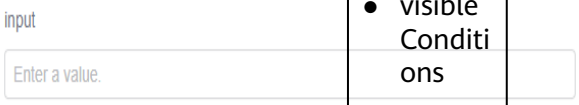
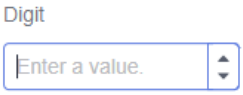
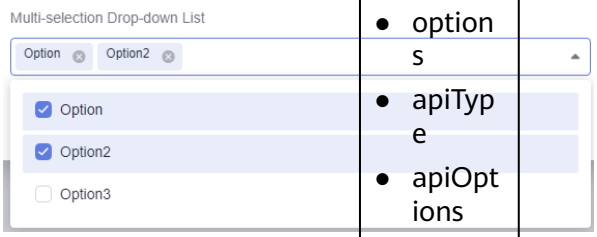
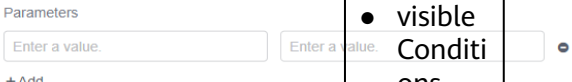
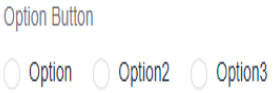
The parameters of **codearts-extension.json** are described in the following table.

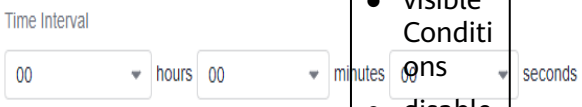
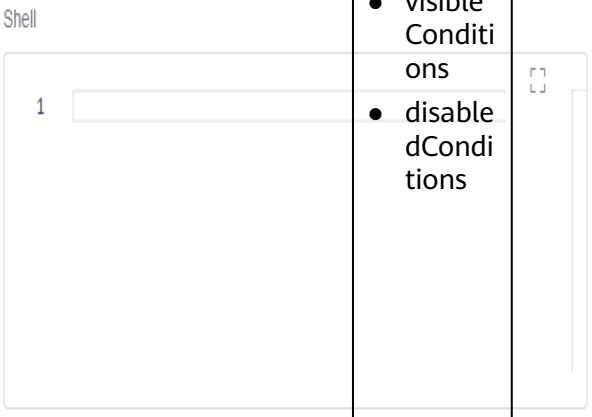
Table 11-8 Parameters

Parameter	Description
type	The value is fixed to **Task**, which indicates an extension type.
name	Same as the **Unique Identifier** field set for extension registration
friendlyName	Same as the **Name** field set for extension registration
category	Same as the **Type** field set for extension registration, which can be: • Build: corresponds to the extension of the **Build** type. • Test: corresponds to the extension of the **Test** type. • Gate: corresponds to the extension of the **Check** type. • Normal: corresponds to the extension of the **Normal** type. • Deploy: corresponds to the extension of the **Deploy** type.
version	Version of the extension, which consists of three numbers separated by dots (.), with each number ranges from 0 to 99. Modify this parameter only when you need to add an official version.
description	Description of the extension.
versionDescription	Description of the extension version's unique features.
dataSourceBindings	Disabled currently. Set it to **[]**.
inputs	Extension input content. This parameter corresponds to the extension display format on the pipeline page. The values can be referenced by environment variables in service scripts.
execution	Extension execution content. The **type** field indicates the service script language, and the **target** field indicates the path to the execution file. You are advised to create a **scripts** folder and place the content under it.
outputs	Extension output content. The value can be used as the gate metrics. **output** has different display. **output** is used in the pipeline context. The output content can be used in subsequent stages or displayed in the task execution result. **metrics** data is used to create pass rules. For details, see **Table 11-12**.

Supported inputs are listed in the following table.

Table 11-9 inputs

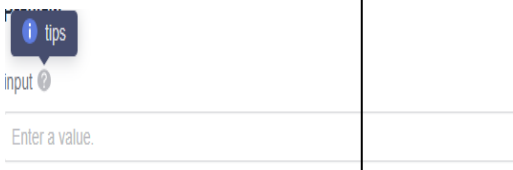
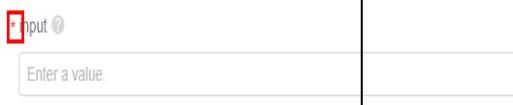
Type	Widget	Example	extendProp
input	Single-line Text Box		• visibleConditions • disabledConditions
inputNumber	Digit		• visibleConditions • disabledConditions
switch	Switch		• visibleConditions • disabledConditions
singleSelect	Single-selection Drop-down List		• options • apiType • apiOptions
multipleSelect	Multi-selection Drop-down List		• options • apiType • apiOptions
keyValuePair	Key-Value Pair		• visibleConditions • disabledConditions
radio	Option Button		options

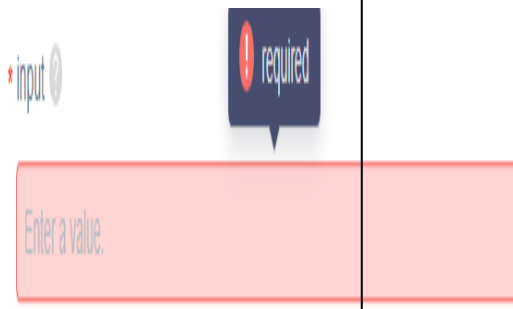
Type	Widget	Example	extendProp
timeInterval	Time Interval		• visibleConditions • disabledConditions
shell	Shell		• visibleConditions • disabledConditions
endpoint:*${module_id}*	Endpoint		• visibleConditions • disabledConditions

inputs fields are listed in the following table.

Table 11-10 inputs fields

Field	Description	Mandatory	Remarks
name	Unique ID of the widget	Yes	The value must be unique.
label	Widget title	Yes	-
type	Widget type	Yes	-
defaultValue	Initial value	No	Initial default value of a widget. This field can be left blank.

Field	Description	Mandatory	Remarks
description	Widget description	No	The infotip message next to a widget name 
required	Whether a parameter is mandatory.	No	Fields marked with asterisks (*) are mandatory. 

validation	Validation information, which is an object that contains the **requiredMessage**, **regex**, and **regexMessage** properties.
	{
	requiredMessage: '', // Prompt message for a mandatory field
	regex: '', // RegEx validation
	regexMessage: '' // The message displayed when RegEx validation failed
	}
		No	

extendProp	Extension field
	{
	visibleConditions: [],
	disabledConditions: []
	...
	}
		No	For details about extendProp, see **Table 11-11**.

extendProp functions are listed in the following table.

Table 11-11 extendProp functions

Field	Description	Mandatory	Remarks

visibleConditions	Widgets are displayed if conditions are met.	No	Multiple conditions can be included:
			`[{},{},{},...]`
			Example:
			`[{comp:'key_001',symbol:'===', value: 'xxx'}]`
			In this example, widget A will be displayed if widget B has a unique ID of *key_001* and has a value that is equal to (===) *xxx*.
			symbol can be:
			• ===: Equal
			• !==: Not equal
			• empty: Empty
			• notEmpty: Not empty

disabledConditions	Widgets are disabled if conditions are met.	No	Multiple conditions can be included:
			`[{},{},{},...]`
			Example:
			`[{comp:'key_002',symbol:'!==', value: 'yyy'}]`
			In this example, widget A will be disabled if widget B has a unique ID of *key_002* and has a value that is not equal to (!==) *yyy*.
			symbol can be:
			• ===: Equal
			• !==: Not equal
			• empty: Empty
			• notEmpty: Not empty

options	The fixed drop-down list. The field's type is **list**.	No	Example: `[{label:'option 1',value: 1},{label:'option 2',value: 2}]`

Field	Description	Mandatory	Remarks
apiType	Options in the drop-down list box: • **fixed**: The values in **options** are used as options. • **api**: API requests, available only when **apiOptions** is configured.	No	If this field is left blank, **fixed** is used.

Field	Description	Mandatory	Remarks

apiOptions	JSON body, including parameters used by APIs.	No	Example:
			'{"body":{"xxx":111},"header":{"yyy":222},"linkedFields":["key_001"],"method":"POST","params":{"zzz":333},"remote":true,"remoteName":"xxx","remoteQueryField":"body","responseUrl":"data","label":"name","value":"id","url":"https://sss/lll/mmm"}'
			JSON (parsed):
			{
			body: {xxx:111}, // Request parameters for calling an API
			header: {yyy: 222}, // Request header field
			params: {zzz: 333}, // Request parameters for calling an API
			linkedFields: ['key_001'], // This field is associated with other widgets. When the values of other widgets change, the API will be called again and the current options will be cleared.
			method: 'POST', // Request mode: POST or GET
			remote: true, // Whether to enable remote search
			remoteName: 'tt', // Field to be searched in a remote search
			remoteQueryField: 'body', // Parameter passing method for the remote search field: body or params
			responseUrl: 'data', // The path of the option list obtained in the returned data
			label: 'name', // Parameter corresponding to the label in ComboBox
			value: 'id', // Parameter corresponding to the actual value in ComboBox
			url: 'https://sss/lll/mmm' // API URL
			}

outputs fields are listed in the following table.

Table 11-12 outputs fields

Field	Description	Mandatory	Remarks
name	Output name.	Yes	The value must be unique.

Field	Description	Mandatory	Remarks
type	Output type. • **output**: displayed on the **Others** card and used in **pipeline contexts**. • **metrics**: displayed on the **Others** card. It outputs metric thresholds. The thresholds information can be referenced in the extension and finally applied to pipelines. For details, see **Configuring a Rule**. • **link**: displayed on the **Link** card. You can click the link to go to the corresponding page. • **table**: displayed on the **Tabular Data** page. It is an object array and displays the array information in a table.	Yes	-
description	Description of the value. This parameter is mandatory when **type** is set to **metrics**.	-	-

Field	Description	Mandatory	Remarks
prop	When **type** is set to **metrics**, **group** corresponds to the group name in pass rules and this parameter is mandatory.	-	-

11.5 Executing Images

You can use the **ExecuteImage** extension to download public images from SWR to a custom executor and start the images.

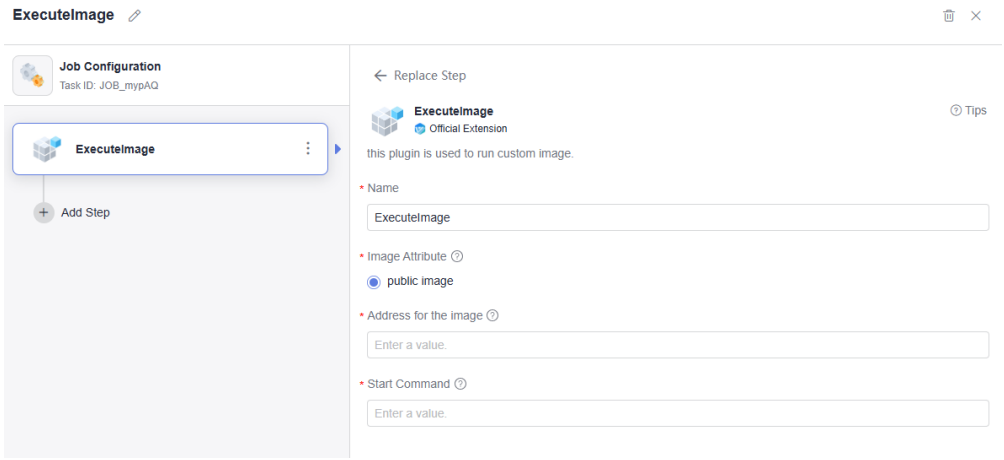
Constraints

This extension supports only custom executors. Before using a custom executor, add an agent pool. For details, see **Agent Pools**.

Configuration Method

Step 1 Add the **ExecuteImage** extension when you **orchestrate a pipeline**.

Figure 11-3 Extension for executing images



Step 2 Set parameters as shown in the following table.

Parameter	Description
Name	Extension name. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter 1 to 128 characters.
Image Attribute	Only public images are supported.
SWR Image Address	Address of the SWR images to be downloaded. To obtain the address: 1. Log in to SWR. 2. In the navigation pane, click **My Images**, click the image name to go to the image details page. 3. Click to copy the image download command. The part following **docker pull** is the image path.
Startup Command	Container startup command. Enter Docker commands to run specific applications or scripts in the container.

----End

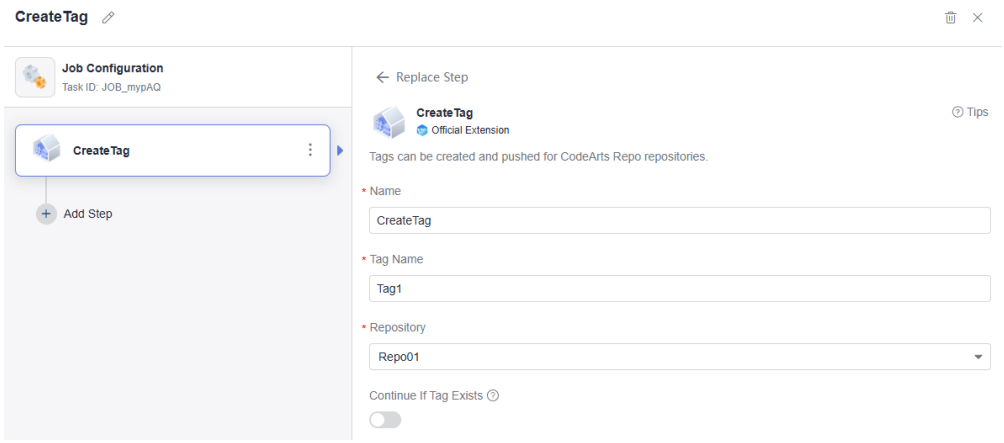
11.6 Pushing Tags for Code Repositories

You can create and push tags for code repositories.

Configuration Method

Step 1 Add the **CreateTag** extension when you **orchestrate a pipeline**.

Figure 11-4 Adding the CreateTag extension



Step 2 Set parameters as shown in the following table.

Parameter	Description
Name	Extension name. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter at least one character.
Tag Name	Tag name.
Repository	Select the corresponding code repository.

- Click the toggle to enable **Continue If Tag Exists**. If the created tag exists in the code repository, no error is reported and the pipeline continues to run. Otherwise, the execution fails.

----End

11.7 Calling Third-Party APIs

You can configure web APIs to call third-party APIs.

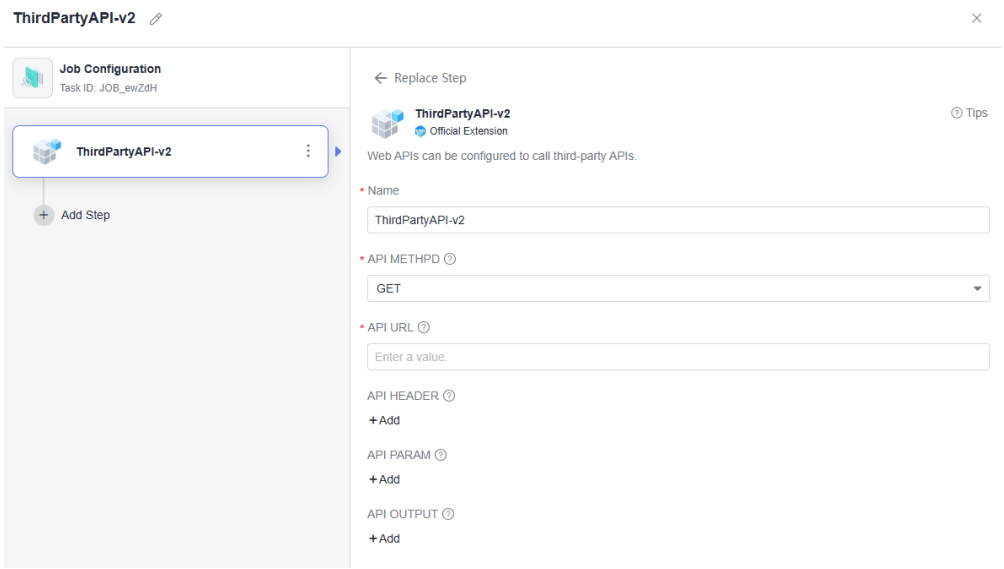
Constraints

- The **ThirdPartyAPI-v2** extension requires certificate verification. Custom certificate is not supported.
- If the API return code is not **2***XX*, the operation fails.

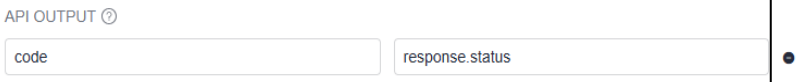
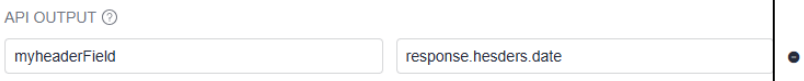
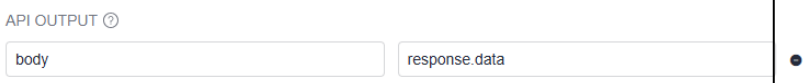
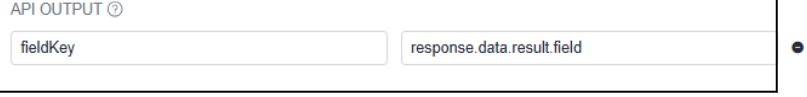
Configuration Method

Step 1 Add the **ThirdPartyAPI-v2** extension when you **orchestrate a pipeline**.

Figure 11-5 Adding the ThirdPartyAPI-v2 extension



Step 2 Set parameters as shown in the following table.

Parameter	Description
Name	Extension name. • Enter only letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces. • Enter at least one character.
Request method	Select **GET**, **POST**, **PATCH**, **PUT**, or **DELETE**.
API URL	API that can be called by the public network. Certificate verification is required.
Request header	Configure key-value pairs.
Request parameter	Configure key-value pairs.
Request body	If the request method is **POST**, **PATCH**, or **PUT**, enter the body in JSON format.
API output	The root object name is **response** which includes multiple outputs. • To obtain the specific request return code, and if the output key is **code**, refer to the following figure.  • If the returned request header contains the **date** field, refer to the following figure to obtain the **date** value.  • To output the entire body, refer to the following figure.  • If the returned JSON body is as follows and you need to obtain the value of **field**, refer to the following figure. `{ "result": { "field": "value", "field1": "value1" } }` 

Step 3 After the extension is executed, if you want to use the output, see **Pipeline Contexts**.

----End

12 Creating Service Endpoints

Scenario

A service endpoint is an extension of CodeArts. It enables CodeArts to connect to third-party services.

For example, when your CodeArts tasks need to obtain project source code from a third-party GitHub repository or need to run with Jenkins, you can create an endpoint to connect to each service.

The following table lists the endpoints supported by CodeArts.

Table 12-1 Service endpoints

Type	Scenario
Docker repository	Connect to a third-party Docker image repository. After the connection is successful, CodeArts Deploy can obtain Docker images from the repository.
Jenkins	Connect to a third-party Jenkins service. After the connection is successful, pipelines can call and execute the tasks in the Jenkins service.
Kubernetes	Connect to a Kubernetes cluster. After the connection is successful, you can deploy applications to the relevant Kubernetes cluster.
Nexus repository	Connect to a third-party private Maven repository. After the connection is successful, build tasks can obtain the file information of the repository.
Git repository	Connect to a third-party Git repository. After the connection is successful, CodeArts Pipeline and CodeArts Build can obtain the branch information of the repository.
GitHub	Connect to a GitHub account. After the connection is successful, CodeArts Pipeline and CodeArts Build can obtain the repository and branch information of the account.

Type	Scenario
IAM user	Delegate your AK/SK to an IAM user so that the user can obtain a token to perform tasks that require higher permissions.
CodeArts Repo HTTPS	Authorize CodeArts to download code, create branches, merge branches, and commit code in CodeArts Repo repositories. Currently, it is used for change-triggered pipelines and related extensions.
GitLab	Connect to a GitLab repository. After the connection is successful, CodeArts Pipeline and CodeArts Build can obtain the branch information of the repository.
Bitbucket	Connect to a Bitbucket account. After the connection is successful, CodeArts Pipeline and CodeArts Build can obtain the repository and branch information of the account.
AGC	Connect to AppGallery Connect (AGC) APIs. After the connection is successful, pipelines can use the service.

Prerequisites

- You must have the **DevMarket** > **Endpoint** > **create** permission to create service endpoints. For details, see **How Do I Check and Obtain Required Project Permissions?**
- The third-party service to connect can be accessed from the public network without restrictions.
- Jenkins, Nexus repository, and GitHub service endpoints cannot be created in the **LA-Santiago** region.
- Jenkins and Nexus repository service endpoints cannot be created in the **TR-Istanbul** area.
- Bitbucket service endpoints can be created only in the **AP-Singapore** region.

Creating a Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region where you have enabled CodeArts.
2. Click **Go to Workspace**.

   If your account uses the old billing mode (see **Old Billing Modes**), click **Access Service**.

Step 2 Click the target project name to go to the project.

Step 3 In the navigation pane, choose **Settings** > **General** > **Service Endpoints**.

Step 4 Click **Create Endpoint** and select an endpoint type from the drop-down list.

Step 5 In the displayed dialog box, configure the service endpoint.

The new endpoint is displayed.

----End

The tables below describe the parameters for configuring different types of service endpoints.

Docker Repository

Table 12-2 Creating a Docker repository service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Repository Address	The address of the Docker repository to connect. HTTP and HTTPS addresses are supported.
Username	The username of this Docker repository.
Password	The password of this Docker repository.

Jenkins

Table 12-3 Creating a Jenkins service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Server URL	The address of the Jenkins service to connect. The address can be in format "**http://***ip*:*Port*" or "**https://***ip*:*Port*".
Username	The username of this Jenkins service.
Password	The password of this Jenkins service.

Kubernetes

Table 12-4 Creating a Kubernetes service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Kubernetes URL	The server address of the cluster to connect. Obtain it by searching for **server** in the cluster configuration file **kubeconfig.json**.
Kubeconfig	The configuration of this cluster. You can enter all the content of the **kubeconfig.json** file.

Nexus Repository

Table 12-5 Creating a Nexus repository service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Repository URL	The address of the Nexus repository to connect. HTTP and HTTPS addresses are supported.
Username	The username of this Nexus repository.
Password	The password of this Nexus repository.

Git Repository

Table 12-6 Creating a Git service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Git Repository URL	The HTTPS address of the Git repository to connect. For example, **https ://*.*.*.*/ user/repo.git**.
Username	The username of this Git repository. This parameter is optional. Configure it as needed.

Parameter	Description
Password or Access Token	The password or access token of this Git repository. This parameter is optional. Configure it as needed.

GitHub

Table 12-7 Creating a GitHub service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Authentication Mode	Two authentication modes are supported: • **OAuth**: After clicking **Authorize and Confirm**, log in to GitHub for manual authorization. • **Access token**: Enter your access token obtained in GitHub. For details, visit the **GitHub official website**.

IAM User

Table 12-8 Creating an IAM user service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Access Key Id	The AK of the IAM user to connect. Obtain it from the **My Credentials** page. For details, see **Access Keys**.
Secret Access Key	The SK of the IAM user to connect. Obtain it from the **My Credentials** page. For details, see **Access Keys**.

CodeArts Repo HTTPS

Table 12-9 Creating a CodeArts Repo HTTPS service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.

Parameter	Description
CodeArts Repo URL	The HTTPS address of the CodeArts Repo repository to connect. Go to the target repository, and click **Clone/Download**. Click **Clone with HTTPS**, and obtain the repository address.
Username	The HTTPS username of this CodeArts Repo repository. Format: *Tenant name*/*IAM username*. Click the username  on the top navigation bar and choose **This Account Settings**. Obtain the username on the **Repo** > **HTTPS Password** page.
Password	The HTTPS password of this CodeArts Repo repository. Enter a maximum of 300 characters. Click the username  on the top navigation bar and choose **This Account Settings**. Obtain the password on the **Repo** > **HTTPS Password** page.

GitLab

Table 12-10 Creating a GitLab repository service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
GitLab URL	The HTTPS address of the GitLab repository to connect.
Username	The username of this GitLab repository. This parameter is optional. Configure it as needed.
Access Token	The access token of the GitLab repository. For details about how to obtain the access token, visit the **GitLab official website**. This parameter is optional. Configure it as needed.

Bitbucket

Table 12-11 Creating a Bitbucket repository service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
Username	Bitbucket username. To obtain the username, log in to Bitbucket, and obtain the value of **Username** on the **Account settings** page.
Password	Bitbucket password. To obtain the password, log in to Bitbucket, and create a password on the **App passwords** page, with the **Read** permission selected under both **Account** and **Projects**.

AGC

Table 12-12 Creating an AGC service endpoint

Parameter	Description
Service Endpoint Name	The name of the service endpoint. Enter a maximum of 256 characters. Letters, digits, hyphens (-), underscores (_), periods (.), and spaces are supported.
client_id	The ID of the AppGallery Connect (AGC) API client to connect. For details about how to obtain the client ID, visit the **AppGallery Connect official website**.
client_secret	The secret of the AGC API client to connect. For details about how to obtain the client secret, visit the **AppGallery Connect official website**.

Managing a Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region where you have enabled CodeArts.
2. Click **Go to Workspace**.

   If your account uses the old billing mode (see **Old Billing Modes**), click **Access Service**.

Step 2 Click the target project name to go to the project.

Step 3 In the navigation pane, choose **Settings** > **General** > **Service Endpoints**.

Step 4 Locate the service endpoint you want to manage, and perform the operations listed in the table below as needed.

Table 12-13 Managing a service endpoint

Operation	Description
Edit	Click **Edit**. In the displayed dialog box, edit the parameters and click **Save**. The modified information is displayed.
Delete	**WARNING** The deletion cannot be undone. Exercise caution when performing this operation. Click **Delete**. In the displayed dialog box, click **OK**. The deleted service endpoint is no longer displayed in the list.

----End

13 Checking Audit Logs

Cloud Trace Service (CTS) records operations on CodeArts Pipeline for query, audit, and backtrack.

After you enable CTS, the system starts recording operations on CodeArts Pipeline. You can view the operation records of the last seven days on the management console.

CodeArts Pipeline Operations Recorded by CTS

Table 13-1 CodeArts Pipeline operations recorded by CTS

Operation	Resource Type	Event Name
Executing a pipeline	pipeline	run
Editing a pipeline	pipeline	update
Creating a pipeline	pipeline	create
Deleting a pipeline	pipeline	delete
Stopping a pipeline	pipeline	stop

Related Document

For details about how to query CodeArts Pipeline operations on the CTS console, see **Querying Real-Time Traces**.

14 Reference

14.1 Pipeline Contexts

14.1.1 Pipeline Contexts

About Contexts

Pipeline contexts store and transfer pipeline information during pipeline running. They store information such as pipeline instances, variables, and tasks. Each context is an object that contains various attributes. You can use pipeline contexts to transfer information among jobs to streamline a pipeline. The following table lists pipeline contexts.

Table 14-1 Pipeline contexts

Context	Type	Description
pipeline	object	Information about the pipeline run.
sources	object	Information about the pipeline sources in each pipeline run.
env	object	Information about the custom parameters in each pipeline run.
jobs	object	Information about jobs that have been executed in each pipeline run.

Context Reference Format

```
${{ <context>.<attribute_name> }}
```

- *context*: pipeline context.
- *attribute_name*: attribute.

Contexts Attributes

Table 14-2 Context attributes

Context	Attribute	Type	Description	Example

pipeline context	pipeline	object	Information about the pipeline run. This object contains the following attributes: **project_id**, **pipeline_id**, **run_number**, **timestamp**, **trigger_type**, and **run_id**.	• Content example
				The following example shows the pipeline context information contained in a manually executed pipeline.
				`{ "project_id": "6428c2e2b4b64affa14ec80896695c49", "pipeline_id": "f9981060660249a3856f46c2c402f244", "run_number": "168", "timestamp": "20231016000004", "trigger_type": "Manual", "run_id": "c2f507f93510459190b543e47f6c9bec" }`
				• Usage example
				To obtain the trigger type of the current pipeline, you can use the following syntax:
				`${{ pipeline.trigger_type }}`

	pipeline.project_id	string	ID of the project to which the current pipeline belongs. This string is the same as the **predefined parameter** **PROJECT_ID**.	
	pipeline.pipeline_id	string	Current pipeline ID. This string is the same as the **predefined parameter** **PIPELINE_ID**.	
	pipeline.run_number	string	Pipeline execution number. This string is the same as the **predefined parameter** **PIPELINE_NUMBER**.	
	pipeline.timestamp	string	Pipeline execution timestamp. This string is the same as the **predefined parameter** **TIMESTAMP**. The format is *yyyyMMddHHmmss*. For example, **20211222124301**.	

Context	Attribute	Type	Description	Example
	pipeline.trigger_type	string	Pipeline trigger type. This string is the same as the **predefined parameter** **PIPELINE_TRIGGER_TYPE**.	
	pipeline.run_id	string	Pipeline execution ID. This string is the same as the **predefined parameter** **PIPELINE_RUN_ID**.	

sources context	sources	object	Information about the pipeline sources in each pipeline run. This object contains the following attributes: **alias**, **repo_name**, **commit_id**, **commit_id_short**, **commit_message**, **repo_url**, **repo_type**, **repo_name**, **ssh_repo_url**, **tag**, **merge_id**, **source_branch**, and **target_branch**.	• Content example
				The following example shows the sources context information contained in a manually executed pipeline with a single code source. The alias of pipeline source is **my_repo**.
				{
				"my_repo": {
				"commit_id": "dedb73bb9abfdaab7d810f2616bae9d2b6632ecc",
				"commit_id_short": "dedb73bb",
				"commit_message": "maven0529 update pipeline0615.yml",
				"repo_url": "https://example.com/clsyz00001/maven0529.git",
				"repo_type": "codehub",
				"repo_name": "maven0529",
				"ssh_repo_url": "git@example.com:clsyz00001/maven0529.git",
				"target_branch": "master"
				}
				}
				• Usage example
				To obtain the running branch of the pipeline, you can use the following syntax:
				${{ sources.my_repo.target_branch }}

	sources.<alias>	object	Information about the pipeline source which has an alias.	
	sources.<repo_name>	object	Information about the pipeline source which does not have an alias but only a repository name. It contains the same information as that in **sources.***<alias>*.	

Context	Attribute	Type	Description	Example
	sources.<alias>.commit_id	string	The last commit ID before execution. This string is the same as the **predefined parameter** **COMMIT_ID**.	
	sources.<alias>.commit_id_short	string	The first 8 characters of the last commit ID before execution. This string is the same as the **predefined parameter** **COMMIT_ID_SHORT**.	
	sources.<alias>.commit_message	string	The commit information from the last code commit before the pipeline execution.	
	sources.<alias>.repo_url	string	Code repository address (HTTPS). This string is the same as the **predefined parameter** **REPO_URL**.	
	sources.<alias>.repo_type	string	Type of the code repository. For example, **codehub**, **gitlab**, **github**, **gitee**, and **general_git**.	
	sources.<alias>.repo_name	string	Name of the code repository.	
	sources.<alias>.ssh_repo_url	string	Code repository address (SSH).	
	sources.<alias>.tag	string	Tag name when the tag is triggered.	

Context	Attribute	Type	Description	Example
	sources.<alias>.merge_id	string	Merge request ID when the merge request is triggered.	
	sources.<alias>.source_branch	string	Source branch name when the merge request is triggered.	
	sources.<alias>.target_branch	string	If the merge request is triggered, this string indicates the name of the target branch. Otherwise, this string indicates the name of the running branch.	

env context	name	string	Name of a custom parameter.	• Content example
				The following example shows the env context information in a run, which includes two custom parameters.
				{
				"var_1": "val1",
				"var_2": "val2"
				}
				• Usage example
				To obtain the value of the custom parameter **var_1**, you can use the following syntax:
				${{ env.var_1 }}

	value	string	Value of a custom parameter.	

Context	Attribute	Type	Description	Example

jobs context	jobs	object	Information about jobs in a pipeline. This object contains the following attributes: **job_id**, **status**, **outputs**, **output_name**, **metrics**, and **metric_name**.	• Content example
				The following example shows the jobs context information in a run. There are two successfully executed jobs. The output of the **check_job** job is two metrics, and the output of the **demo_job** job is two general outputs.
				`{ "check_job": { "status": "COMPLETED", "metrics": { "critical": "0", "major": "0" } }, "demo_job": { "status": "COMPLETED", "outputs": { "output1": "val1", "output2": "val2" } } }`
				• Usage example
				To obtain the value of **output1** of **demo_job**, you can use the following syntax:
				`${{ jobs.demo_job.outputs.output1 }}`

	jobs.<job_id>	object	Information about the job with a specified ID.	
	jobs.<job_id>.status	string	Job execution result. The value can be **INIT**, **QUEUED**, **RUNNING**, **CANCELED**, **COMPLETED**, **FAILED**, **PAUSED**, **IGNORED**, **SUSPEND**, or **UNSELECTED**.	
	jobs.<job_id>.outputs	object	The running value, as a key-value pair.	
	jobs.<job_id>.outputs.<output_name>	string	The running value name.	
	jobs.<job_id>.metrics	object	The running metrics of a job. For example, the number of code check issues and the test pass rate.	
	jobs.<job_id>.metrics.<metric_name>	string	The running metric name of a job.	

Related Information

The following are context scenarios:

- **Configuring Expressions**.
- **Obtaining Artifact Information Using the Pipeline Context**.
- **Creating a Repository Tag Using the Pipeline Contexts**.

14.1.2 Configuring Expressions

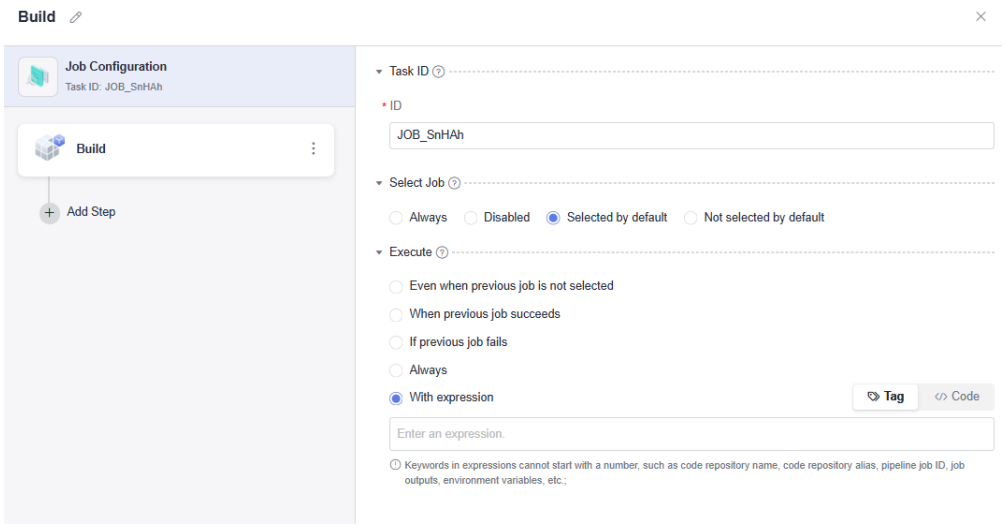
You can reference pipeline contexts with expressions to specify the execution condition of a job. An expression can be any combination of **contexts**, operators, functions, or literals. Contexts can be accessed programmatically with expressions, so information such as pipeline runs, variables, and jobs can be transferred across pipelines.

Step 1 **Create a pipeline**.

Step 2 Add stage jobs or edit existing jobs.

Step 3 Click **Job Configuration** and set **Execute** to **With expression** to configure the expression for job execution. For a new stage and job, add an extension first and then click **Job Configuration**.

Figure 14-1 Expressions



Example:

The following expression shows that a job runs only when the running branch of the specified code source is **master**.

```
${{ sources.my_repo.target_branch == 'master' }}
```

----End

References

- **Operator**

  The following table lists the operators that can be used in expressions.

Table 14-3 Operators

Operator	Description
.	Attribute reference. For example, the **${{ pipeline.trigger_type }}** expression can be used to obtain the trigger type.
!	False. For example, the **${{ !startsWith(sources.my_repo.target_branch, 'release') }}** can be used to check whether the branch of the pipeline's code source does not start with "release".
==	Equal. For example, the **${{ pipeline.trigger_type == 'Manual' }}** expression can be used to check whether a pipeline is triggered manually.
!=	Not equal. For example, the **${{ pipeline.trigger_type != 'Manual' }}** expression can be used to check whether a pipeline is not triggered manually.
&&	And. For example, the **${{ pipeline.trigger_type == 'Manual' && sources.my_repo.target_branch == 'master' }}** expression can be used to check whether a pipeline is triggered manually and the branch of the pipeline code source is **master**.
||	Or. For example, the **${{ pipeline.trigger_type == 'Manual' || sources.my_repo.target_branch == 'master' }}** expression can be used to check whether a pipeline is triggered manually or the branch of the pipeline code source is **master**.

- **Function**

  The following table lists the built-in functions that can be used in expressions.

Table 14-4 Built-in functions

Function	Description
contains	• Format contains(search, item) • Description If *search* contains *item*, this function returns **true**. – If *search* is an array and *item* is an element in the *array*, this function returns **true**. – If *search* is a string and *item* is a substring of *search*, the function returns **true**. • Example **contains('abc', 'bc')** returns **true**.

Function	Description
startsWith	• Format startsWith(searchString, searchValue) • Description If *searchString* starts with *searchValue*, this function returns **true**. • Example **startsWith('abc', 'ab')** returns **true**.
endsWith	• Format endsWith(searchString, searchValue) • Description If *searchString* ends with *searchValue*, this function returns **true**. • Example **endsWith('abc', 'bc')** returns **true**.
format	• Format format(string, replaceValue0, replaceValue1, ..., replaceValueN) • Description Replaces the value in *string* with the value of *replaceValueN*. • Example **format('Hello {0} {1}', 'a', 'b')** returns **Hello a b**.
substring	• Format substring(string, beginIndex, endIndex) • Description Returns the characters in *string* from beginIndex to endIndex-1. If endIndex is not configured, the characters in *string* from beginIndex to the end are returned. • Example **substring('202412091101', 0, 8)** returns **20241209**. **substring ('202412091101', 8)** returns **1101**.
replace	• Format replace(string, target, replacement) • Description Replaces each character same as *target* in *string* with *replacement*. • Example **replace('hello a', 'a', 'b')** returns **hello b**.

Function	Description
completed	• Format completed(job1, job2, ..., jobN) • Description Returns **true** when the dependent job (no job specified) or the specified job is successfully executed. For example, **completed()** returns **true** when the dependent job is completed. Or, you can set **Execute** to **When previous job succeeds** to achieve the same purpose. • Example **completed ('job1', 'job2')** returns **true** when both **job1** and **job2** are successfully executed.
always	• Format always() • Description Executes the current job regardless of the status of the executed dependent job. You can set **Execute** to **Always** to achieve the same purpose.
canceled	• Format canceled() • Description Executes the current job when the dependent job is stopped.
failed	• Format failed(job1, job2, ..., jobN) • Description Returns **true** when the dependent job (no job specified) or the specified job fails. For example, **failed()** returns **true** when the dependent job fails. You can set **Execute** to **If previous job fails** to achieve the same purpose. • Example **failed('job1', 'job2')** returns **true** when either **job1** or **job2** fails to be executed.

Function	Description
Object filter	You can use the * syntax to apply a filter and select matching items in a collection. The following is the context of a job execution. `{ "check_job": { "status": "COMPLETED", "metrics": { "critical": "0", "major": "0" } }, "demo_job": { "status": "FAILED" } }` • **jobs.*.status** indicates the status of all jobs. Therefore, ['COMPLETED', 'FAILED'] is returned. • Filters can be used together with the **contains** function. For example, **contains(jobs.*.status, 'FAILED')** will return **true** because **jobs.*.status** contains **FAILED**.

14.1.3 Obtaining Artifact Information Using the Pipeline Context

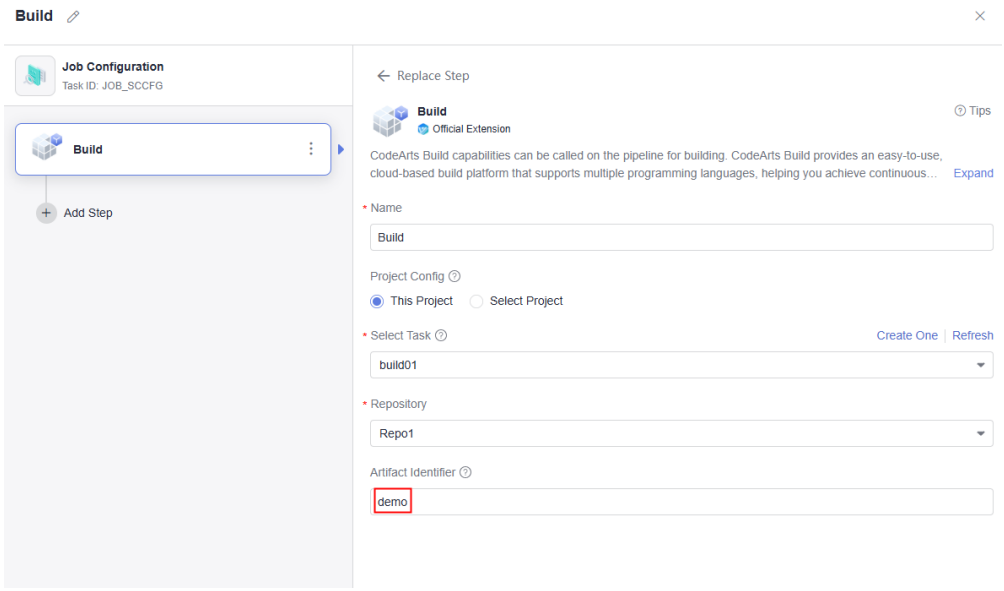
You can reference pipeline context during job configuration to obtain desired information. The following example uses the **Build** extension to generate an artifact and retrieves the artifact information by referencing context in the **ExecuteShellCommand** job.

Step 1 Create a pipeline.

Step 2 Add the **Build** extension to **Stage_1**, obtain the task ID as shown in Figure 14-2, and set the artifact identifier to **demo** as shown in Figure 14-3.

Figure 14-2 Obtaining the task ID

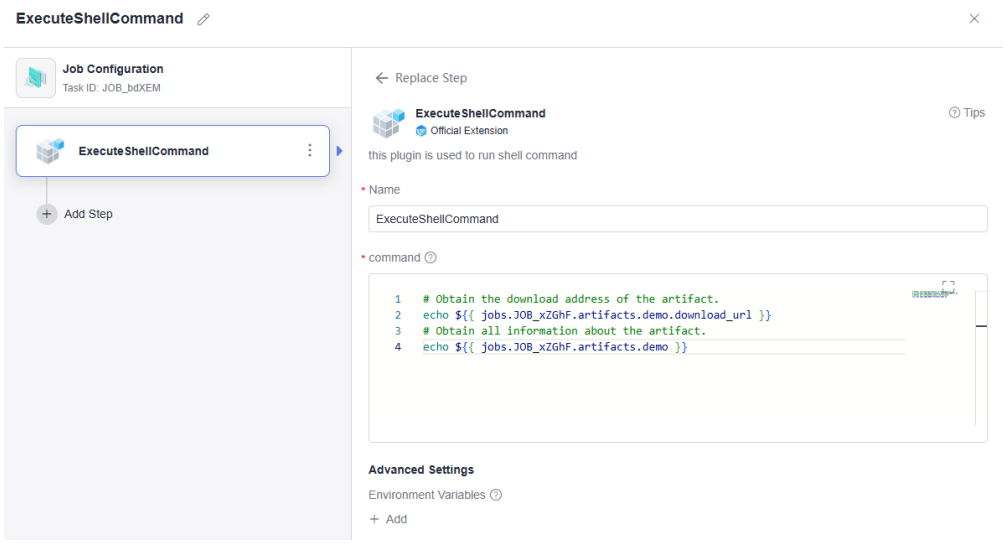
Figure 14-3 Adding the Build extension



Step 3 Add the **ExecuteShellCommand** extension to **Stage_2**. Run the following commands to obtain the artifact information:

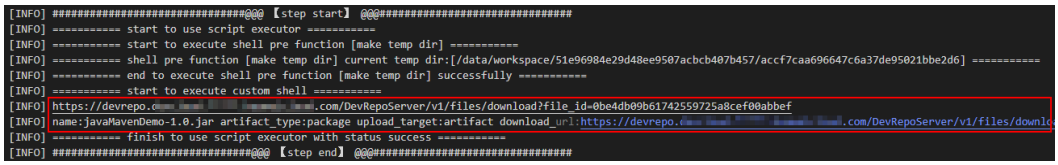
```
# Obtain the download address of the artifact.
echo ${{ jobs.JOB_xZGhF.artifacts.demo.download_url }}
# Obtain all information about the artifact.
echo ${{ jobs.JOB_xZGhF.artifacts.demo }}
```

Figure 14-4 Adding the ExecuteShellCommand extension



Step 4 Execute the pipeline. After the execution is successful, check the printed artifact information in the log.

Figure 14-5 Artifact information



----End

14.2 YAML Syntax

14.2.1 on

Use **on** to specify events that automatically trigger a pipeline. These events include changing branches, files, and tags, and executing scheduled tasks.

on.<event_name>.types

Use **on.<event_name>.types** to specify the type of a merge request (MR) event that can trigger a pipeline.

Table 14-5 Merge request types

MR Type	Description
opened	An MR is created.
synchronize	The source branch is updated.
closed	An MR is merged.
reopened	An MR is reopened.

If you do not define a type, the pipeline will be triggered when an MR is opened, synchronized, or reopened. The following example shows the syntax for triggering a pipeline when an MR is closed.

```
on:
  pull_request:
    types:
      - closed
```

on.<pull_request>.<branches|branches-ignore>

Use the **pull_request** event to define a pipeline to run for MRs that target specific branches. Do not use **branches** and **branches-ignore** for a **pull_request** event at the same time.

- **branches**: Use **branches** to include specific branches, or both include and exclude specific branches.
- **branches-ignore**: Use **branches-ignore** to exclude specific branches.

branches and **branches-ignore** support the use of glob patterns to match branch names.

Table 14-6 Matching characters

Character	Description	Example
*	Matches zero or more characters, except the slash (/).	**dev***: matches **dev** and **develop**, but cannot match **dev/test**.
?	Matches one character.	**dev?**: matches **dev1** and **dev2**, but cannot match **dev12**.
**	Matches zero or more characters.	**dev****: matches **dev**, **develop**, and **dev/test**.
[]	Matches one character listed in or within the specified range in the brackets. Ranges only include **a-z**, **A-Z**, and **0-9**.	• **v[a-z].[0-9]***: matches **va.1**, **vb.111**, and so on. • **v[01]**: matches **v0** and **v1**.
{}	Matches strings listed in the parentheses.	**{branch1,branch2}**: matches **branch1** and **branch2**.
!	It is at the start of a string, indicating that the subsequent characters are not matched.	**!develop**: matches all characters except **develop**.

Example

- **Including branches**

```
on:
  pull_request:
    branches:
      - master
      - 'release**'
```

  This example indicates that the pipeline would run when there is a **pull_request** event that targets the following branches:

  – a branch named **master**, or;

  – a branch whose name starts with **release**, for example, **release**, **release-1.0.0**, and **release/1.0.0**

- **Excluding branches**

```
on:
  pull_request:
    branches-ignore:
      - test
      - 'dev**'
```

  This example indicates that the pipeline would run when there is a **pull_request** event, unless the event targets the following branches:

  – a branch named **test**, or;

  – a branch whose name starts with **dev**, for example, **dev**, **develop-1.0.0**, and **develop/1.0.0**

- **Both including and excluding branches**

  Use **branches** and an exclamation mark (!) to both include and exclude specific branches.

```
on:
  pull_request:
    branches:
      - 'release**'
      - '!release/v1**'
```

  This example indicates that the pipeline would run when there is a **pull_request** event, and the event's targeting branch names meet all of the following conditions:

  – Start with **release**, for example, **release**, **release-1.0.0**, and **release/1.0.0**
  – Do not start with **release/v1/****, for example, **release/v1**, **release/v1.0**, and **release/v1/1.0**

on.<push>.<branches|branches-ignore|tags|tags-ignore>

Use the **push** event to define a pipeline to run on specific branches or tags. Do not use **branches**, **branches-ignore**, **tags**, and **tags-ignore** for a **push** event at the same time.

- **branches**: Use **branches** to include specific branches, or both include and exclude specific branches.
- **branches-ignore**: Use **branches-ignore** to exclude specific branches.
- **tags**: Use **tags** to include specific tags, or both include and exclude specific tags.
- **tags-ignore**: Use **tags-ignore** to exclude specific tags.

Example

- **Including branches/tags**

```
on:
  push:
    branches:
      - master
      - 'release**'
    tags:
      - v1
      - 'v2.*'
```

  This example indicates that the pipeline would run when there is a **push** event that targets the following branches and tags:

  – a branch named master, or;
  – a branch whose name starts with **release**, for example, **release**, **release-1.0.0**, and **release/1.0.0**
  – a tag named v1
  – a tag whose name starts with **v2.**, for example, **v2.1** and **v2.1.1**

- **Excluding branches/tags**

```
on:
  push:
    branches-ignore:
      - test
      - 'dev**'
    tags-ignore:
      - v1
      - 'v2.*'
```


This example indicates that the pipeline would run when there is a **push** event, unless the event targets the following branches and tags:

- a branch named **test**, or;
- a branch whose name starts with **dev**, for example, **dev**, **develop-1.0.0**, and **develop/1.0.0**.
- a tag named v1
- a tag whose name starts with **v2.**, for example, **v2.1** and **v2.1.1**

- **Both including and excluding branches/tags**

  Use **branches**, **tags**, and an exclamation mark (!) to both include and exclude specific branches and tags.

```
on:
  push:
    branches:
      - 'release**'
      - '!release/v1**'
    tags:
      - 'v1**'
      - '!v1.1'
```

  This example indicates that the pipeline would run when there is a **push** event, and the event's targeting branches and tags meet all of the following conditions:

  - a branch whose name starts with **release**, for example, **release**, **release-1.0.0**, and **release/1.0.0**
  - a branch whose name does not start with **release/v1/****, for example, **release/v1**, **release/v1.0**, and **release/v1/1.0**
  - a tag whose name starts with **v1**, for example, **v1** and **v1.2**
  - a tag whose name is not **v1.1**

on.<push|pull_request>.<paths|paths-ignore>

Use the **push** and **pull_request** events to define a pipeline to run when specified files change. Do not use **paths** and **paths-ignore** for **push** and **pull_request** events at the same time.

- **paths**: Use **paths** to include specific files or both include and exclude specific files.
- **paths-ignore**: Use **paths-ignore** to exclude specific files.

Example

- **Including files**

```
on:
  push:
    paths:
      - '**.java'
```

  This example indicates that the pipeline would run when you push a **.java** file.

- **Excluding files**

```
on:
  push:
    paths-ignore:
      - 'docs/**'
```

  This example indicates that the pipeline would run when there is a **push** event, unless all pushed files are in the **docs** directory.

- **Both including and excluding files**

  Use **paths** and an exclamation mark (!) to both include and exclude specific files.

  ```
  on:
    push:
      paths:
        - 'src/**'
        - '!src/docs/**'
  ```

  This example indicates that the pipeline would run when the changed files are in the **src** directory or its subdirectories, but not in the **src/docs** directory.

on.schedule

Use **on.schedule** to schedule a pipeline to run at a specified UTC time by defining a cron expression. The pipeline will run based on the lasted scheduled task information. To update the scheduled task, edit YAML and save it. The default branch of the pipeline source is used.

```
on:
  schedule:
    - cron:  '0 0 12 * * ?'
    - cron:  '0 0 20 * * ?'
```

The above example indicates that the pipeline would run at 12:00 and 20:00 (UTC time) every day.

14.2.2 env

Use **env** to define environment variables as key-value pairs. Environment variables can be referenced in any job within the pipeline.

Example

```
env:
  version: 1.0.0
```

You can use **${version}** or **${{ env.version }}** to reference the variable in any job.

${{ env.version }} is recommended, as shown in the example. For more information about the expression syntax, see **Configuring Expressions**.

14.2.3 jobs

A pipeline can consist of multiple jobs.

jobs.<job_id>

Use **jobs.<job_id>** to give jobs an ID, unique within a pipeline. **<job_id>** can contain letters, digits, hyphens (-), and underscores (_), with a maximum of 32 characters.

```
jobs:
  job1:
    name: first job
  job2:
    name: second job
```

This example indicates that there are two jobs, whose unique identifiers are **job1** and **job2**.

jobs.<job_id>.name

Use **jobs.<job_id>.name** to define a job name. The name is displayed on the CodeArts Pipeline UI.

```
jobs:
  job1:
    name: first job
  job2:
    name: second job
```

This example indicates that the names of **job1** and **job2** are **first job** and **second job**.

jobs.<job_id>.needs

Use **jobs.<job_id>.needs** to specify which job must succeed before you run a new one.

```
jobs:
  job1:
    name: first job
  job2:
    needs: [ job1 ]
    name: second job
```

This example indicates that **job2** will run only after **job1** is complete successfully.

jobs.<job_id>.if

Use **jobs.<job_id>.if** to define the job running condition. For details about the conditional expression, see **Configuring Expressions**.

```
jobs:
  job1:
    name: first job
  job2:
    needs: [ job1 ]
    if: ${{ always() }}
    name: second job
```

This example indicates that **job2** will always run after **job1** is complete, regardless of whether it is successful.

jobs.<job_id>.steps

A job can consist of multiple steps. Each step can run an extension.

jobs.<job_id>.steps<*>.name

Use **jobs.<job_id>.steps<*>.name** to define a job name, which is displayed on the CodeArts Pipeline UI.

jobs.<job_id>.steps<*>.uses

Use **jobs.<job_id>.steps<*>.uses** to specify the extension used in a step.

```
jobs:
  demo_job:
    name: simple demo job
    steps:
```



```
  - name: simple custom step
    uses: custom_plugin@1.0.0
```

This example indicates that a step uses an extension whose name is **custom_plugin** and version is **1.0.0**.

YAML Syntax for the Pipeline Official Extensions

- **Build**

  The following example calls the **Build** extension to use CodeArts Build capabilities.

  ```
  uses: CodeArtsBuild
  with:
    jobId: 878b4d13cb284d9e8f33f988a902f57c
    artifactIdentifier: my_pkg
    customParam: value
  ```

  - **jobId**: ID of the build task. To obtain the ID, copy the 32 digits and letters at the end of the browser URL on the build task details page.
  - **artifactIdentifier**: Build artifact identifier.
  - **customParam**: Parameter value defined in the build task. There may be zero to multiple values.

- **TestPlan**

  The following example calls the **TestPlan** extension to use CodeArts TestPlan capabilities.

  ```
  uses: CodeArtsTestPlan
  with:
    jobId: vb180000vnrgoeib
    environmentModel: 1
    environmentId: 7c2eff2377584811b7981674900158e8
  ```

  - **jobId**: ID of the API test task.
  - **environmentModel**: Parameter source. The value **0** indicates that new parameters will be used, and the value **1** indicates that the global parameters of the selected environment will be used.
  - **environmentId**: Environment ID when **environmentModel** is set to **1**.

- **Deploy**

  The following example calls the **Deploy** extension to use CodeArts Deploy capabilities.

  ```
  uses: CodeArtsDeploy
  with:
    jobId: 9c5a5cda6ffa4ab583380f5a014b2b31
    customParam: value
  ```

  - **jobId**: ID of the deployment task.
  - **customParam**: Parameter value defined in the deployment task. There may be zero to multiple values.

- **Check**

  The following example calls the **Check** extension to use CodeArts Check capabilities.

  ```
  uses: CodeArtsCheck
  with:
    jobId: 43885d46e13d4bf583d3a648e9b39d1e
    checkMode: full|push_inc_full||push_multi_inc_full
  ```

  - **jobId**: ID of a code check task.

  - **checkMode**: Check mode.
    - **full**: Checks all code.
    - **push_inc_full**: Checks all files changed in this code commit.
    - **push_multi_inc_full**: Checks all files changed between this code commit and the last successful code commit.

- **CreateTag**

  The following example calls the **CreateTag** extension to create and push a tag for code repositories.

```
uses: CreateTag
with:
  tagName: v1
```

  tagName: Tag name.

- **Subpipeline**

  The following example calls the **Subpipeline** extension to configure other pipelines in a project.

```
uses: SubPipeline
with:
  pipelineId: 80ea2d9ffba94c20b9a0a0be47d3a0d8
  branch: master
```

  - **pipelineId**: ID of the called pipeline.
  - **branch**: (Optional) Branch used for running the sub-pipeline.
    - The default branch of the sub-pipeline is used if this parameter is not set.
    - You can reference a parameter or context to define **branch**. For example, if you want to run the parent pipeline source, and the code source alias is **my_repo**, the reference format is **${{sources.my_repo.target_branch}}**.

- **JenkinsTask**

  The following example calls the **JenkinsTask** extension to configure a Jenkins task.

```
uses: Jenkins
with:
  endpoint: eac965b206e74e2b898a24a4375b6df6
  jobName: job
  params: '{ \"key\":\"value\" }'
  async: true|false
  description: description
```

  - **endpoint**: ID of the Jenkins endpoint.
  - **jobName**: Jenkins job name.
  - **params**: Parameters (in JSON format) transferred for starting the job.
  - **async**: Whether to execute the job asynchronously.
  - **description**: Execution description.

- **PipelineSuspension**

  The following example calls the **PipelineSuspension** extension to suspend the current pipeline.

```
uses: SuspendPipeline
```


- **DelayedExecution**

  The following example calls the **DelayedExecution** extension. It can pause pipeline for a period of time or until a specified time. You can manually resume or stop a pipeline, or delay the execution for a maximum of three times.

```
uses: Delay
with:
  timerType: delay|scheduled
  delayTime: 300
  scheduledTime: '00:00'
  timeZone: China Standard Time
```

  - **timerType**: Delay type. **delay** indicates pausing a pipeline for a period of time. **scheduled** indicates pausing a pipeline until a specified time.
  - **delayTime**: Time duration (in seconds) when **timerType** is set to **delay**.
  - **scheduledTime**: Exact time when **timerType** is set to **scheduled**.
  - **timeZone**: Time zone. Available values are listed in the following table.

Table 14-7 Time zone

Available Value	Time Zone
GMT Standard Time	GMT
South Africa Standard Time	GMT+02:00
SE Asia Standard Time	GMT+07:00
Singapore Standard Time	GMT+08:00
China Standard Time	GMT+08:00
Pacific SA Standard Time	GMT-04:00
E. South America Standard Time	GMT-03:00
Central Standard Time (Mexico)	GMT-06:00
Egypt Standard Time	GMT+02:00
Saudi Arabia Standard Time	GMT+03:00

- **ManualReview**

  The following example calls the **ManualReview** extension to create manual review tasks by assigning one person or one group.

```
uses: Checkpoint
with:
  mode: members|roles
  approvers: 05d8ca972f114765a8984795a8aa4d41
  roles: '3'
  checkStrategy: all|any
  timeout: 300
```



```
timeoutStrategy: reject|pass
comment: comment
```

- **mode**: Review mode. **members** indicates that the review is performed by member, and **roles** indicates that the review is performed by role.
- **approvers**: User IDs of the approvers when **mode** is set to **members**. Use commas (,) to separate multiple user IDs.
- **role**: Roles when the **mode** is set to **roles**. For details about the options, see **Table 14-8**. Use commas (,) to separate multiple roles.
- **checkStrategy**: Strategy used when the **mode** is set to **members**. **all** indicates that the application can be approved only by all users. **any** indicates that the application can be approved by any user.
- **timeout**: Review timeout, in seconds.
- **timeoutStrategy**: Strategy used when the review times out. **reject** indicates that the pipeline is stopped to run. **pass** indicates that the pipeline can continue to run.
- **comment**: Review description.

Table 14-8 Review roles

Role	YAML Identifier
Project manager	'3'
Developer	'4'
Test manager	'5'
Tester	'6'
Participant	'7'
Viewer	'8'
Operation manager	'9'
Product manager	'10'
System engineer	'11'
Committer	'12'

15 CodeArts Release User Guide

15.1 Overview

CodeArts Release is an E2E solution for version compatibility and automated rollout. It helps developers efficiently deliver and upgrade applications without affecting the existing production environment. If you want to deploy applications to containers, CodeArts Release is a good choice.

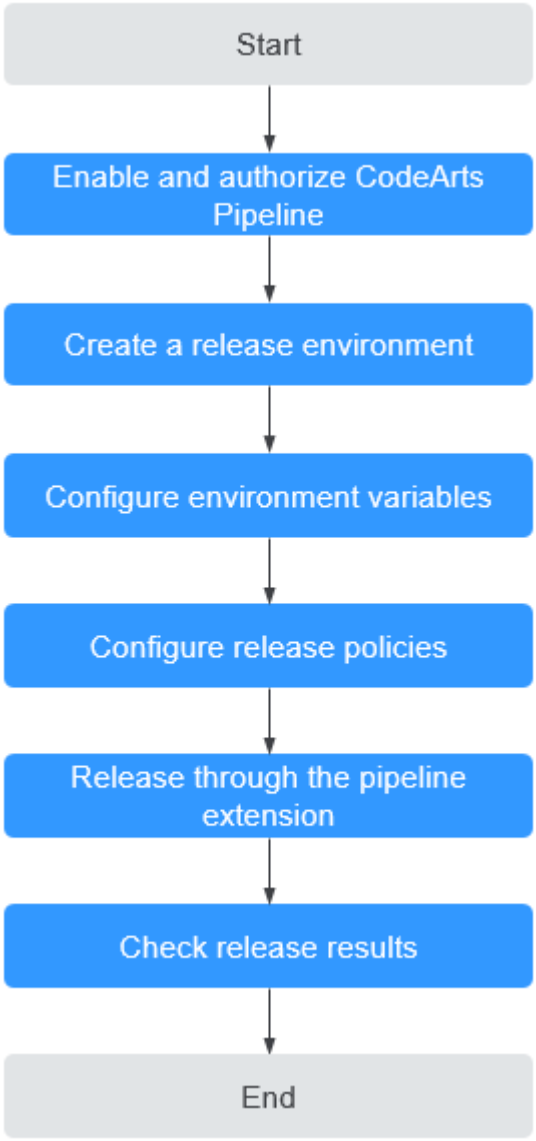
CodeArts Release has the following features:

- Provides solution-oriented baseline management capabilities, supports multi-dimensional version orchestration at the microservice, module, and product levels, and supports multi-cloud version mapping.
- Provides cloud-native microservice release management capabilities, supports gray orchestration and release of microservices, supports blue-green and canary gray release, and implements cross-cloud orchestration based on UCS.

Operation Process

The basic operation process of CodeArts Release includes: **Enable and authorize CodeArts Pipeline**, create a release environment, configure environment variables, configure release policies, perform release through the **CloudNativeRelease** extension, and check release results.

Figure 15-1 Operation process



15.2 Creating a Release Environment

Creating a Release Environment

Step 1 **Log in to the Huawei Cloud console**.

Step 2 Click ☰ in the upper left corner of the page, and choose **Developer Services** > **CodeArts** from the service list.

Step 3 Click **Access Service** to go to the CodeArts homepage.

Step 4 Click a project name to access the project.

Step 5 Choose **CICD** > **Release** to access the release environment list page.

Step 6 Click **Create Environment**. On the displayed page, set basic information. For details, see **Table 15-1**.

Table 15-1 Parameter description

Parameter	Description
Project	Project to which the environment belongs. The project cannot be changed.
Environment Name	Unique identifier of the release environment. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
Resource Type	**CCE**, **UCS**, and **K8s** are available. They support different deployment extensions. • **CCE**: a type of Kubernetes cluster encapsulated by Huawei Cloud. Select this type if you want to use Huawei Cloud resources. **Learn more**. • **UCS**: a type of Kubernetes cluster encapsulated by Huawei Cloud for multi-cloud deployment. Select this type if you want to deploy clusters on multiple clouds. **Learn more**. • **K8s**: the native Kubernetes cluster. Select this type if you want to use self-built clusters or third-party clusters.
Publish User	Options: **Current User** or **Other Users**. • **Current User**: Create an environment based on the cluster of the current user. • **Other Users**: Create an environment based on the clusters of other users. Obtain other users' cluster permissions through endpoints. For details, see *Creating Service Endpoints*.
Region	This parameter is required when you select **CCE** for **Resource Type**. Select the region where the environment is to be deployed.
Cluster	This parameter is required when you select **CCE** for **Resource Type**. Select the **purchased Kubernetes clusters** in Cloud Container Engine (CCE).
Association Type	This parameter is required when you select **UCS** for **Resource Type**. Associated UCS resources. Only **fleet** is supported.
Fleets	This parameter is required when you select **UCS** for **Resource Type**. **Select a fleet**.

Parameter	Description
Kubernetes Endpoint	This parameter is required when you select **K8s** for **Resource Type**. Select a created Kubernetes endpoint to access cluster resources with credential. For details, see *Creating Service Endpoints*.
Environment Level	Available environment types: development, test, pre-release, and production. For details about environment permissions, see **Project-level Permissions**.
Description	Enter the description of the environment with no more than 200 characters.

Step 7 After setting all parameters, click **OK**. The environment information page is displayed.

Table 15-2 Environment information

Parameter	Description
Resource Type	Resource types associated with the environment.
Publish User	Current user.
Service Endpoint	Service endpoint of CCE resources.
Cluster Region	Kubernetes cluster region applied in CCE.
Cluster ID	Kubernetes cluster ID applied in CCE.
Variable Version	Version number of the environment variable in the current environment.
Tag	Environment type.
Description	Description of the environment.

- Click **Edit** in the upper right corner of the page to edit the environment information.
- Switch tabs to **configure environment variables**, **configure release policies**, and **check deployment results**.

----End

15.3 Configuring an Environment Variable

You can use *${variable name}* to reference an environment variable when creating or editing a release policy, or use *{{variable name}}* to reference an environment variable in YAML files. Environment variables include:

- Custom variables: can be added as needed. Currently, only variables of the string type are supported.
- Default variables: system parameters, which cannot be deleted or modified.

Table 15-3 Default variables

Variable	Description
ARTIFACT	Artifact path. In the deployment YAML file, use {{ARTIFACT}} to reference the build artifacts.
TIMESTAMP	Timestamp when the extension is executed. For example, **20230401095436**.
PROJECT_ID	ID of the project to which the environment belongs.

Configuring an Environment Variable

Step 1 Access the release environment list page.

Step 2 Click an environment name. The **Environment Information** page is displayed.

Step 3 Click the **Environment Variable** tab.

Step 4 Click **Edit Variable** to add a variable and set parameters.

Table 15-4 Parameters for creating a custom variable

Parameter	Description
Variable	Variable name. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
Type	Only the string type is supported.
Value	The current value of a variable, or empty if you are adding a new variable. The value contains no more than 512 characters.
Change Value	Updated value of the environment variable.
Description	Variable description. It can contain a maximum of 128 characters.

Parameter	Description
Private Variable	If a parameter is private, the system encrypts the parameter for storage and decrypts the parameter for usage. Private parameters will not be displayed in run logs.

- Click in the **Operation** column to delete a variable.
- Click Add to add a variable.

Step 5 After setting all parameters, click **Save**. The **Save Changes** dialog box is displayed.

Step 6 Confirm the variable information, enter the remarks, and click **OK**.

You can click the **Versions** tab to check variable versions.

- Click a version name to view the variable details.
- Click in the **Operation** column to compare the current version with a specified version.

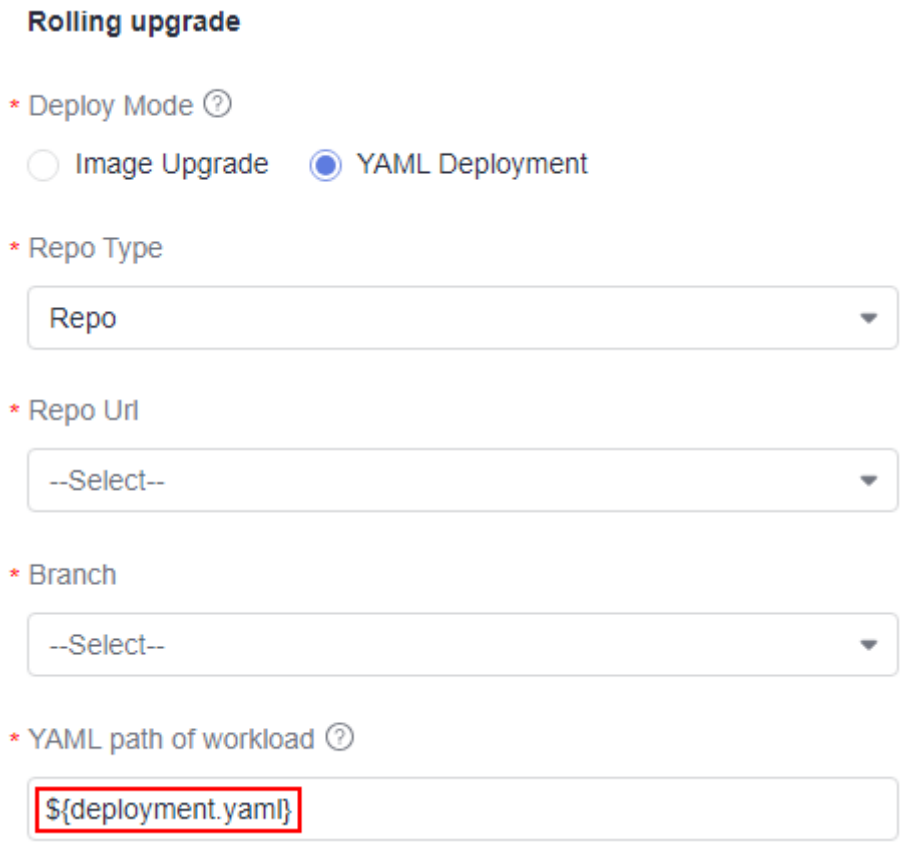
----End

Using an Environment Variable

You can use environment variables in the following scenarios:

- When **configuring a release policy**, you can use *${variable name}* to reference an environment variable in the YAML path, for example, the workload YAML path in the rolling upgrade task.

Figure 15-2 Referencing an environment variable



- Use *{{variable name}}* to reference an environment variable in the YAML configuration file associated with the release policy.

Figure 15-3 Referencing an environment variable

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rolling-test
  labels:
    run: rolling-test
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: rolling-test
  template:
    metadata:
      labels:
        run: rolling-test
    spec:
      containers:
        - name: main
          image: {{ARTIFACT}}
          ports:
            - containerPort: 8080
          env:
            - name: TIMESTAMP
              value: {{TIMESTAMP}}
            - name: PROJECT_ID
              value: {{PROJECT_ID}}
            - name: COMPONENT_ID
              value: {{COMPONENT_ID}}
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
```

15.4 Configuring an Environment Release Policy

Creating a Policy

You can add atomic extensions and edit release policies based on the preset **RollingUpgrade**, **GrayscaleUpgrade**, or **EmptyYamlTemplate** templates.

Step 1 Access the environment list page.

Step 2 Click an environment name. The Environment Information page is displayed.

Step 3 On the displayed page, click the **Release Policy** tab.

Step 4 Click + next to **Custom Policies**, on the displayed dialog box, select a policy template and click **OK**.

Step 5 Orchestrate extensions based on the selected template.

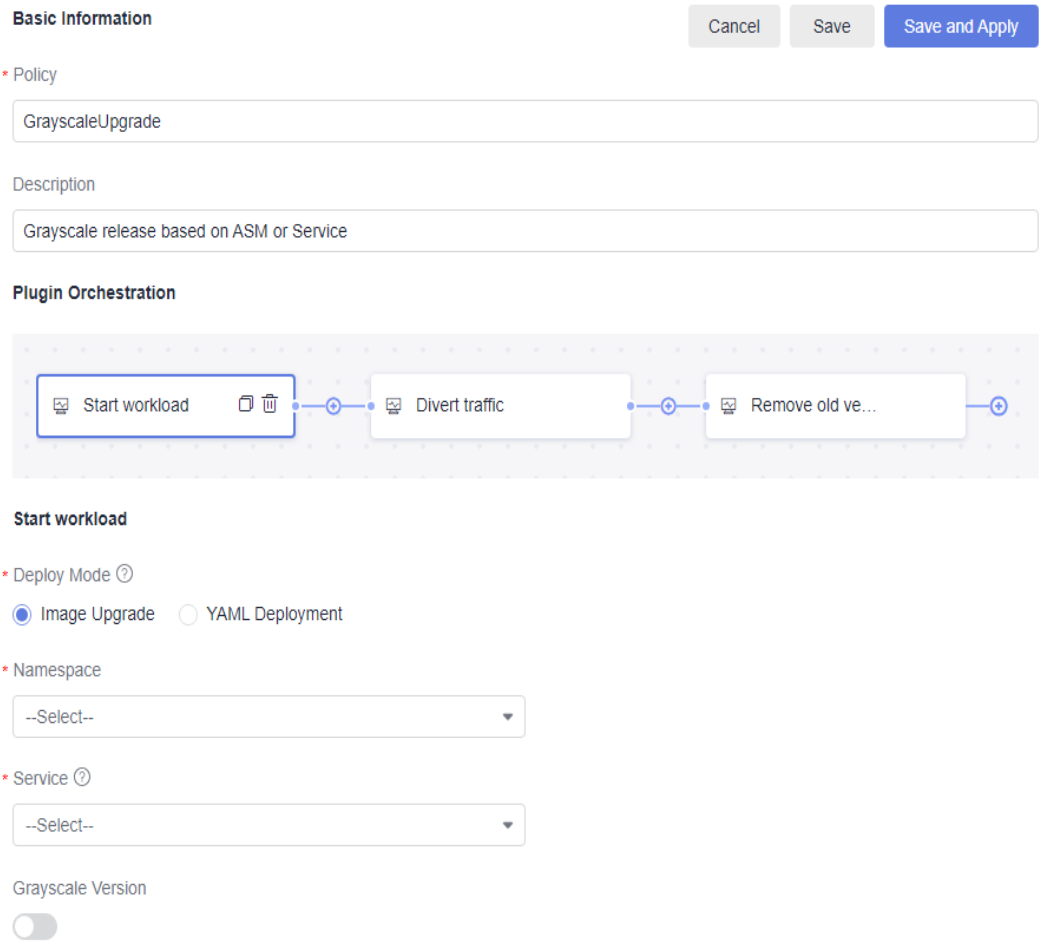


Table 15-5 Policy parameters

Parameter	Description
Policy	Policy name. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
Description	Enter a description with no more than 200 characters.
Orchestrating an extension	Set orchestration parameters by referring to **Configuring Atomic Extensions**. • Click to add an extension. • Click to clone an extension. • Click to delete an extension.

Step 6 Click **Save** after the configuration.

Step 7 Find the created policy on the left and click to apply it. The applied policy will be marked as **Default**.

----End

Configuring Atomic Extensions

CodeArts Release provides the following extensions (**Rolling upgrade**, **Generic k8s upgrade**, **Start workload**, **Divert traffic**, **Remove old version**, and **Manual check**) for rolling upgrade and grayscale upgrade:

- **Rolling upgrade**

  Rolling upgrade supports image upgrade and YAML deployment.

  – Image upgrade: Replace the container image in the workload.

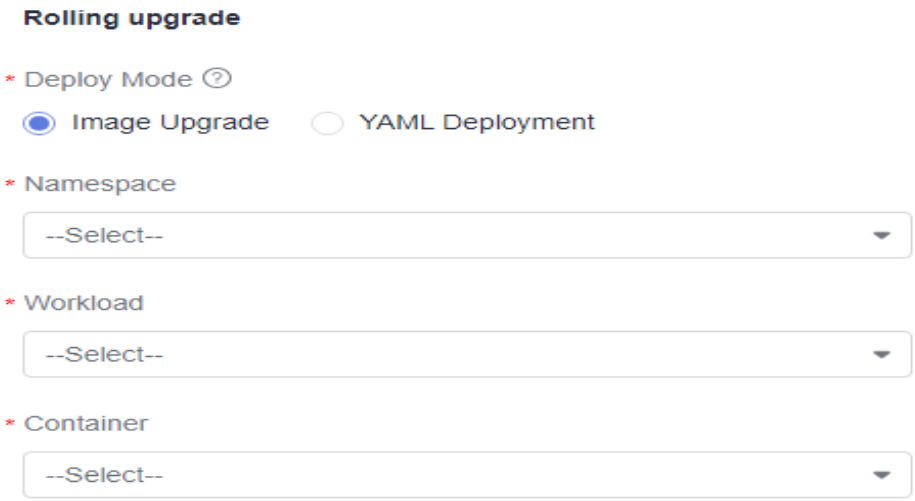


Table 15-6 Parameter description

Parameter	Description
Namespace	Namespace to which the service to be upgraded belongs.
Workload	Workload in the namespace.
Container	Container to be upgraded in the workload.

  – YAML deployment: Use the YAML file to deploy or upgrade the workload.

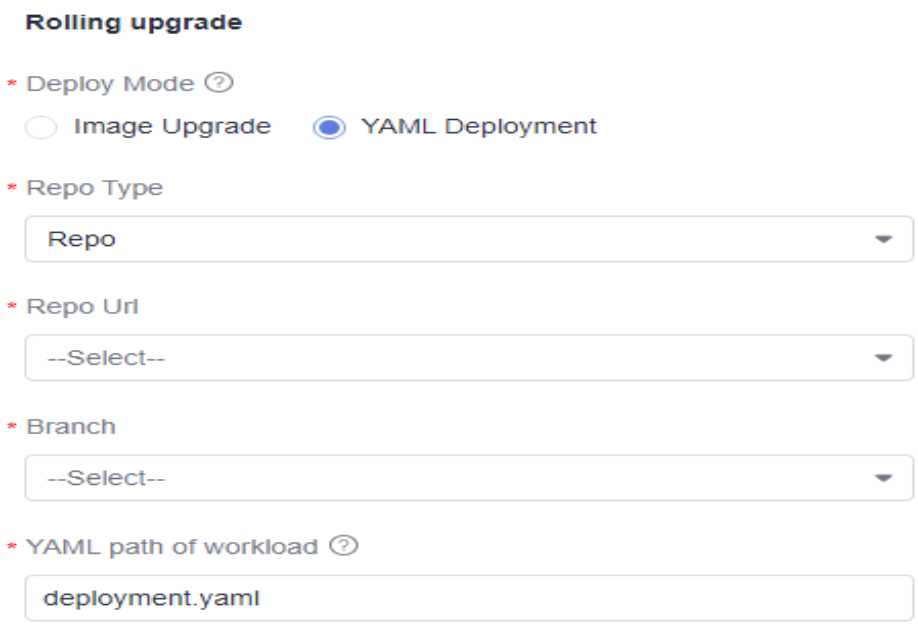


Table 15-7 Parameter description

Parameter	Description
Repo Type	Repository type. Only **Repo** is supported.
Repository	Code repository of the current project.
Branch	Branch of a code repository.
YAML Path of Workload	YAML path of the workload to be upgraded. Enter a relative path of the YAML file. • The current directory is the root directory of the code branch. • Only one YAML file is supported. • You can use *${variable name}* in a YAML path to reference an environment variable, and *{{variable name}}* in a YAML file to reference an environment variable.

- **Generic k8s upgrade**

  This extension obtains the required file based on the set repository information and file path and delivers the file to the Kubernetes cluster.

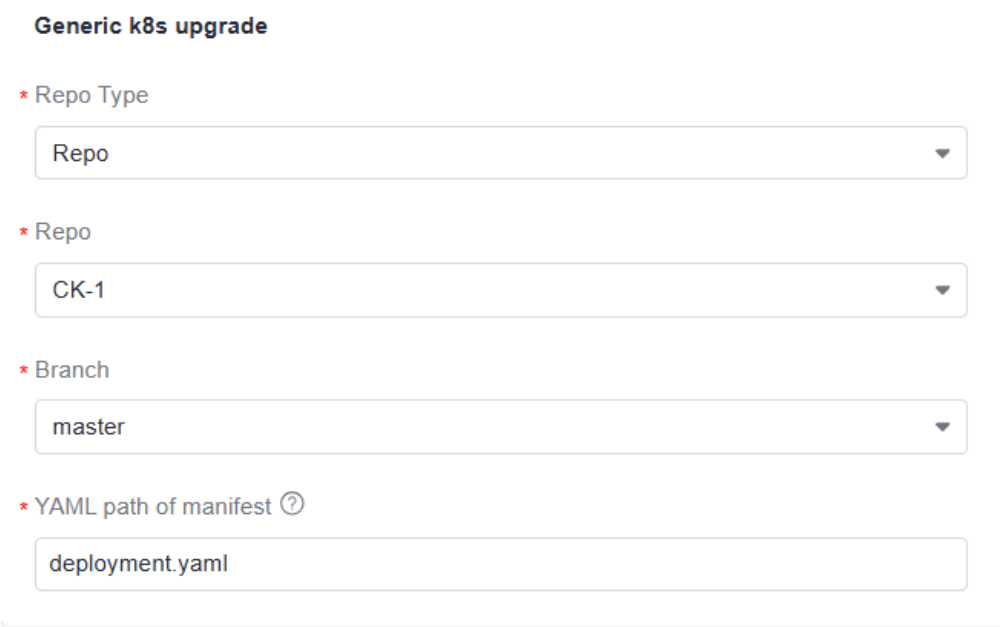


Table 15-8 Parameter description

Parameter	Description
Repo Type	Repository type. Only **Repo** is supported.
Repository	Code repository of the current project.
Branch	A branch in the code repository.
YAML path of manifest	Relative path of the YAML file. • The current directory is the root directory of the code branch. • You can use *${variable name}* in a YAML path to reference an environment variable, and *{{variable name}}* in a YAML file to reference an environment variable.

- **Start workload**

  Start workload supports image upgrade and YAML deployment.

  – Image upgrade: Upgrade a workload by replacing the container image with a new one, ensuring it has the same configurations as the currently running packages without changing anything else.

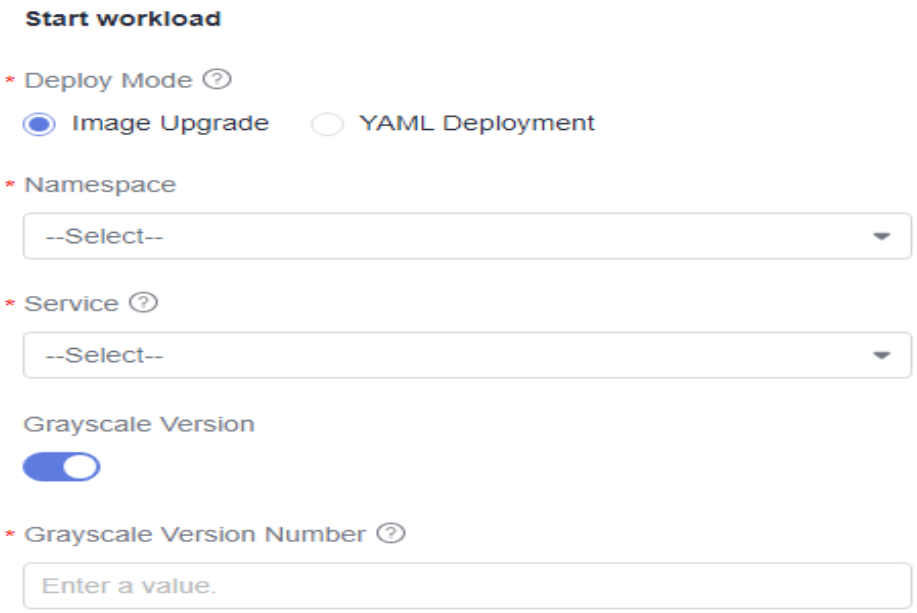


Table 15-9 Parameter description

Parameter	Description
Namespace	Namespace to which the service to be upgraded belongs.
Service	The service to be upgraded, which is associated with only one workload.
Grayscale Version	Disabled: The system automatically generates a grayscale version number. Enabled: You can configure a grayscale version number as needed.
Grayscale Version Number	The grayscale version number is used to distinguish the official version from the grayscale version. You can use *${ENV}* to reference environment variables. For example, **${TIMESTAMP}** indicates that the system timestamp variable is referenced as the gray version number. Enter only letters, digits, and underscores (_) with a maximum of 62 characters.

- YAML deployment: Use the YAML file to deploy or upgrade the workload.

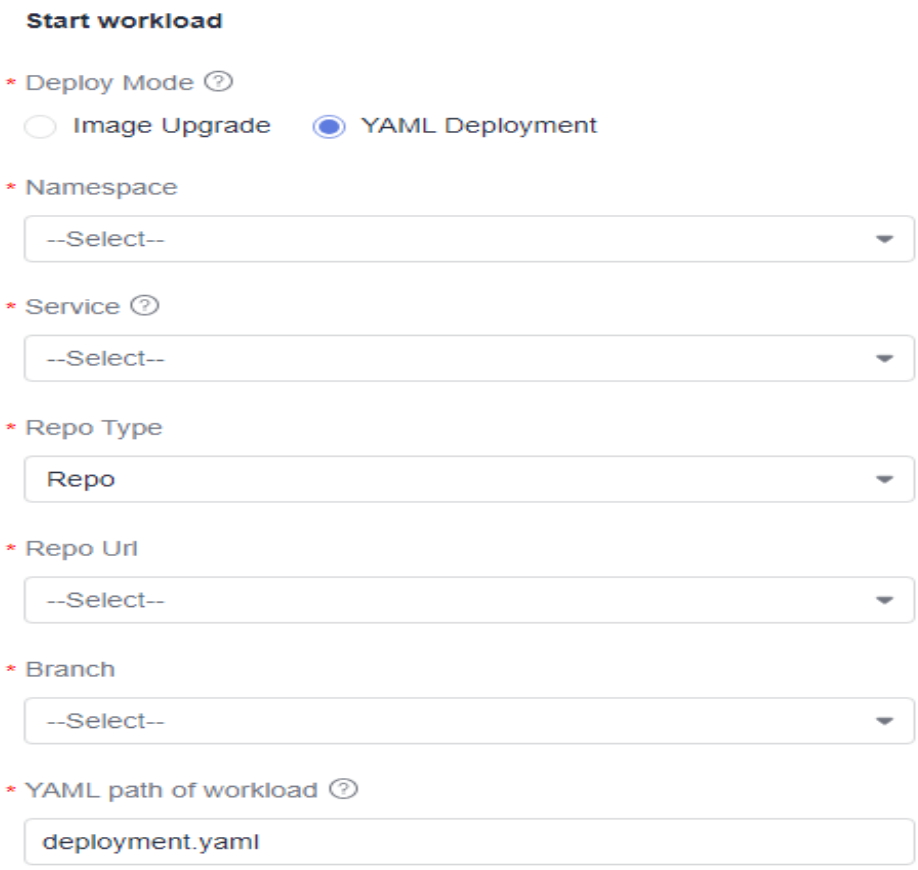


Table 15-10 Parameter description

Parameter	Description
Namespace	Namespace to which the service to be upgraded belongs.
Service	The service to be upgraded, which is associated with only one workload.
Repo Type	Repository type. Only **Repo** is supported.
Repository	Code repository of the current project.
Branch	Branch of a code repository.
YAML Path of Workload	Relative path of the YAML file. • The current directory is the root directory of the code branch. • Only one YAML file is supported. • You can use *${variable name}* in a YAML path to reference an environment variable, and *{{variable name}}* in a YAML file to reference an environment variable.

- **Divert traffic**

Traffic diversion includes:

- Service blue-green release: All traffic will be switched to the new workload (gray load).
- ASM grayscale release: Use ASM (Application Services Mesh) VirtualService and DestinationRule configurations to control access traffic, perform grayscale diversion based on traffic proportion and request headers. ASM must be installed in the cluster.

- **Remove old version**

  This extension automatically removes the old workload associated with the service. No configurations are required.

- **Manual check**

  With this extension, you can approve or reject the deployment policy when the pipeline pauses at a checkpoint, allowing the pipeline to either continue running or to stop.

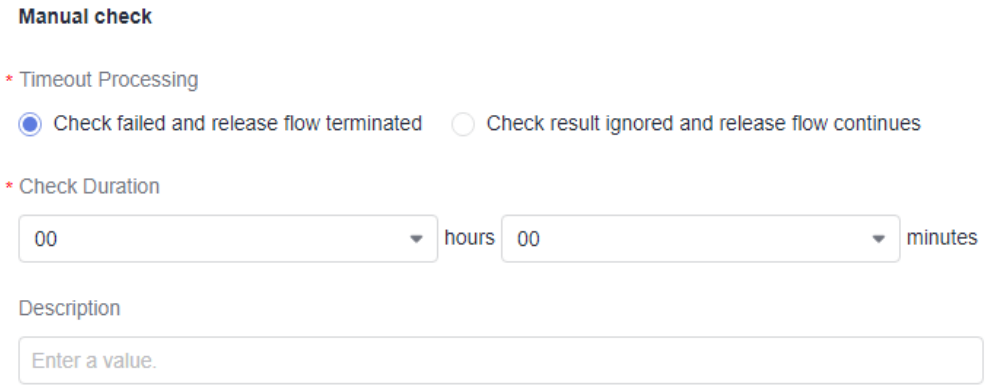


Table 15-11 Parameter description

Parameter	Description
Timeout Processing	Processing solutions: • Check failed and release flow terminated: Pipeline will pause at the checkpoint. If the policy is not approved within the specified period, the pipeline will stop. • Check result ignored and release flow continues: Pipeline will pause at the checkpoint. If the policy is not approved within the specified period, the pipeline will continue to run.
Check Duration	Time window for checking, which ranges from one minute to 12 hours.
Description	Description of the manual check. Enter no more than 200 characters.

15.5 Releasing an Environment

You can release an environment directly. Or you can use the **CloudNativeRelease** extension in a pipeline to trigger the configured release policy for releasing an environment.

Releasing an Environment

Step 1 Create a release environment.

Step 2 Click **Execute** in the upper right corner. In the displayed window, set parameters.

- Release policy: Select the policy configured in **Configuring an Environment Release Policy**.
- Artifact address: address of the image to be released. You can enter the SWR image address, or use the image from Image Center (for example, **nginx:latest**).

Step 3 Click **Confirm**.

----End

Releasing an Environment Through the CloudNativeRelease Extension

Step 1 Configure a pipeline.

Step 2 Add the **CloudNativeRelease** extension to the pipeline. For details, see **Table 15-12**.

This extension allows you to orchestrate environment release policies in CCE clusters. There is rolling upgrade release and grayscale release.

Figure 15-4 Configuring CloudNativeRelease

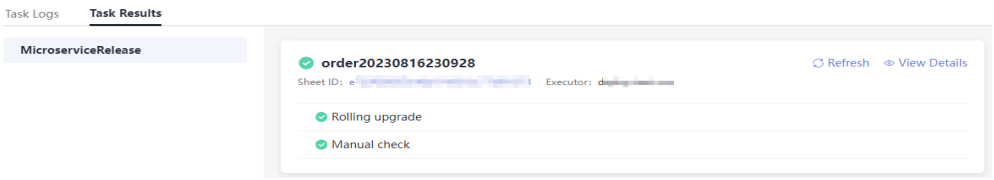
Table 15-12 Parameter description

Parameter	Description
Name	Extension name Enter only letters, digits, underscores (_), hyphens (-), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces, with a maximum of 128 characters.
Environment Level	Release environment type. Available environment types: development, test, pre-production, and production.
Environment	Environment to be released. For details, see **Creating a Release Environment**.
Artifact Path	Image path for deployment and release. Example: **swr.example.com/demo/springboot-helloworld:v1.1**. You can use ${} to reference pipeline parameters. Example: **swr.example.com/demo/springboot-helloworld:${version}**. **NOTE** SoftWare Repository for Container (SWR) is recommended. You can **build an image and push it to SWR** through CodeArts Build.

Step 3 Execute the pipeline after the configuration is complete.

Step 4 Click the task card to view the **Task Logs** and **Task Results**.

Figure 15-5 Checking the execution result



- **Task Logs**: displays real-time log information and running status.
- **Task Results**: displays basic task information, including the service ticket name, ticket ID, and trigger person.

  Click the service ticket ID or the **View Details** button to go to the details page. For details, see **Checking the Environment Release Result**.

----End

15.6 Checking the Environment Release Result

Step 1 Access the release environment list page.

Step 2 Click an environment name. The Environment Information page is displayed.

Step 3 On the displayed page, click the **Deployment History** tab.

Step 4 Click a service ticket name to check details. The details page displays the information of **release flow**, **atomic extension**, and some **basic information**.

- Release flow
  - Release flow displays information such as the execution result, service ticket type, executor, pipeline, and release policy. You can click an atomic extension to check its details.
  - Cancel: You can click **Cancel** to cancel the release.
  - Retry: If the release fails or the release is canceled, you can click **Retry** to retry the release.
  - Rollback: Click **Rollback**. In the displayed dialog box, if you confirm the rollback, the release will be canceled and the service state will be restored to its pre-release state.

    NOTE

    When the environment is directly released, the pipeline information is not displayed in the deployment ticket.

    Rollback can be performed anytime. If the current deployment version does not meet your expectation, you can quickly restore the environment to the previous one through rollback.
- Basic information

  Basic information includes environment name, policy, service endpoint, variable version, image, start time, and end time.
- Atomic extension release

  The release detail of atomic extension is displayed. You can click Refresh to refresh the details.

Figure 15-6 Atomic extension release

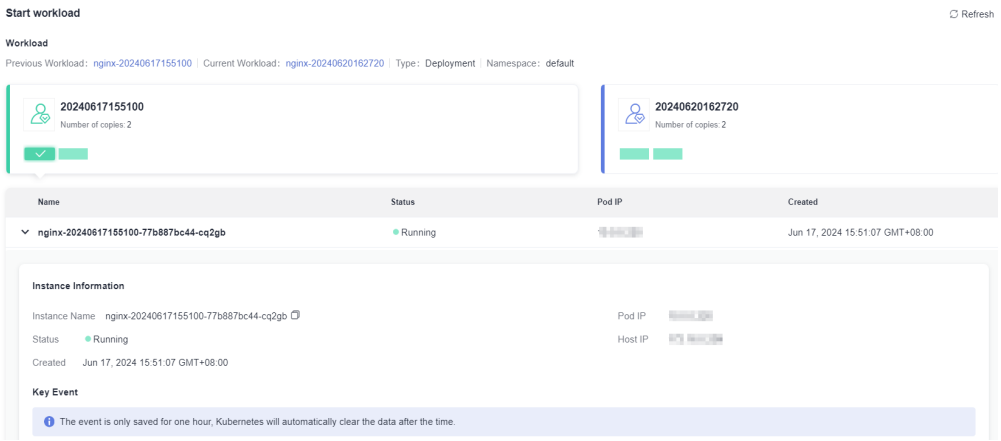


Table 15-13 Atomic extension release

Extension	Release Information
Rolling upgrade	The details page displays the workload to be upgraded, instance information, and key event. – Workload Workload name, type, namespace, and creation time – Instance information Instance name, status, pod IP address, host IP address (IP address of the node where the pod is located), and creation time. – Key event K8s component name, event type, K8s event, first occurrence time, and recent occurrence time. Key event information can help you locate pod faults.
Start workload	The details page displays workload to be upgraded, pod information, and key event. You can click the version cards to check the previous or the current workload information. – Workload Previous workload name, current workload name, type, and namespace – Instance information Instance name, status, pod IP address, host IP address (IP address of the node where the pod is located), and creation time. – Key event K8s component name, event type, K8s event, first occurrence time, and recent occurrence time. Key event information can help you locate pod faults.
Divert traffic	The details page displays the service name, old version number, new version number, and namespace.
Remove old version	The details page displays the workload name, workload type, and namespace.
Manual check	The details page displays check duration, operation time, description, and status.

----End

15.7 Deploying a Microservice Based on GitOps

GitOps-based deployment provides a fully managed, secure, and automated approach to Kubernetes deployments. This is achieved by the GitOps runtime that is installed on customers' Kubernetes clusters using ArgoCD and CodeArts agents. This solution fully operates behind customers' firewall without requiring direct external access to their clusters. The GitOps runtime installed on the clusters pulls deployment configurations from Git and communicates securely with CodeArts

Release via CodeArts agents. This solution ensures that all deployment configurations are version-controlled in Git, allowing for declarative, auditable, and consistent application delivery across environments.

Constraints

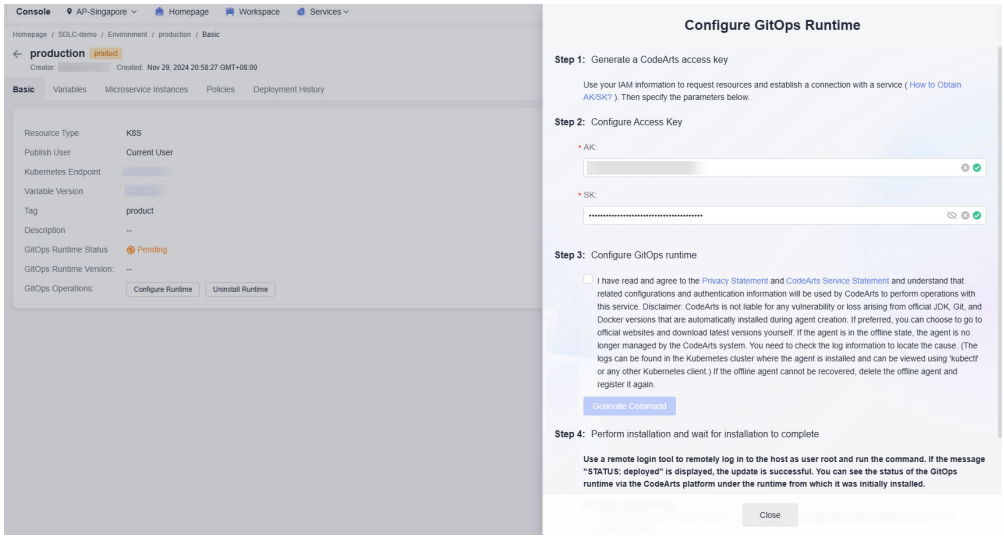
This feature is only available in the **AP-Singapore** region.

Installing the GitOps Runtime

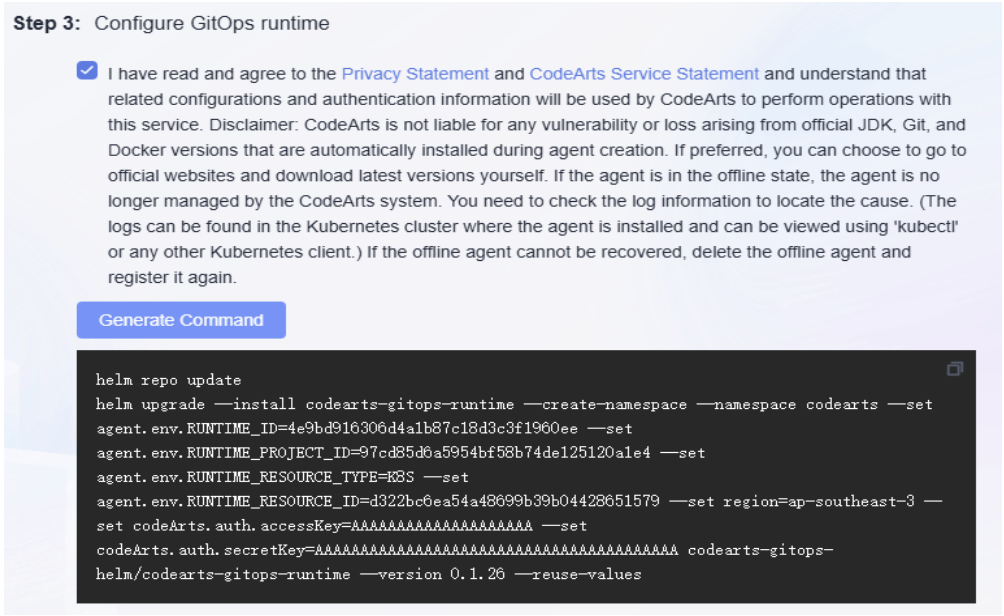
Step 1 Create a release environment by referring to **Creating a Release Environment**.

Step 2 Click the environment name to go to the **Basic** tab page.

Step 3 Click **Configure Runtime**. On the displayed **Configure GitOps Runtime** page, obtain and configure an access key by referring to Step 1 in the wizard.



Step 4 Click  and then click **Generate Command**. Expand the **helm** installation commands, copy the commands, and run them on your Kubernetes cluster.



Step 5 Click **Close**. The GitOps runtime is configured.

Step 6 On the **Basic** tab page, check and monitor the GitOps runtime status.

----End

Creating a Microservice Deployment Configuration

Step 1 Access the CodeArts Pipeline homepage by referring to **Accessing CodeArts Pipeline**.

Step 2 Choose **CICD** > **Pipeline** > **Microservices**. The microservice list page is displayed.

Step 3 Create a microservice by referring to **Creating a Microservice**, or search for the microservice for which you want to create a deployment configuration. Click the microservice name to go to the **Overview** tab.

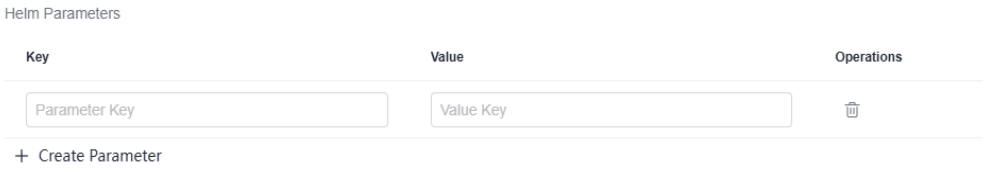
Step 4 Click the **Deployment Configurations** tab, and then click + Create Deployment Configuration. On the **Create Deployment Configuration** page, set the basic information. The parameters are described in **Table 15-14**.

Table 15-14 Parameter description

Parameter	Description
Project	Project to which the deployment configuration belongs. The project cannot be changed.
Deployment Configuration Name	Name of the deployment configuration. Enter a maximum of 128 characters, including letters, digits, hyphens (-), and underscores (_).
Deployment Engine	Choose between **Release Policy** and **GitOps**.
Artifact Type	Choose between **Helm** and **Kustomize**.
Artifact Source	Artifact repository source. Currently, only Git repositories are supported.
Code Source	Code repository source. Currently, only CodeArts Repo is supported.
Repository	A repository in the current project. Select a GitOps repository.
Branch	A branch in the code repository.
Path	A path in the code repository.
Namespace	Namespace on which the deployment will be performed.
Description	Description of the deployment configuration. Enter a maximum of 200 characters.
Helm Value Files	List of files inside the same Git repository. These files will be used as Helm values for Helm templating during deployment.

Parameter	Description
Helm Parameters	Parameters injected for Helm templating during deployment.

Step 5 If **Artifact Type** is set to **Helm**, click **Create Parameter** under **Helm Parameters** and set the key and value. To delete a parameter, click .



Step 6 Click **OK**. The microservice deployment configuration is created.

----End

Deploying a Microservice Instance

Step 1 Access the CodeArts Pipeline homepage by referring to **Accessing CodeArts Pipeline**.

Step 2 Choose **CICD** > **Release** to access the **Environment** page.

Step 3 Click the environment where you want to deploy a microservice to go to the **Basic** tab page.

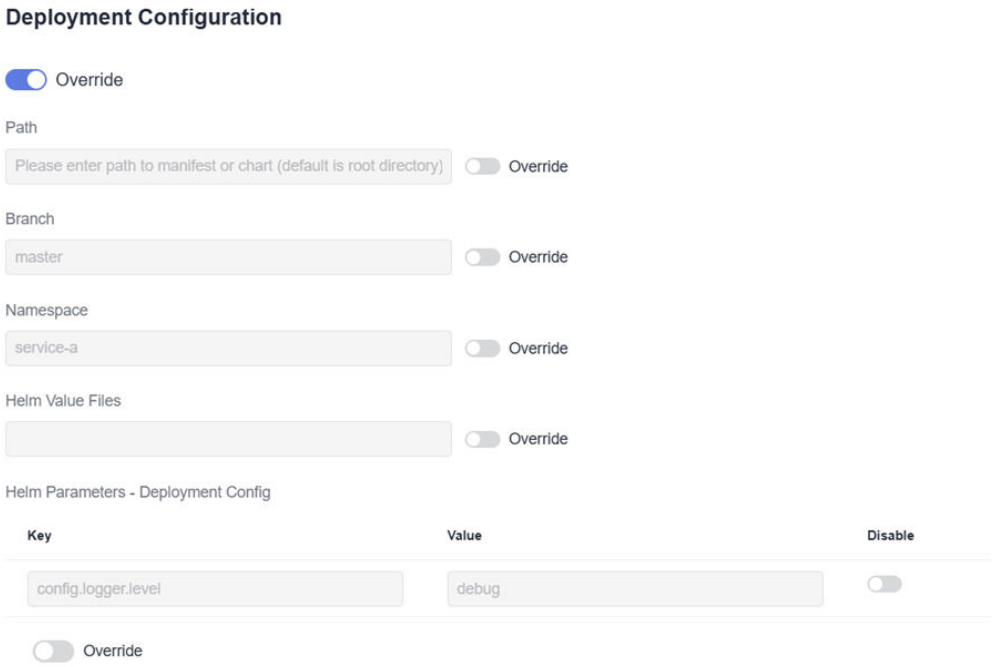
Step 4 Click the **Microservice Instances** tab.

Step 5 Click **Create Microservice Instance**. On the displayed page, set the basic information. The parameters are described in **Table 15-15**.

Table 15-15 Parameter description

Parameter	Description
Project	Project to which the deployment configuration belongs. The project cannot be changed.
Microservice	Name of the microservice to be deployed.
Deployment Configuration	Name of the deployment configuration to be used to deploy the microservice.
Description	Description of the microservice instance. Enter a maximum of 200 characters.

Step 6 Click to determine whether to overwrite the default deployment configuration.



Step 7 Click the microservice name. The **Overview** tab page is displayed. Check the deployed microservice.

Step 8 Choose **CICD** > **Release**, click the name of the environment where the microservice has been deployed, and then click **Deployment History** to check the microservice deployment records.

- Only one microservice instance is allowed for a microservice on a single environment.

----End

Automatically Updating a Deployed Microservice Using CodeArts Pipeline

Step 1 Create a pipeline by referring to **Creating a Pipeline**.

Step 2 On the **Task Orchestration** tab of the pipeline details page, click **New Job**. The **New Job** page is displayed.

Step 3 Click **Add** next to the **DeployMicroserviceInstance** extension, and set the basic information. The parameters are described in **Table 15-16**.

Table 15-16 Parameter description

Parameter	Description
Name	Name of the extension. Enter a maximum of 128 characters. Only letters, digits, spaces, and these special characters are allowed: -_,;:./()
Environment	Release environment, which is associated with your cluster resources. It contains the configurations for the upcoming deployment.

Parameter	Description
Microservice	Microservice that has been deployed. It is associated with the microservice instance configured in **Deploying a Microservice Instance** and will be automatically updated.
Helm Parameters Patch	Parameters injected for Helm templating during deployment.

Step 4 Click **OK**. The **DeployMicroserviceInstance** extension is added.

Step 5 Click **Save**. CodeArts Pipeline will automatically update this deployed microservice.

----End

