CodeArts Pipeline

User Guide

Issue	01
Date	2024-10-15

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

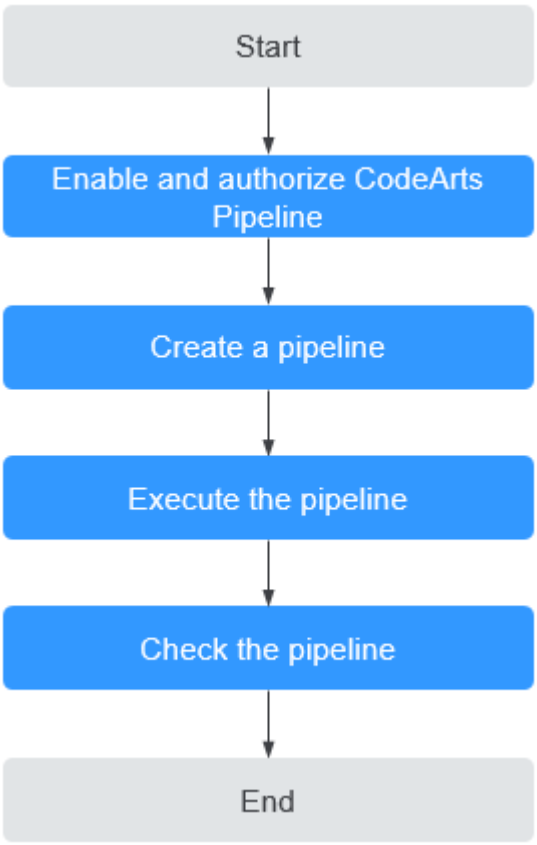
CodeArts Pipeline is a visual platform for automating task scheduling. You can use Pipeline to automate tasks in services like **CodeArts Build**, **CodeArts Check**, **CodeArts TestPlan**, and **CodeArts Deploy**.

You can orchestrate these automated jobs for different scenarios, such as application deployment in the development, test, or production environment. A single configuration triggers executions repeatedly to avoid inefficient 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



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 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, choose **Settings** > **General** > **Service Permissions**.

Pipeline-related resources are in CodeArts Pipeline, including pipelines, policies (project-level), microservices, environments, changes, and parameter groups.

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
Project creator	√	√	√	√	√	√
Project manager	√	√	√	√	√	√
Developer	√	√	√	×	×	×
Test manager	√	×	×	×	×	×
Tester	√	×	×	×	×	×
Participant	√	×	×	×	×	×
Viewer	√	×	×	×	×	×
Product manager	√	×	×	×	×	×
System engineer	√	√	√	√	√	√
Committer	√	√	√	×	×	×

NOTE

- 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 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 creator	√	√	√	√
Project manager	√	√	√	√
Developer	√	√	√	√
Test manager	√	×	×	×
Tester	√	×	×	×
Participant	√	×	×	×
Viewer	√	×	×	×
Product manager	√	×	×	×
System engineer	√	√	√	√
Committer	√	√	√	√

NOTE

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 creator	√	√	√	√
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 creator	√	√	√	√
Project manager	√	√	√	√
Developer	√	√	√	√
Test manager	√	×	×	×
Tester	√	×	×	×
Participant	√	×	×	×
Viewer	√	×	×	×
Product manager	√	×	×	×

Role	View	Create	Edit	Execute
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 creator	√	√	√	√	√	√
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 creator	√	√	√	√	√	√

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

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

Table 2-10 Project-level production permissions

Role	View	Create	Edit	Delete	Execute	Roll Back
Project creator	√	√	√	√	√	√
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 creator	√	√	√	√

Role	Create	Delete	Edit	Associate
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 creator, 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 creator, 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.

3 Accessing CodeArts Pipeline

This section describes how to access 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 On the displayed page, 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 with the GUI

Preparations

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

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 Pipeline basic information

Parameter	Description
Name	Enter a pipeline name. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
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.

Parameter	Description
Code Source	Code source associated with the pipeline: • CodeArts Repo: provides comprehensive code hosting services for enterprises and Git-based online code hosting services for software developers. • Third-party code source – GitLab: After connecting to a GitLab 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. **NOTE** • GitLab code source is available in **LA-Mexico City2**, **LA-Sao Paulo1**, and **AP-Singapore** regions. • 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 **FAQs**.
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*.
Repository	Code repository associated with the pipeline.
Default Branch	Branch used when a pipeline is executed manually or at a specified time.
Repo Endpoint	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	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}*.
Description	Enter a maximum of 1,024 characters.

Step 3 After configuring the basic information, click **Next**. The **Select Template** page is displayed.

- You can select a system or custom template to quickly create a pipeline. Jobs will be automatically generated based on the selected template. For more information, see **Managing Pipeline Templates**.
- You can also select **Blank Template** to 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.

----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.

- Access the template list via:
  - Homepage: Access the Pipeline homepage, and switch to the **Templates** tab page.
  - A project: Access the pipeline list in a project, click **More** > **Templates** in the upper right corner.

  You can perform the following operations on templates.

  Table 4-2 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** to delete the template as prompted. **NOTE** System templates are used to clone or generate pipelines. They cannot be edited or deleted.

- Customize a pipeline template
  a. Access the template list.
  b. Click **Create Pipeline 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), and description (optional). Language is **None** by default.

- **Task Orchestration**: Pipeline stages and some extensions can be added to a pipeline template. After jobs such as build, code check, deployment, and API test are configured in a template, corresponding jobs will be created when you create a pipeline using this template.

  NOTE

  - Code source is not required for a template.
  - Stage entry configuration is not supported for template orchestration.

- **Parameter Configuration**: Add parameters to the template. Pipeline template parameters include custom and predefined parameters. Custom parameters include string, enumeration, and auto-increment types. For details about how to configure parameters, see **Configuring Custom Parameters**.

d. Click **Save** to complete the template creation.

4.2 Creating a Pipeline with YAML

Preparations

- **Create a project**.
- **Create a code repository**.

Creating a Pipeline with YAML

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

Step 2 Click **Create Pipeline**. Configure parameters by referring to **Table 4-3**.

Table 4-3 Pipeline basic information

Parameter	Description
Name	Enter a pipeline name. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
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.
Code Source	Select **Repo** (CodeArts Repo). It provides comprehensive code hosting services for enterprises and Git-based online code hosting services for software developers. **NOTE** You can only use **Repo** to create a YAML-based pipeline.

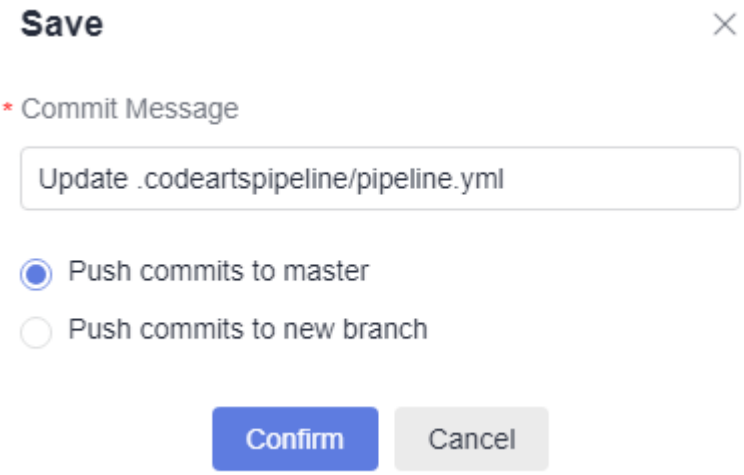
Parameter	Description
Orchestration Method	Select **YAML**: Use YAML to orchestrate a pipeline (one YAML file can be used for multiple pipelines). Syntax auto-completion and validation are available.
Repository	Code repository associated with the pipeline.
Default Branch	Branch used when a pipeline is executed manually or at a specified time.
Configuration File	• **New**: Create a YAML file. • **Existing**: Orchestrate a pipeline based on the existing YAML file. The orchestrated content will overwrite the original YAML file. For details about how to compile a YAML file, see **YAML Syntax**.
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	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	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}*.
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**.
- 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.

Step 4 After orchestration, click **Save**, enter the commits message, and push commits in one of the following ways:



- Push commits to the existing branch: If you created the pipeline with a new YAML file, commits will be pushed to the default branch. If you created 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**.

----End

YAML Syntax

- 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
          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 }}
```


5 Configuring a Pipeline

5.1 Orchestrating Pipeline Stages

A stage is a basic part of a pipeline. Jobs can be orchestrated and managed in different stages. Closely associated jobs can be managed in one stage for intuitive workflows.

Configuring Stages

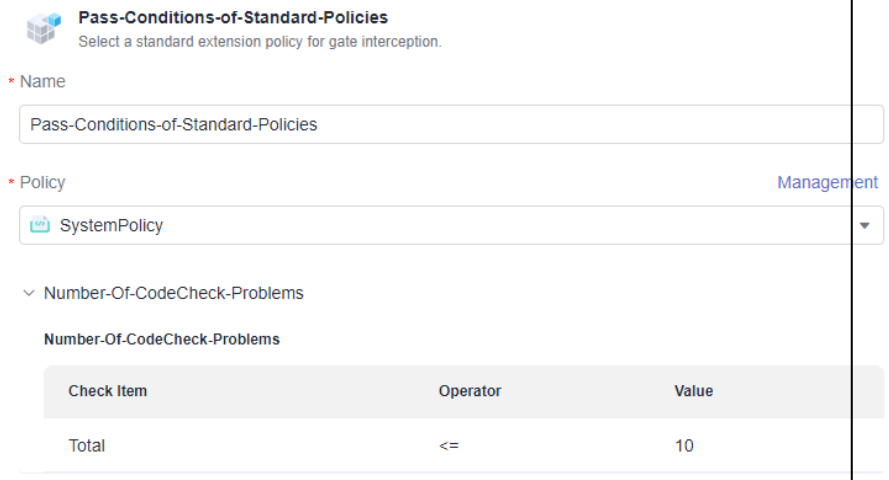
Step 1 Access the CodeArts Pipeline homepage.

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

Table 5-1 Stage management

Operation	Description
Editing a stage	Click  . In the displayed window, you can configure the stage name and whether to always run jobs in the stage. • **Always Run**: If you select **Yes**, jobs in this stage will be executed and cannot be canceled. • **Always Run**: If you select **No**, jobs will be selected by default but can be deselected.
Cloning a stage	Click  to clone a pipeline stage.
Deleting a stage	Click  and confirm the deletion as prompted.

Operation	Description
Sorting a stage	Click and drag to adjust the stage sequence.
Setting the entry type	You can set in what conditions can a pipeline proceed to the next stage. Click . In the displayed window, configure the entry 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.
Pass conditions	You can set rules and policies to decide in what conditions can a pipeline complete the current stage. • Rules determine whether to pass an extension output by comparing its relationship with a threshold, and a policy is composed of multiple such rules. For details, see **Configuring a Rule**. • A policy is a set of rules and can be applied to multiple pipelines. There are **tenant-level policies** and **project-level policies**. Policies control pipeline runs and ensure high-quality delivery. **NOTE** • Only **Pass-Conditions-of-Standard-Policies** is available. You can select a created policy. • You can set exclusive pass conditions for each stage. • You can set multiple pass conditions for one stage. 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, save the pipeline.

----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 5-1 Creating a rule

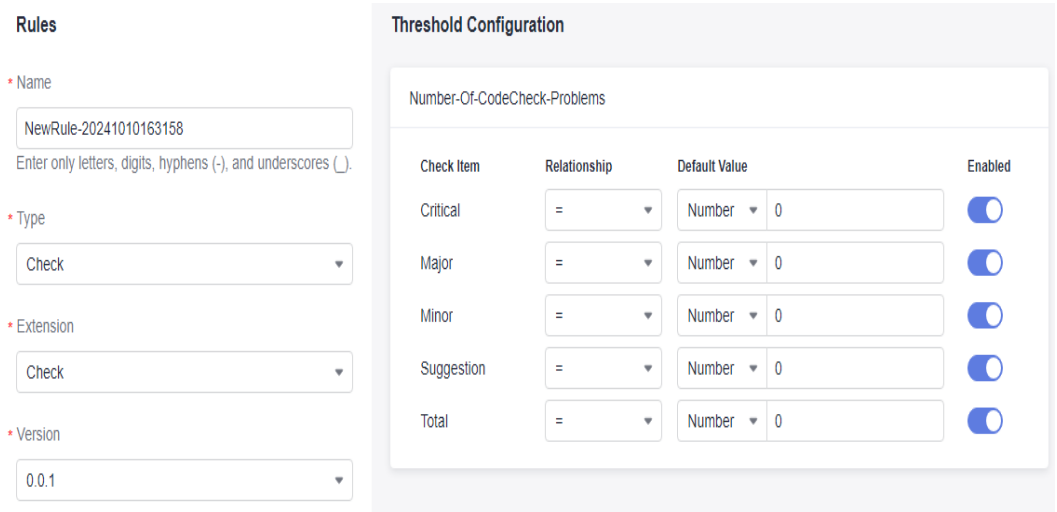


Table 5-2 Rule parameters

Parameter	Description
Name	Rule name, which is generated based on the current time. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
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.

Parameter	Description
Extension	All extensions of the selected type. • Extensions of the **Build** type: Set thresholds for build results. For example, you can select the official **Build** extension to set thresholds for the Maven unit test. • Extensions of the **Check** type: Set thresholds for code check results. For example, you can select the official **Check** extension to set thresholds for code check issues. • Extensions of the **Test** type: Set thresholds for test results. For example, you can select the official **TestMan** extension to set thresholds for the test case pass rate in test suites.
Version	Extension versions that allow for threshold settings.
Threshold Configuration	(Optional) Automatically generated based on the selected extension version. Note that if you changed threshold settings, the relevant rule and policy would also be changed. **NOTE** If you set the relationship to **Exclude** or **Include**, **Text** is usually used. For the check item **Pass Ratio**, the value ranges from 0 to 1.

Step 5 Click **Confirm** to create a rule. 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.

  NOTE

  - 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. On the displayed dialog box, confirm the deletion.

  NOTE

  After a rule is deleted, all policies that reference the rule automatically cancel the reference.

----End

Configuring the Tenant-level Policy

Tenant-level policies are tenant-level resources and can be used in pass conditions for all pipelines 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** > **Policies**. The policy list page is displayed.

NOTE

There is a system policy by default. You can check and use the policy, but cannot edit or delete it.

Step 4 Click **Create Policy** and set parameters.

Table 5-3 Policy parameters

Parameter	Description
Name	Policy name, which is generated based on the current time by default. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
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: You can 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. **NOTE** A maximum of 20 rules can be selected for a policy.

Step 5 Click **Confirm** to create a policy. You can also perform the following operations in the policy list.

- Click to edit a policy.
- 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.

  NOTE

  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 to enable or disable a policy.

  NOTE

  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 the pass conditions.

----End

Configuring Project-level Policies

Project-level policies are project-level resources and can be used in pass conditions for all pipelines of the current project.

Step 1 Access the CodeArts Pipeline homepage through a project.

Step 2 Click the **Policies** tab.

Step 3 On the displayed page, click **Create Policy**.

Table 5-4 Policy parameters

Parameter	Description
Name	Policy name, which is generated based on the current time by default. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
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: You can 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. **NOTE** A maximum of 20 rules can be selected for a policy.

Step 4 Click **Confirm** to create a policy.

- 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  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 the 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 a project-level policy based on the selected tenant-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

A job is the minimum manageable execution unit in a pipeline. Jobs can be orchestrated in serial and parallel mode in a stage.

Orchestrating 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.

NOTE

- Click under a job to add a serial job. For example, a build job and 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 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**. Configure the following information: • Enter an extension name. • Select a job to be called. If no proper job is available, create a job as prompted. • If the called job 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.

Operation	Description
Replacing an extension	Move the cursor to an extension card, click , and select **Replace** to replace the extension. Or, click **Replace Extension** 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	Set the job ID, executor, and execution condition. • Job ID: The job ID should be unique. Enter only letters, digits, hyphens (-), and underscores (_) with a maximum of 128 characters. • You can use the built-in executor or customize one. – 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 a custom executor, add an agent pool. For details, see **Agent Pools**. **NOTE** You only need to configure executors for non-job-level extensions. • Select Job – Always: Job will always be selected for execution and cannot be canceled. – Disabled: Job cannot be selected for execution. – Selected by default: Job is selected for execution by default. – Not selected by default: Job is not selected for execution by default. • Execution conditions are the triggers for executing jobs in a pipeline. – **Even when previous job is not selected**: The current job is executed if the previous job is completed or not selected. – **When previous job succeeds**: The current job is executed only when the previous job is successfully executed. – **If previous job fails**: The current job is executed only when the previous job fails. – **Always**: The current job is always executed regardless of the previous job's final state (failed, completed, canceled, or ignored). – **With expression**: When the previous job is **COMPLETED**, **FAILED**, **CANCELED**, and **IGNORED** and the expression result is **true**, the current job will be executed. The expression is in the format of *${{value}}* and can be any combination of contexts, operators, functions, or literals. For details about the expression, see **Expressions**. Example: If the current job is executed regardless of whether the previous job (ID: **job_1**) succeeded or failed, the expression can be as follows: `${{ jobs.job_1.status == 'COMPLETED' || jobs.job_1.status == 'FAILED' }}`

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 clone 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. **NOTE** Job sequence cannot be adjusted when jobs are executed in parallel.

Step 6 After the configuration, save the pipeline.

----End

Expressions

An expression can be any combination of contexts, operators, functions, or literals. You can use an expression as the execution condition to control job execution. Contexts can be accessed programmatically with expressions, so information such as pipeline runs, sources, variables, and jobs can be transferred within a pipeline.

- Pipeline contexts

  Contexts are a way to access information about pipeline runs, sources, variables, and jobs. Each context is an object that contains various attributes. The following table lists pipeline contexts.

  Table 5-7 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 reached the final states in each pipeline run.

  - Context reference format

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

    context indicates the pipeline context, *attribute_name* indicates the attribute.

- Context scenarios

  Most contexts can be used in any job and step of a pipeline.

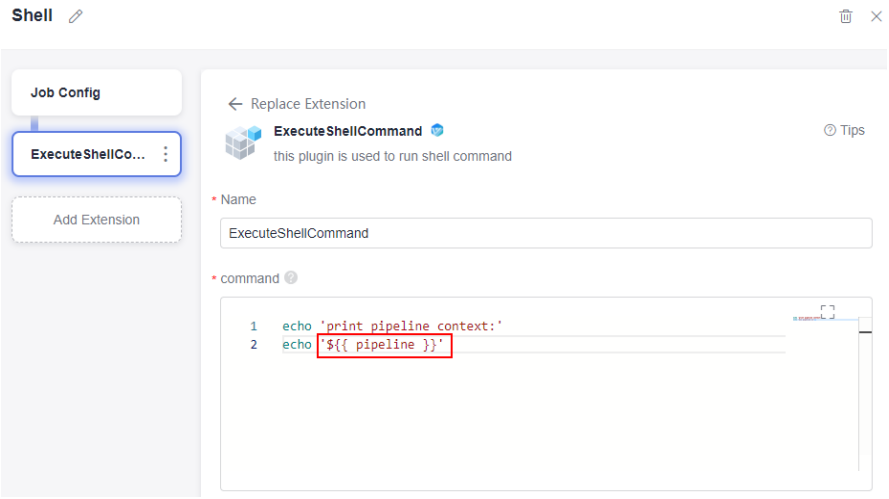
  - You can use contexts to specify the execution condition of a job.

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

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

  - You can use contexts when configuring parameters to query information.

    Figure 5-2 Referencing pipeline contexts

    The following expression shows how to obtain all pipeline run information.

    ```
    ${{ pipeline }}
    ```

    The following expression shows how to obtain the triggering mode of a pipeline.

    ```
    ${{ pipeline.trigger_type }}
    ```

- Contexts attributes

Table 5-8 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 triggering mode 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 triggering 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).	

Context	Attribute	Type	Description	Example
	sources.<alias>.tag	string	Tag name when the tag is triggered.	
	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.	

Context	Attribute	Type	Description	Example
	jobs.<job_id>.metrics.<metric_name>	string	The running metric name of a job.	

- **Operator**

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

  Table 5-9 Expression 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 functions that can be used in expressions.

Table 5-10 Expression 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**.
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**.
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**.

5.3 Configuring Pipeline Parameters

Pipeline parameters can be transferred among jobs. By configuring pipeline parameters, you can streamline data of build, deployment, and API test jobs. Parameters include:

- **Predefined Parameters**: They cannot be configured, deleted, or edited.
- **Custom Parameters**: You can add parameters of string, enumeration, or auto-increment type.
- **Parameter Groups**: You can associate all pipelines in the project with a parameter group.

NOTE

- If a code source alias is set, the repository-related system parameter 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 pipeline is associated with multiple parameter groups and parameters with the same name exist, the value of the parameter in the last associated parameter group will be used.
- The parameter reference format is *${Parameter name}*. Enter **$** in the text box and the parameter list will be displayed.

Predefined Parameters

Table 5-11 Predefined parameters

Parameter	Description
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.
PASS_CONDITIONS_LINK	Pipeline execution details link.
PIPELINE_RUN_ID	Pipeline run ID.

Parameter	Description
MERGE_ID	Merge request ID.
WEBHOOK_PAYLOAD	Webhook request payload information.
Repo01_REPOSITORY_NAME	Repository name.
Repo01_SOURCE_BRANCH	Name of the source branch for repository operations.
Repo01_TARGET_BRANCH	Name of the target branch for repository operations.
Repo01_TAG	Repository tag name.
Repo01_COMMIT_ID	The last commit ID before 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 custom parameters.

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. Or click **Create Parameter** to add new parameters.

Table 5-12 Custom parameters

Parameter	Description
Name	Enter only letters, digits, and underscores (_) with a maximum of 128 characters. **NOTE** The specified name cannot be the same as that of a predefined parameter.
Type	Parameter types include **String**, **Auto-Increment**, and **Enumeration**.
Default	Default value of the parameter. • **String**: The value contains no more than 8,192 characters. • **Auto-Increment**: The value contains no more than 8,192 characters. **NOTE** 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 letters, digits, hyphens (-), underscores (_), commas (,), periods (.), and slashes (/) with a maximum of 8,192 characters. After you select **Enumeration**, the **Enumeration** dialog box is displayed. You can set optional values. After the setting is complete, you can click the **Default** drop-down list box to select a value. Or, you can click **Enumeration** to change the value.
Private Parameter	If a parameter is private, the system encrypts the parameter for storage and decrypts the parameter for usage. Private parameters are not displayed in run logs.
Runtime Setting	If **Runtime Setting** is enabled, you can change the value of the parameter during execution configuration.
Description	Enter a maximum of 512 characters.

NOTE

- You can create a maximum of 20 custom parameters.
- You can click to delete a parameter.

Step 5 After the configuration, save the pipeline.

----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.

Figure 5-3 Creating a parameter group

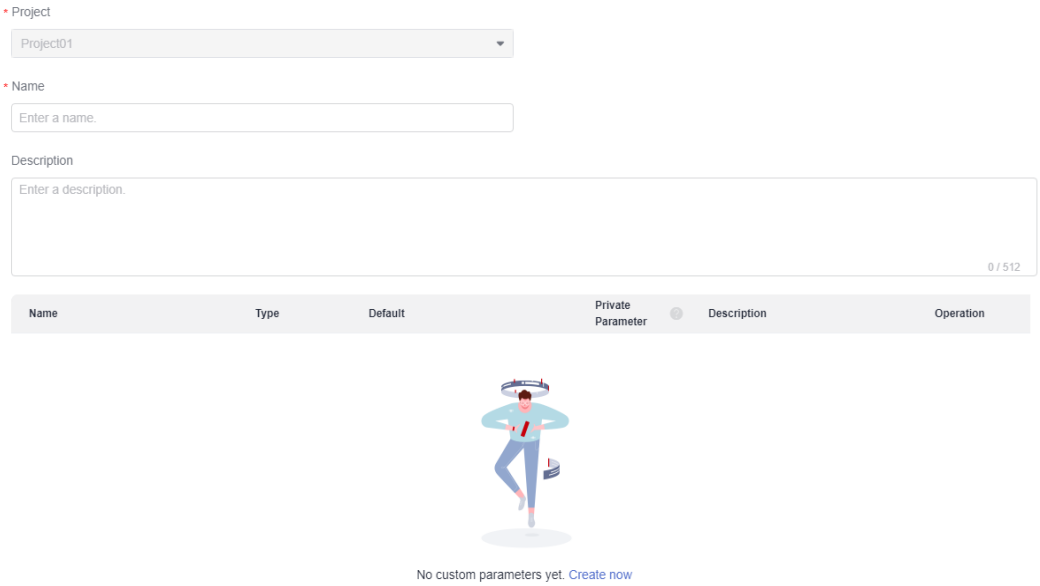


Table 5-13 Parameter group description

Basic Information	Description
Project	Project to which the parameter group belongs. The project cannot be changed.
Name	Enter only letters, digits, and underscores (_) with a maximum of 128 characters.
Description	Enter a maximum of 512 characters.
Custom parameter list	Click **Create now** to create custom parameters. For details, see **Configuring Custom Parameters**.

NOTE

You can create a maximum of 5 parameter groups and add a maximum of 20 custom parameters to a group.

Step 4 Click **OK** to create a parameter group.

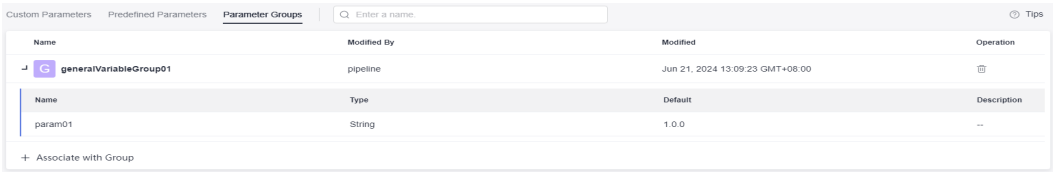
Step 5 Go to the pipeline editing page, choose **Parameter Configuration** > **Parameter Groups**.

Step 6 Click **Associate Now**, select a 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 with the parameter group.

Figure 5-4 Associating with a parameter group



Step 7 After the configuration, save the pipeline.

----End

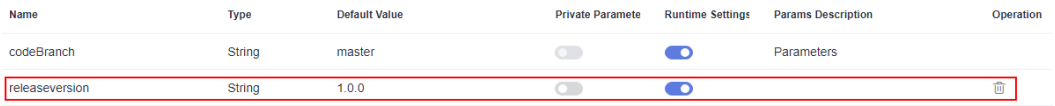
Using a Parameter in a Pipeline

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.

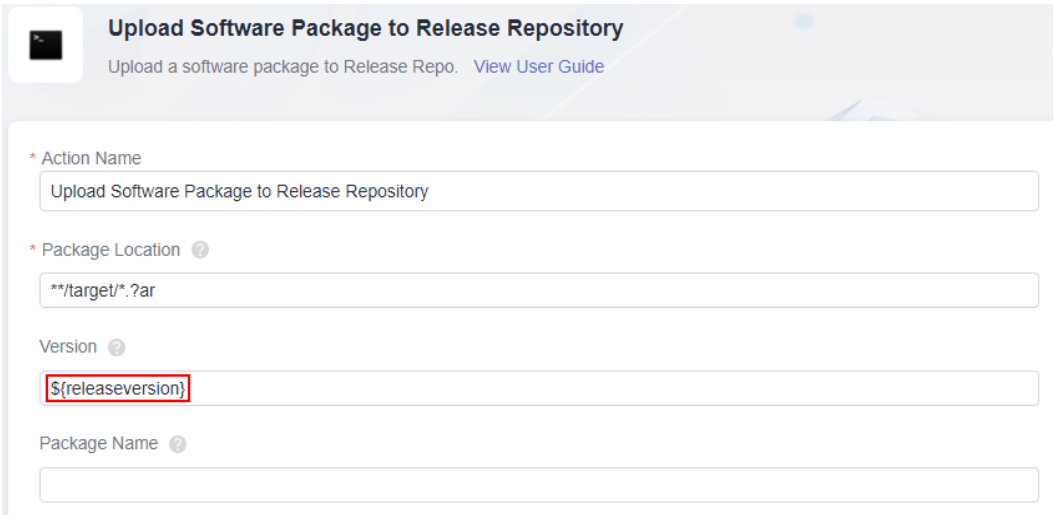
Step 2 On the **Parameters** tab page, add the **releaseversion** parameter, set the default value, and enable **Runtime Settings**.

Figure 5-5 Creating a build task parameter



Step 3 On the **Build Actions** tab page, select **Upload to Release Repos** and set **Release Version** as a reference parameter. After you enter **$** in the text box, a parameter list is displayed. Select the **releaseversion** parameter created in the previous step.

Figure 5-6 Referencing a build task parameter



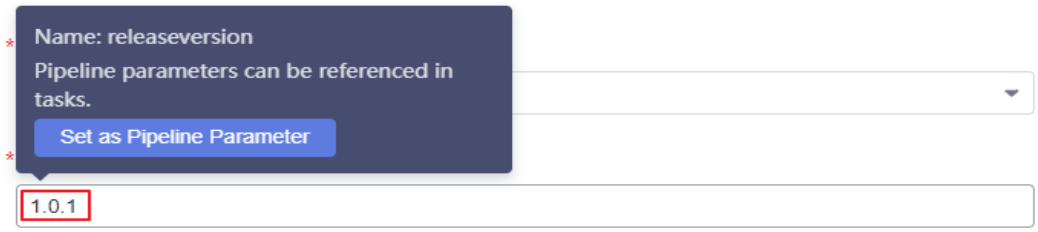
NOTE

$ will not trigger the display of parameter groups.

Step 4 Save the build job.

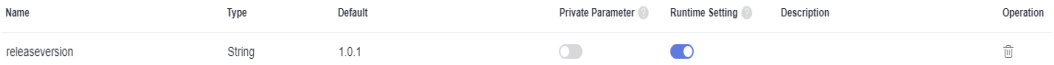
Step 5 Create a pipeline using a blank template, add the **Build** extension and select the created build job. The parameter **releaseversion** is displayed.

Figure 5-7 Configuring a build task parameter



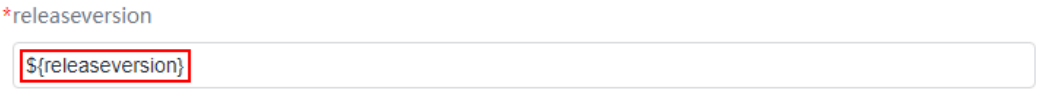
Step 6 Move the cursor to the **releaseversion** parameter to set it as a pipeline parameter. Alternatively, click **OK**, switch to the **Parameter Configuration** tab page, create the pipeline parameter **releaseversion**, set **Type** to **Auto-increment** or **String**, set a default value, and enable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

Figure 5-8 Creating a pipeline parameter



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

Figure 5-9 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 dialog box, you can check the parameter information.

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

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

----End

5.4 Configuring Pipeline Execution Plans

You can configure event triggers and scheduled tasks for a pipeline. You can also configure parallel execution policies to realize efficient resources allocation.

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 click **Edit**.

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

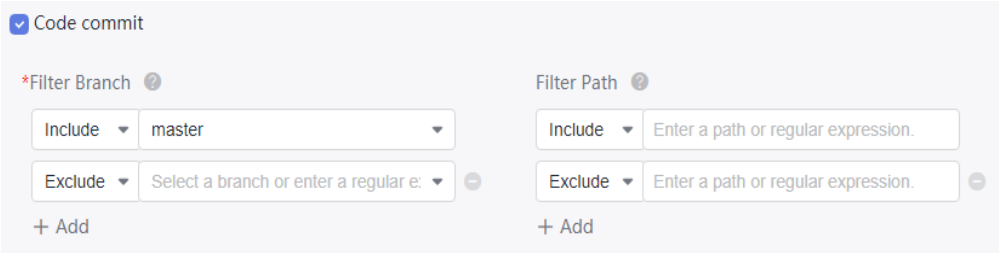
- Triggered upon code commits (Repo supported)

  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 5-10 Configuring code commit trigger

- Triggered upon merge requests (Repo)

  You can filter branches and paths by including or excluding specific ones. Target branches and paths will be monitored for merge request events such as MR creation, updating, reopening, and code merge.

  Event description:

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

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

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

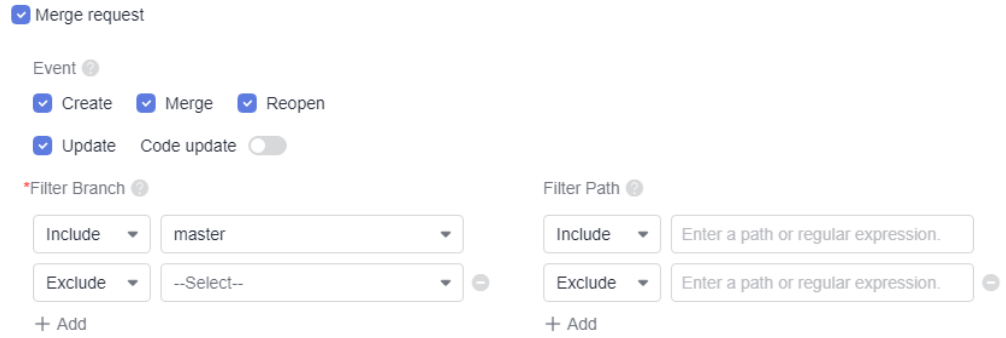
  – **Update**: triggered upon MR content, setting, or source code update. If you enable **Code update** at the same time, the pipeline will be triggered only upon source code update.

  Branch description:

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

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

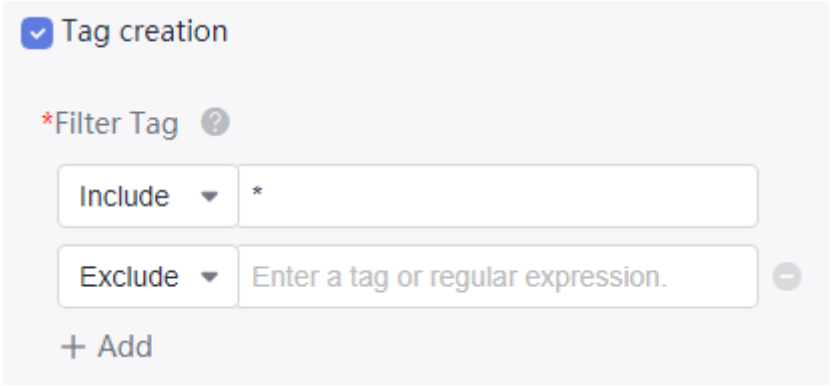
Figure 5-11 Configuring merge request trigger



- Triggered upon tag creation (Repo)

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

Figure 5-12 Configuring tag creation trigger



NOTE

- The branch is matched first, and then the path is matched. If the 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 no be triggered.

Step 4 After the configuration, save the pipeline.

----End

Configuring Scheduled Triggers

Set scheduled tasks for pipeline to execute 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 click **Edit**.

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

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

Figure 5-13 Configuring a scheduled task

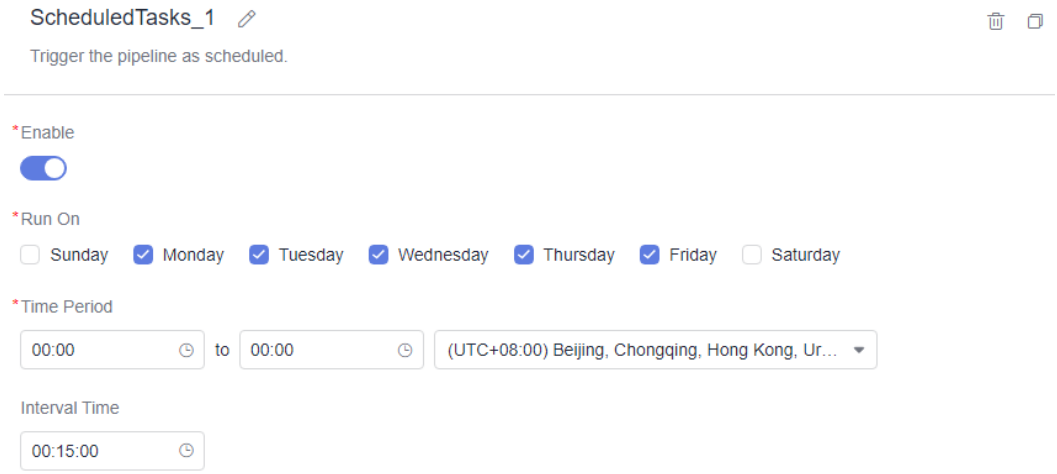


Table 5-14 Scheduled task

Parameter	Description
Run On	Select the execution date.
Time Period	Select the execution period and time zone.
Time Interval	Set the interval for triggering the pipeline.

NOTE

- You can create a maximum of 10 scheduled tasks.
- 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, save the pipeline.

----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 click **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 5-14 Configuring parallel execution

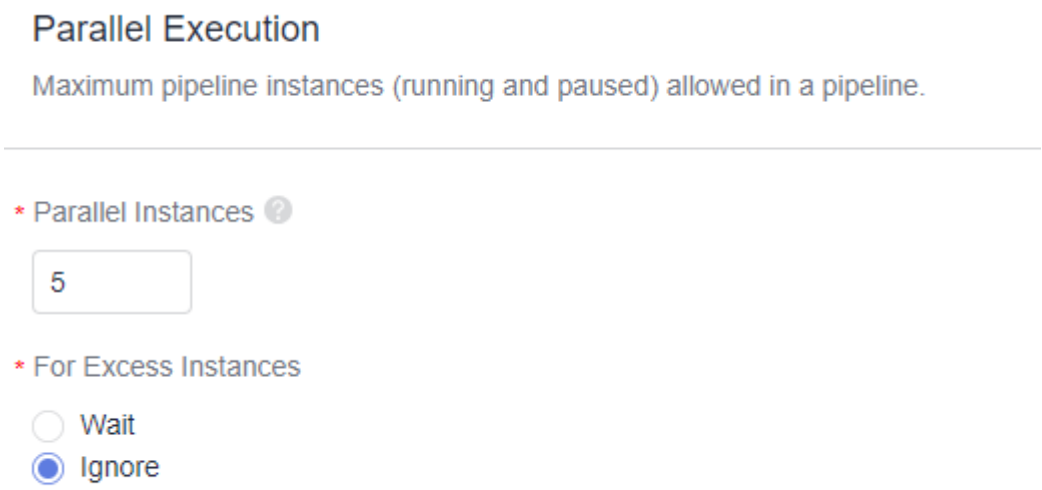


Table 5-15 Parallel execution parameters

Parameter	Description
Parallel Instances	Maximum parallel instances, which vary by your purchases and packages.
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. – Configurations of instances will not be changed once they enter the queue. • **Ignore**: Excess instances will not be executed.

Step 5 After the configuration, save the pipeline.

----End

5.5 Configuring Pipeline Permissions

You can configure permissions for a single pipeline by role or user.

- 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 creator 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.

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 click **Edit**.

Step 3 Switch to the **Permissions** page, disable **Use Project-level Permissions**, and then configure role and user permissions for the pipeline.

- Configure role permissions

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

- Configure user permissions

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

----End

5.6 Configuring Pipeline Notifications

You can configure event notifications for a pipeline.

Configuring Pipeline Internal Messages

You can configure pop-ups and emails to inform creators, executors, and favoriters of pipeline activities (deleted, failed, succeeded, updated).

Step 1 Access the CodeArts Pipeline homepage.

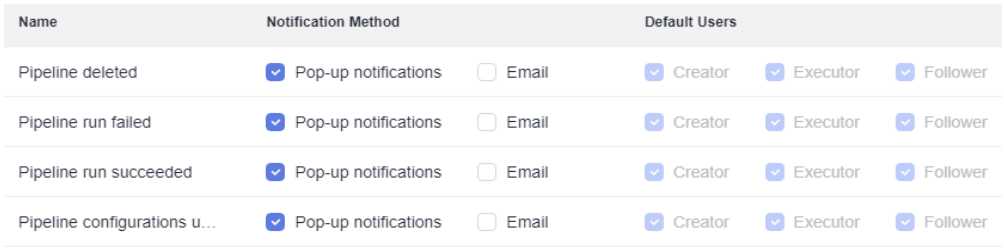
Step 2 On the pipeline list page, search for the target pipeline, click ··· in the **Operation** column, and click **Edit**.

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

Step 4 Click **Internal messages**, and select or deselect the notification methods as needed.

- By default, only pop-up notifications will be sent.
- You can click  in the upper right corner of the pipeline homepage and check the notification messages in the **Notice** dialog box.

Figure 5-15 Configuring internal messages



Step 5 After the configuration, save the pipeline.

----End

6 Grouping Pipelines

Scenarios

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

Constraints

Only project creators and project managers can manage groups.

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** dialog box is displayed.

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 also 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 operations.

Figure 6-1 Operations on multiple pipelines

- Click **Move To**. The **Move Group** dialog box is displayed. Select a group and click **Confirm**.
- Click **Execute**. In the displayed dialog box, click **OK**.
- Click **Permissions**. In the displayed dialog box, configure permissions for selected pipelines.
- Choose **More** > **Set Tag**. In the displayed dialog box, set tags for selected pipelines.
- Choose **More** > **Delete**. In the displayed dialog box, confirm the information and click **OK**

  NOTE

  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.

Executing a Pipeline

Step 1 Access the CodeArts Pipeline homepage.

Step 2 On the pipeline list page, click ▷ in the **Operation** column.

Step 3 In the displayed **Execution Configuration** dialog box, set the following parameters:

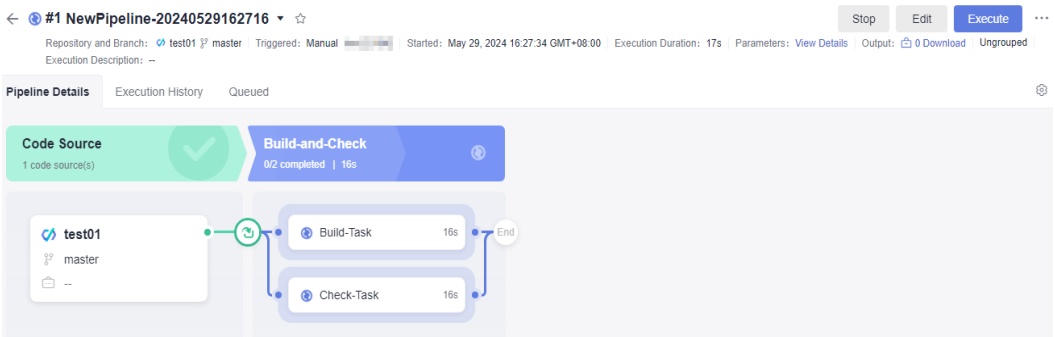
- **Code Source**: Select the branch or label of the code source.
- **Runtime Parameters**: (Optional) Set runtime parameters. For details, see Configuring Pipeline Parameters.
- **Execution Stages**: Select one or more jobs to execute. By default, all jobs are selected for execution.

  NOTE

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

Step 4 Click **Execute**. On the pipeline details page, you can view 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.
- Pipelines can be executed in parallel. You can click **Execute** to continue executing a pipeline. The maximum number of parallel pipeline executions varies based on your purchase (1 for free edition, 5 for basic edition, 10 for professional edition, and 20 for platinum edition).

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.

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.

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 most recently executed pipeline, including the execution mode, branch, latest code commit ID, and executor.
Workflow	Scheduling process and execution status (completed, failed, running, or stopped) of the pipeline.
Started & Lasted	Start time and duration of the last execution.

Parameter	Description
Operation	• Click ▷ to execute the 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 ··· to edit, clone, and preview a pipeline, check pipeline operation history (creation, editing, and failure), set tags, disable a pipeline, and delete a pipeline.

- 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 in the upper right corner to customize the pipeline list information.

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

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.

NOTE

Execution records are generated only after the first execution.

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

Table 8-1 Operations on the pipeline details page

Operation	Description
Retry	If the execution fails, you can click **Retry** in the upper right corner to continue 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. **NOTE** • 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. **NOTE** No log will be generated for jobs of **DelayedExecution** and **PipelineSuspension**.
More operations	Click ••• in the upper right corner of the page to clone, preview, disable, and delete the pipeline, and check the operation history. **NOTE** By default, only project managers, project creators, and pipeline creators can delete pipelines. You can configure permissions for different roles.

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 Configuring a Change-triggered Pipeline

Microservices are a software governance architecture. A complex software project consists of one or more microservices. Microservices in the 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.
- Diversified technologies: 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.
- The change-triggered pipeline cannot be triggered by an event or at a specified time.

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 9-1 Microservice parameters

Parameter	Description
Project	Project to which the microservice belongs. The project cannot be changed.
Microservice Name	The name can contain a maximum of 128 characters, including letters, digits, and underscores (_).
Code Source	Source of the code repository. Only **Repo** is supported. **NOTE** If you set **Code Source** to **None**, after the microservice is created, you can click its name to associate it with a code source on the **Overview** page.
Repository	Code repository associated with the microservice. Select a created code repository. **NOTE** A repository can be associated with only one microservice.
Default Branch	Default branch associated with a microservice. This branch will be used when a change-triggered pipeline is executed. **NOTE** After the change-triggered pipeline is executed, all changed feature branches will be merged into the default branch.
Language	The development language of the microservice. Available languages: **Java**, **Python**, **Node.js**, **Go**, **.Net**, **C++**, and **PHP**.
Description	Enter a maximum of 1,024 characters.

Step 4 Click **OK**. The **Overview** page is displayed.

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

NOTE

When you change the 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.

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

Table 9-2 Microservice list

Item	Description
Microservice	Microservice name.
Creator	Name of the user who created the microservice.

Item	Description
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. **NOTE** 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-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 9-3 Pipeline parameters

Parameter	Description
Project	Project to which the microservice belongs.
Name	Pipeline name, which is generated based on the creation time by default. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
Code Source	Source of the code repository. Only **Repo** is supported.
Repository	Name of the repository associated with the microservice. **NOTE** If you change the code repository of a microservice, the repository for all pipelines of the microservice will also be changed.

Parameter	Description
Default Branch	The default branch associated with the microservice. **NOTE** If you change the default branch of a microservice, the default branch for all pipelines of the microservice will also be changed.
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*. **NOTE** If you use an incorrect username or password when creating this endpoint, the pipeline will fail to run. For details, see **FAQs**.
Alias	Repository alias. Enter only letters, digits, and underscores (_) with a maximum of 128 characters. After an alias is set, a system parameter will be generated. 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  . **NOTE** A microservice can have only one change-triggered 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**, **orchestrate the pipeline**, and click **Save**.

----End

Creating a Change

You can manage changes in the microservice.

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. You can 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.

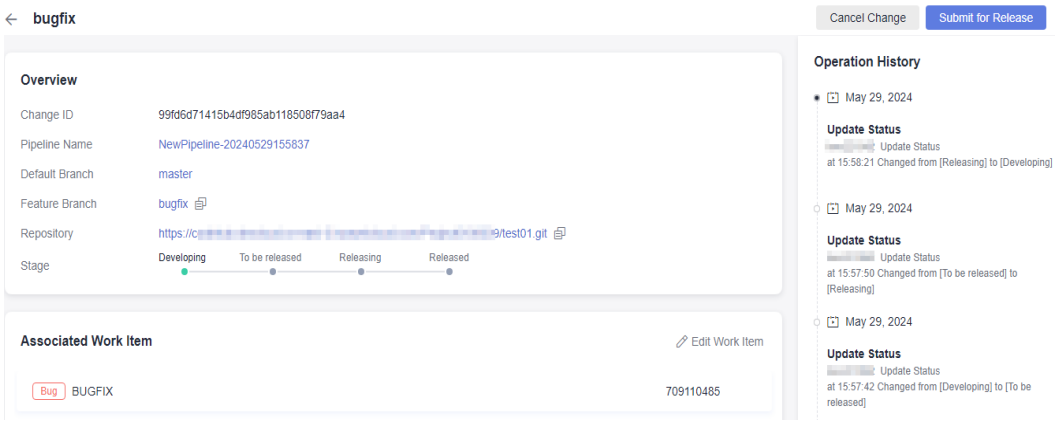
Table 9-4 Change parameters

Parameter	Description
Change Subject	Name of the change. Enter a maximum of 256 characters.
Repository	Name of the repository associated with the microservice. The repository cannot be changed.
Branch	You can pull a new branch from the default branch or associate with an existing branch. **NOTE** After the change is released through the change-triggered pipeline, the code branch will be automatically merged to the default branch of the microservice.
Associated Work Item	Select started or ongoing work items in CodeArts Req.

Step 6 Click **OK**. 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 9-1 Change details



NOTE

- 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**.

  NOTE

  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

Executing a Change-triggered Pipeline

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

Step 2 Click the **Microservices** tab.

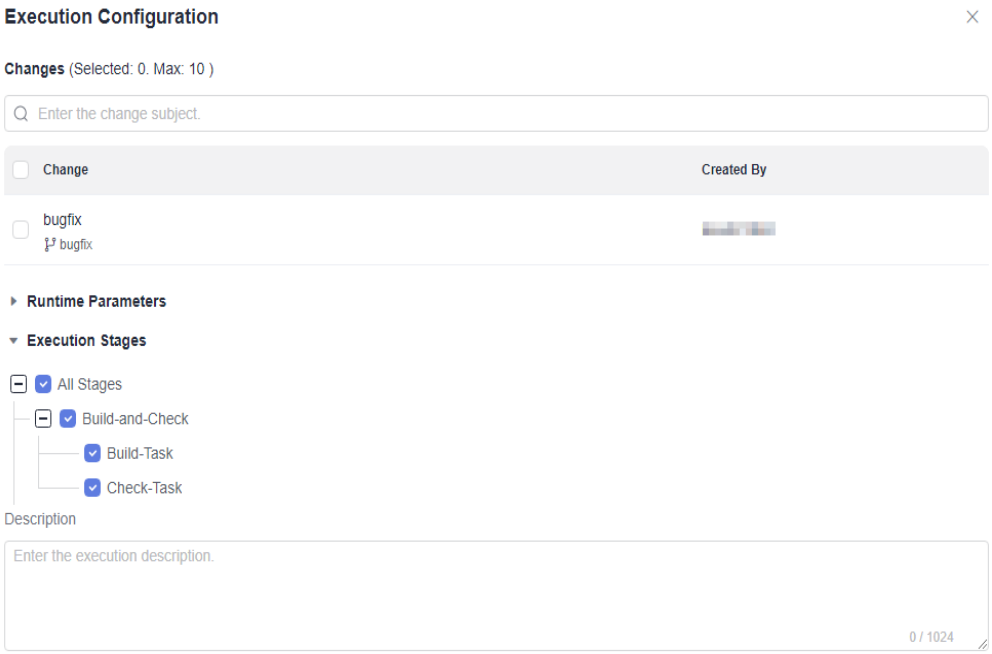
Step 3 Click a microservice name. The **Overview** page is displayed.

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 perform execution configuration.

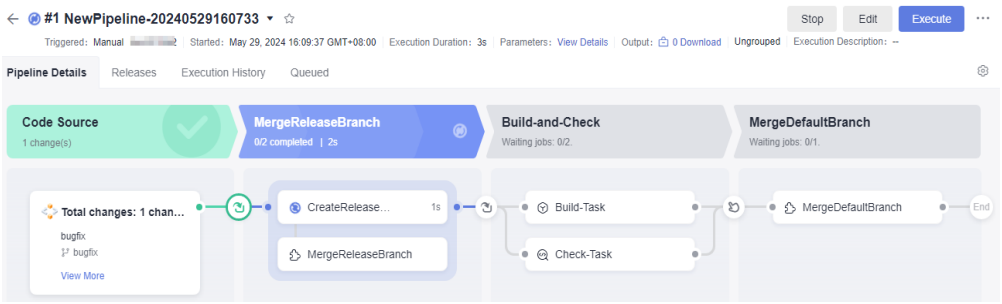
Figure 9-2 Execution configurations for a change-triggered pipeline



- **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 9-3 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, you can check the execution result.

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**.

  NOTE

  For a change in the **Releasing** status, you can exit the release only after the change-triggered pipeline execution is complete or stopped.

----End

10 Managing Pipeline Extensions

10.1 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 Extensions** 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.

10.2 Pipeline Official Extensions

CodeArts Pipeline provides official extensions as listed in **Table 10-1**.

Table 10-1 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.
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 code 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.
Normal	CreateTag	Creates and pushes tags for code repositories.
	Subpipeline	Configures and calls other pipelines in a project.
	JenkinsTask	Calls Jenkins tasks. **NOTE** 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.
	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. **NOTE** Currently, GitClone is available in LA-Mexico City2, LA-Sao Paulo1, AP-Singapore, and TR-Istanbul.
	ExecuteShellCommand	Runs shell commands.
Microservice	CreateReleaseBranch	Creates a release branch based on the default branch of a microservice. This extension is automatically configured by a change-triggered pipeline.

Type	Name	Description
	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.
Pass Conditions	Pass-Conditions-of-Standard-Policies	A standard extension policy for gate control.

10.3 Customizing Extensions on the GUI

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 10-2**.

Table 10-2 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.

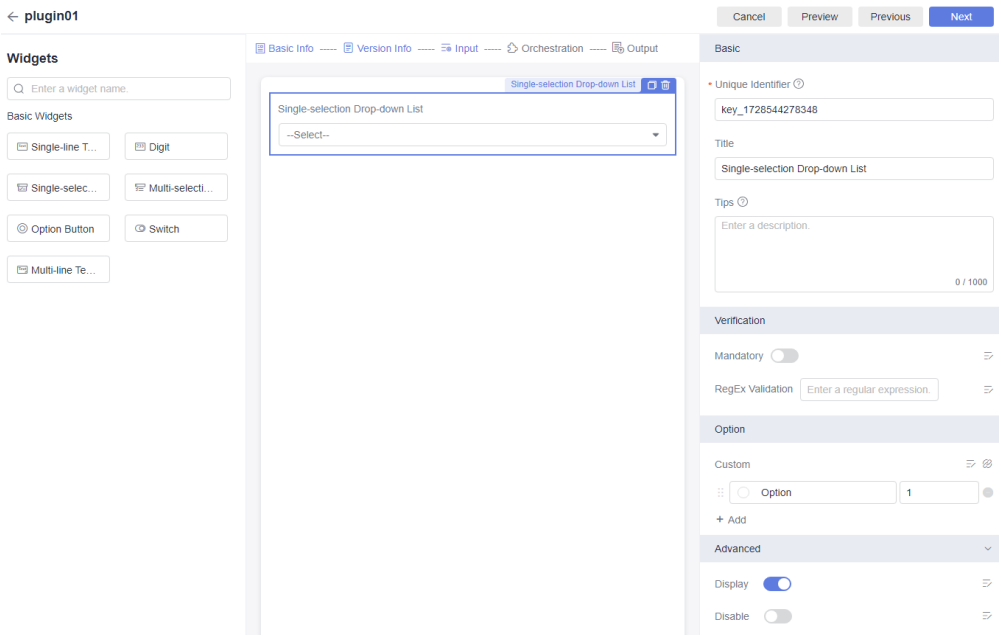
NOTE

- Version of the extension, in X.X.X format. Each digit ranges from 0 to 99.
- Version information of the extension cannot be modified later.

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

You can drag and drop widgets to generate visual forms, which can 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 10-1 Orchestrating widgets



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

Table 10-3 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

Category	Parameter	Description	Widget
	Tips	Tooltip of the widget. Enter no more than 1,000 characters.	All
	Placeholder	Message displayed in the text box. For example, input specifications.	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 10-4**. • 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 set display conditions.	All
	Disable	Whether the widget is disabled (not disabled by default). You can click ✎ on the right and configure the disable conditions.	All

Table 10-4 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.

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

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

  Enter shell commands. Commands will be executed when a pipeline call the extension.

  NOTE

  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 displayed page, click **Add Configuration** to configure **output** or **metric** information.

- **output**: output data together with shell commands.

- **metric**: output metric thresholds. The thresholds information can be referenced in an extension and finally applied to pipelines.

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 by 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 10-5 Pipeline environment variables

Variable	Description
STEP_NAME	Step name of the pipeline.
STEP_ID	Step ID of the pipeline.
PLUGIN_VERSION	Version of the extension.
PIPELINE_ID	Pipeline ID.
PIPELINE_RUN_ID	Pipeline execution ID.
PLUGIN_NAME	Extension name.

Variable	Description
PROJECT_ID	Project ID.
JOB_ID	Job ID of 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.

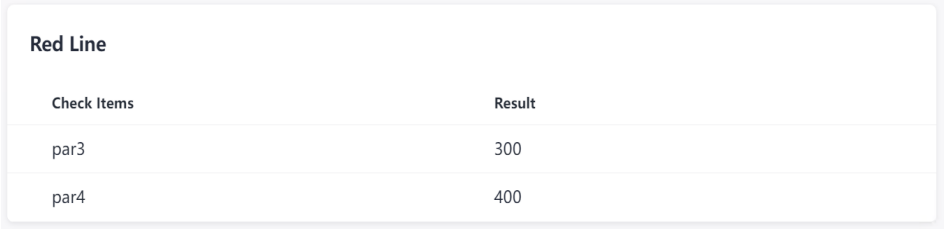
  Table 10-6 Output files

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. **NOTE** Only extensions of the check type can display the result.
${RESULT_MSG_PATH}/${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.

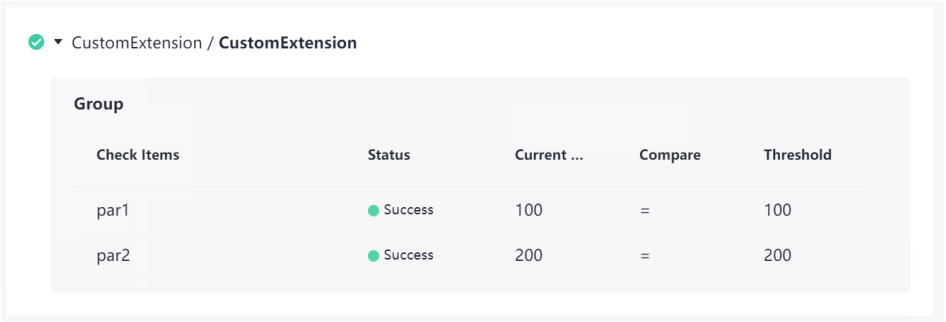
  Example: **par1** and **par2** for pass conditions; **par3** and **par4** for task result display. The sample code is as follows:

```
# 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 view the check status.



10.4 Creating an Extension by Uploading an Extension Package

Preparing an Extension Package

Extension package

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
  |         | -- resources.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
```

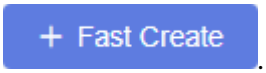
Notes:

- 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.

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 10-7**.

Table 10-7 Extension information

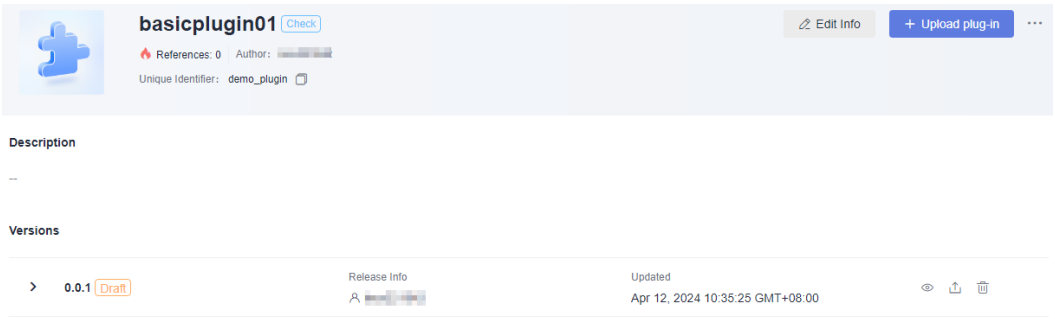
Parameter	Description
Icon	Icon of the extension. Upload an image in PNG, JPEG, or JPG format, with a file size no more than 512 KB (recommended: 128 x 128 pixels). If no image is uploaded, the system generates an icon.
Name	The extension name displayed in the extension platform. Enter only spaces, letters,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

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 10-2 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 Save and execute the pipeline. 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": []
}
```

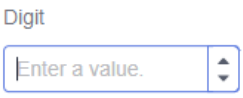
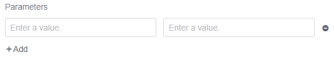
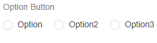
The parameters of **codearts-extension.json** are described in the following table.

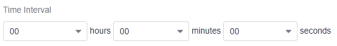
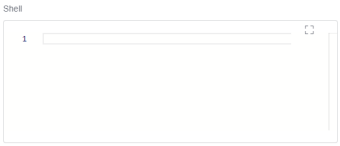
Table 10-8 Parameters

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.

Supported inputs are listed in the following table.

Table 10-9 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

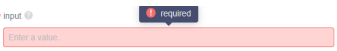
Type	Widget	Example	extendProp
timeInterval	Time Interval		• visibleConditions • disabledConditions
shell	Shell		• visibleConditions • disabledConditions
endpoint:*${module_id}*	Endpoint		• visibleConditions • disabledConditions

inputs fields are listed in the following table.

Table 10-10 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.
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. 

Field	Description	Mandatory	Remarks

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 10-11**.

extendProp functions are listed in the following table.

Table 10-11 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

Field	Description	Mandatory	Remarks

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}]`
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
			remoteQureyField: '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
			}

11 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 11-1 Service endpoints

Type	Scenario
Docker repository	Connects to a Docker image repository to obtain images for CodeArts Deploy.
Jenkins	Connects to Jenkins to execute Jenkins tasks in pipelines.
Kubernetes	Connects to a Kubernetes cluster to deliver deployment tasks.
nexus repository	Connects to a third-party private Maven repository to obtain file information for build tasks.
Git repository	Connects to a third-party Git repository to obtain branch information for CodeArts Pipeline and CodeArts Build.
GitHub	Connects to a GitHub account to obtain its repository and branch information for CodeArts Pipeline and CodeArts Build.
IAM user	Delegates your AK/SK to an IAM user so that the user can obtain a token to perform tasks that require higher permissions.

Type	Scenario
CodeArts Repo HTTPS	Authorizes 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	Connects to a GitLab repository to obtain branch information for CodeArts Pipeline and CodeArts Build.

Prerequisites

- You have the endpoint edit permission for the target CodeArts project.
- The third-party service to connect can be accessed from the public network without restrictions.

Creating a Docker Repository Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name to access the project.

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

Step 4 Click **Create Endpoint** and choose **Docker repository** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-2 Creating a Docker repository service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Repository Address	Yes	Address of the Docker repository to connect. HTTP and HTTPS addresses are supported.
Username	Yes	Username of the Docker repository to connect. Enter a maximum of 256 characters.
Password	Yes	Password of the Docker repository to connect. Enter a maximum of 256 characters.

Step 6 Check the new service endpoint.

----End

Creating a Jenkins Service Endpoint

NOTE

Currently, Jenkins service endpoints are not supported in **LA-Santiago** and **TR-Istanbul**.

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **Jenkins** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-3 Creating a Jenkins service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Server URL	Yes	Address of the Jenkins service to connect. The address can be in format "**http://***ip*:*Port*" or "**https://***ip*:*Port*".
Username	Yes	Username of the Jenkins service to connect. Enter a maximum of 300 characters.
Password	Yes	Password of the Jenkins service to connect. Enter a maximum of 300 characters.

Step 6 Check the new service endpoint.

----End

Creating a Kubernetes Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **Kubernetes** from the drop-down list.

Step 5 Configure the following information and click **Verify and Confirm**.

Table 11-4 Creating a Kubernetes service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Kubernetes URL	Yes	Server address of the cluster to connect. Obtain it by searching for **server** in **kubeconfig.json**.
Kubeconfig	Yes	Content of the cluster's **kubeconfig.json** file.

Step 6 Check the new service endpoint.

----End

Creating a Nexus Repository Service Endpoint

NOTE

Currently, Nexus repository service endpoints are not supported in **LA-Santiago** and **TR-Istanbul**.

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **nexus repository** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-5 Creating a Nexus repository service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Repository Address	Yes	Address of the Nexus repository to connect. HTTP and HTTPS addresses are supported.
Username	Yes	Username of the Nexus repository to connect. Enter a maximum of 300 characters.
Password	Yes	Password of the Nexus repository to connect. Enter a maximum of 300 characters.

Step 6 Check the new service endpoint.

----End

Creating a Git Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **Git repository** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-6 Creating a Git service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Git Repository URL	Yes	HTTPS address of the Git repository to connect.
Username	No	Username of the Git repository to connect. Enter a maximum of 300 characters.
Password or Access Token	No	Password of the Git repository to connect. Enter a maximum of 300 characters.

Step 6 Check the new service endpoint.

----End

Creating a GitHub Service Endpoint

NOTE

Currently, GitHub service endpoints are not supported in **LA-Santiago**.

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **GitHub** from the drop-down list.

Step 5 Configure the following information and click **Authorize and Confirm**.

Table 11-7 Creating a GitHub service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Authentication Mode	Yes	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, see **GitHub official website**.

Step 6 Check the new service endpoint.

----End

Creating an IAM User Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **IAM user** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-8 Creating an IAM user service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Access Key Id	Yes	AK obtained on the **My Credentials** page. For details, see **Access Keys**.

Parameter	Mandatory	Description
Secret Access Key	Yes	SK obtained on the **My Credentials** page. For details, see **Access Keys**.

Step 6 Check the new service endpoint.

----End

Creating a CodeArts Repo HTTPS Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **CodeArts Repo HTTPS** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-9 Creating a CodeArts Repo HTTPS service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
CodeArts Repo URL	Yes	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	No	HTTPS username of the CodeArts Repo repository to connect. Click the username on the navigation bar and choose **This Account Settings**. Obtain the username on the **Repo** > **HTTPS Password** page.
Password	No	HTTPS password of the CodeArts Repo repository to connect. Click the username on the navigation bar and choose **This Account Settings**. Obtain the password on the **Repo** > **HTTPS Password** page.

Step 6 Check the new service endpoint.

----End

Creating a GitLab Repository Service Endpoint

Step 1 Go to the CodeArts homepage.

1. Log in to the **CodeArts console**, click , and select a region.
2. Click **Access Service**.

Step 2 On the CodeArts homepage, click a project name.

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

Step 4 Click **Create Endpoint** and choose **GitLab repository** from the drop-down list.

Step 5 Configure the following information and click **Confirm**.

Table 11-10 Creating a GitLab repository service endpoint

Parameter	Mandatory	Description
Service Endpoint Name	Yes	Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
GitLab URL	Yes	HTTPS address of the GitLab repository to connect.
Username	No	Username of the GitLab repository to connect.
Access Token	No	Access token for the GitLab repository to connect. For details, see **GitLab official website**.

Step 6 Check the new service endpoint.

----End

12 CodeArts Release User Guide

12.1 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.

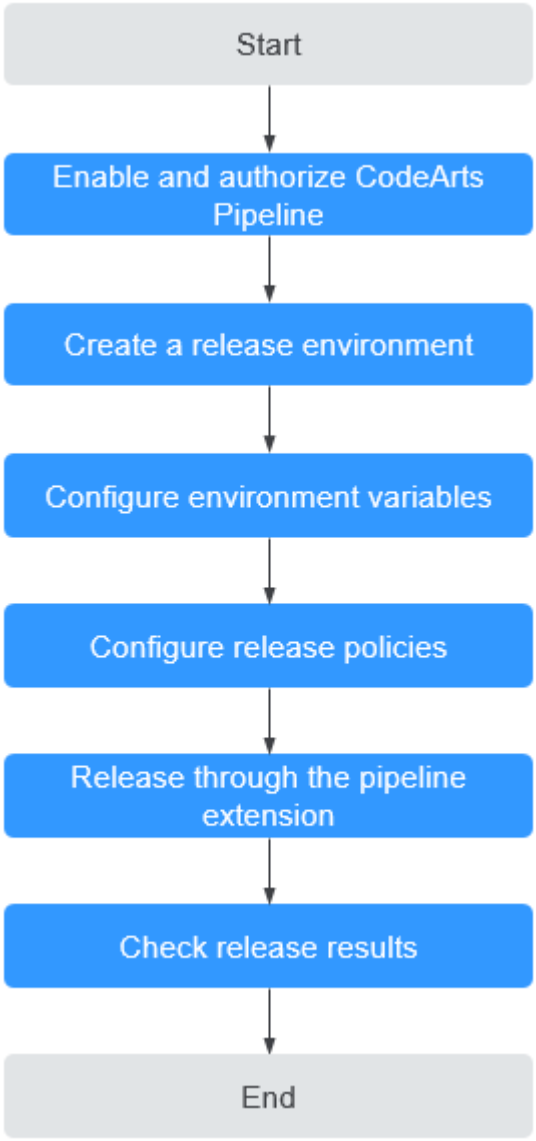
CodeArts Release has the following features:

- 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 12-1 Operation process



12.2 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 12-1**.

Table 12-1 Parameter description

Parameter	Description
Project	Project to which the environment belongs. The project cannot be changed.
Environment Name	Unique environment identifier within the microservice. The name cannot be changed once the environment was created. 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 12-2 Environment information

Parameter	Description
Resource Type	Resource types associated with the environment.
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

12.3 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 12-3 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 12-4 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.
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.

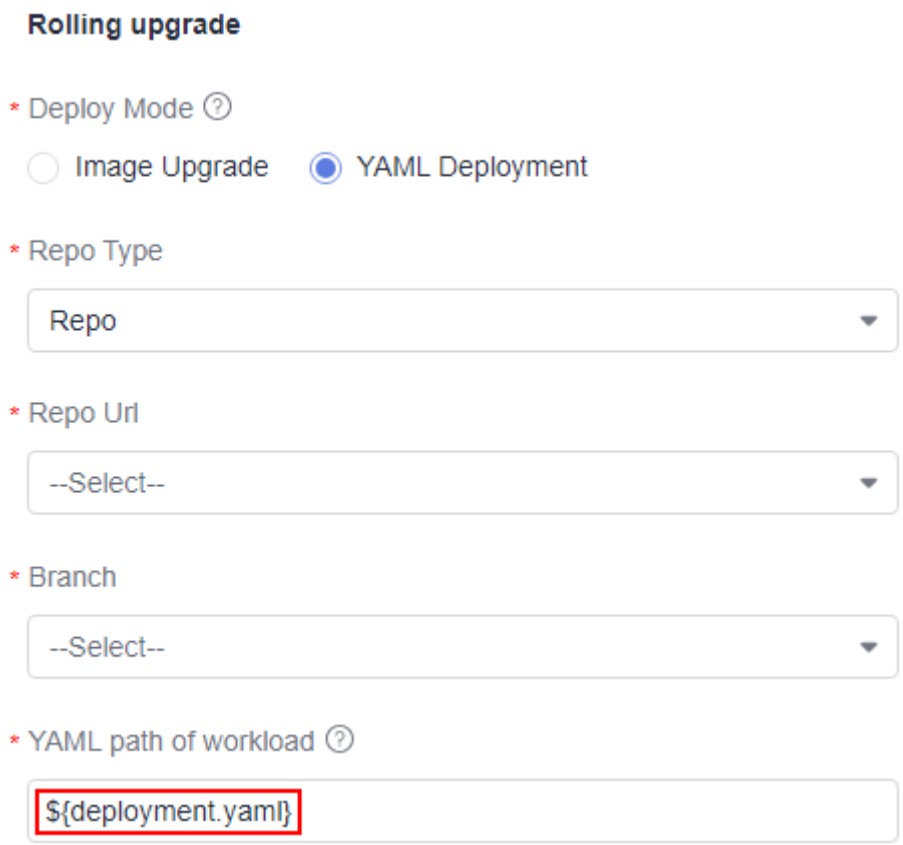
----End

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 12-2 Referencing an environment variable

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

Figure 12-3 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
```

12.4 Configuring an Environment Release Policy

Creating a Policy

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

Step 1 Access the environment list page.

Step 2 Click an environment name.

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.

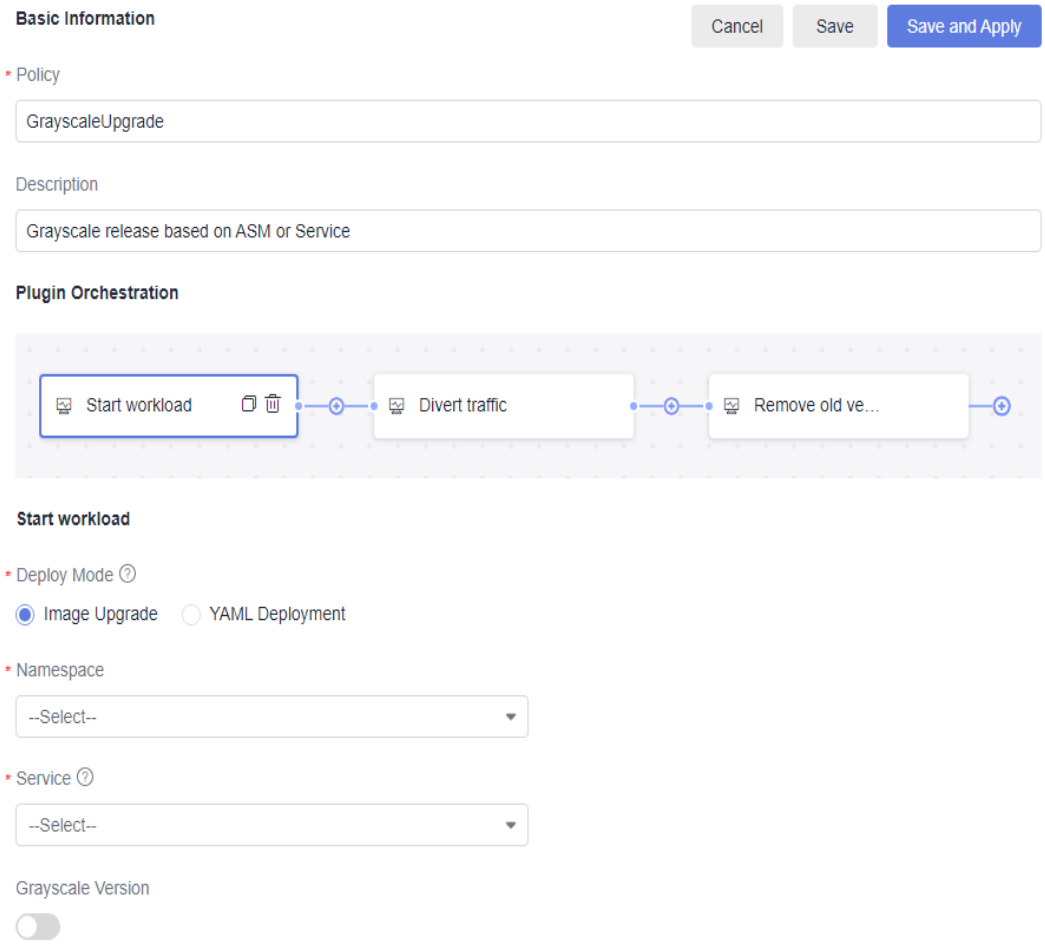


Table 12-5 Policy parameters

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 five extensions (**Rolling 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.

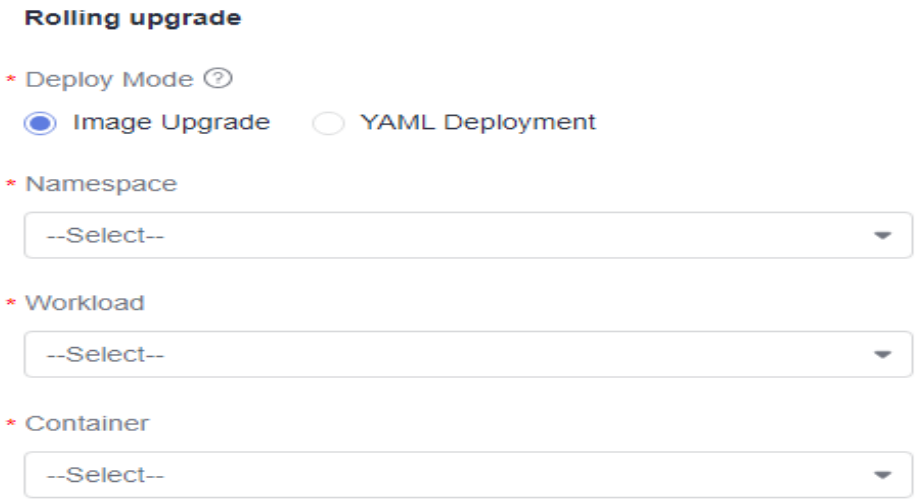


Table 12-6 Parameter description

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

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

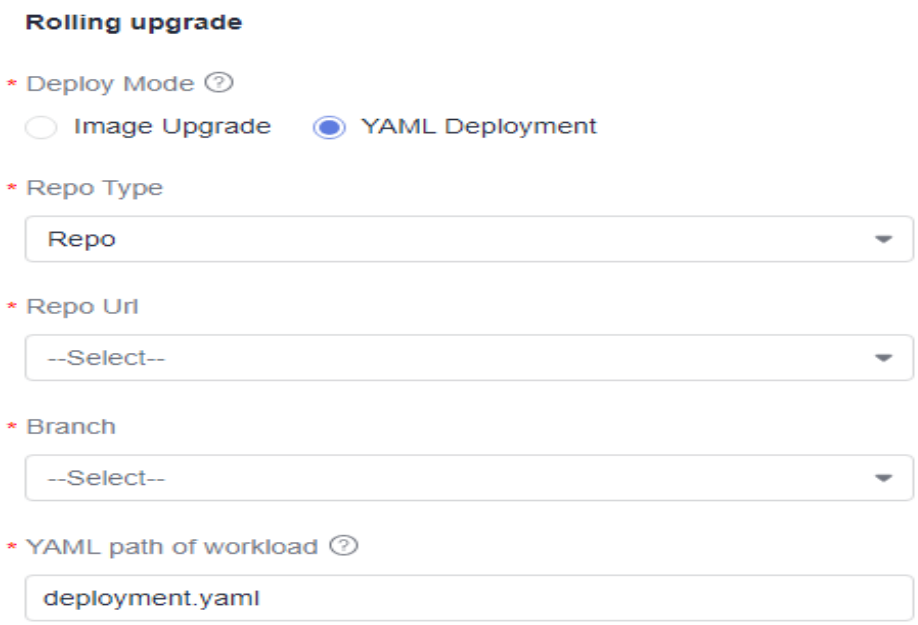


Table 12-7 Parameter description

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.

- **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.

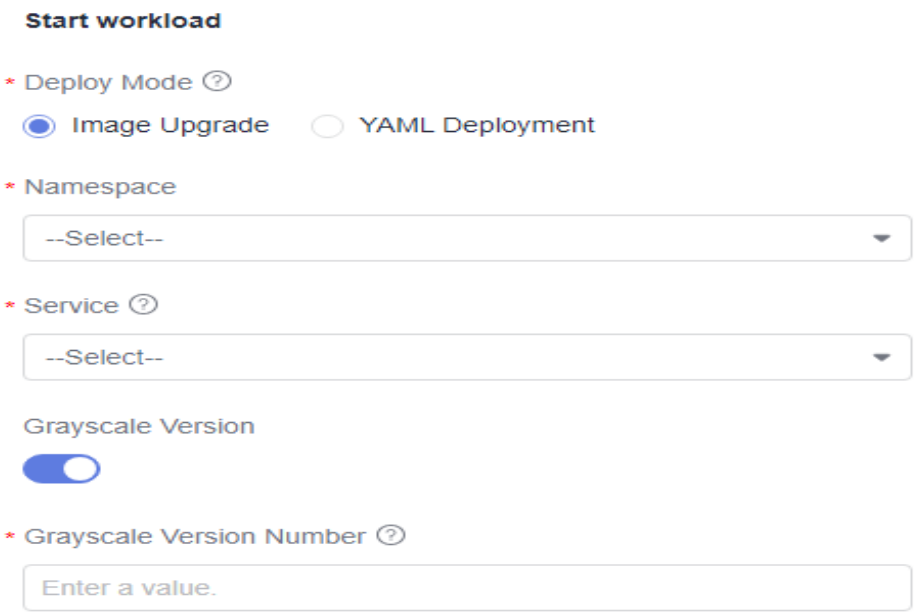


Table 12-8 Parameter description

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.

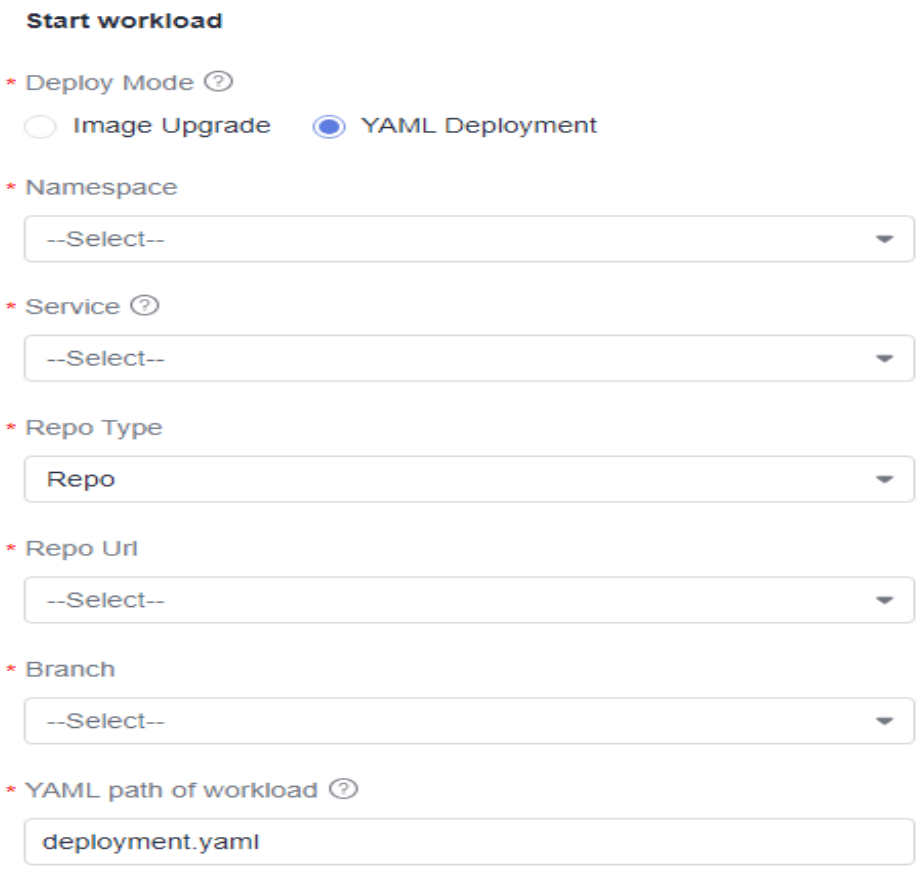


Table 12-9 Parameter description

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.
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.

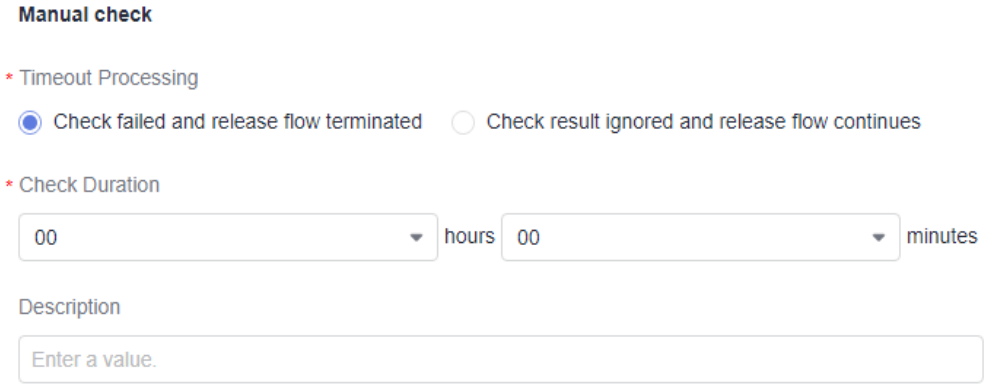


Table 12-10 Parameter description

Parameter	Description
Timeout Processing	• 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.

12.5 Releasing an Environment Through the CloudNativeRelease Extension

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

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 12-11**.

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

Figure 12-4 Configuring CloudNativeRelease

Table 12-11 Parameter description

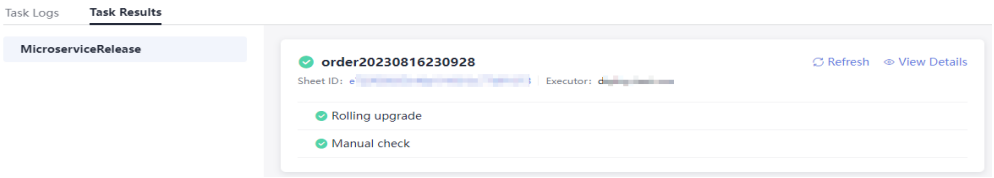
Parameter	Description
Name	Extension name Enter only letters, digits, underscores (_), hyphens (-), commas (,), semicolons (;), colons (:), periods (.), slashes (/), parentheses (), and spaces, with a maximum of 128 characters.
Environment Level	Release environment type. Available environment types: development, test, pre-production, and production.
Environment	Release environment. For details, see **Creating a Release Environment**.

Parameter	Description
Artifact Path	Image path for microservice deployment. 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 12-5 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

12.6 Checking the Environment Release Result

Step 1 Access the release environment list page.

Step 2 Click an environment name.

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. The rollback ticket page will be displayed.

> **NOTE**
>
> 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 12-6 Atomic extension release

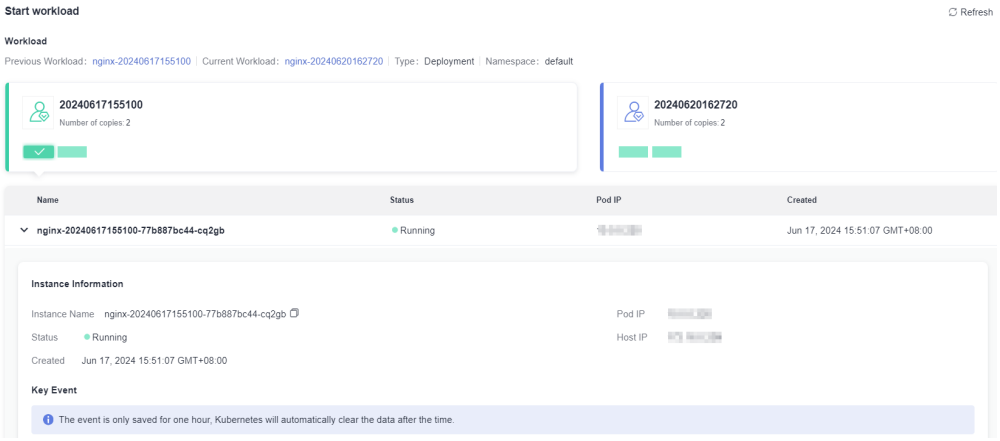


Table 12-12 Atomic extension release

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.

Extension	Release Information
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

13 Viewing 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	pipelineTask	executePipelineTask
Editing a pipeline	pipelineTask	updatePipelineTask
Creating a pipeline	pipelineTask	createPipelineTask
Deleting a pipeline	pipelineTask	deletePipelineTask
Stopping a pipeline	pipelineTask	stopPipelineTask

Querying Real-Time Traces

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

