CodeArts Pipeline

User Guide

Issue 01

Date 2025-09-10

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 CodeArts Pipeline Usage

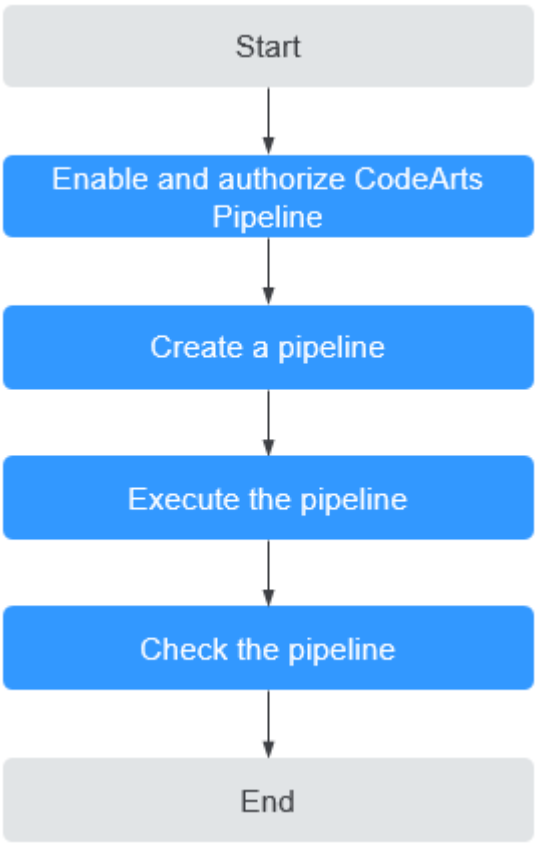
CodeArts Pipeline is a visual platform for automating task scheduling. You can use Pipeline to automate tasks in services like **CodeArts Build**, **CodeArts Check**, **CodeArts TestPlan**, and **CodeArts Deploy**.

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

- Tenant-level permissions

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

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

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

     Table 2-2 Pipeline policies

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

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

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

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

NOTE

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

- Project-level permissions

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

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

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

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

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

e. Click a project name to access the project.

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

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

NOTE

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

Pipeline permissions

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

Table 2-3 Project-level permissions

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

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

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

Policy permissions

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

Table 2-4 Project-level policy permissions

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

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

Microservice permissions

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

Table 2-5 Project-level microservice permissions

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

Change permissions

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

Table 2-6 Project-level change permissions

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

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

Environment permissions

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

Table 2-7 Project-level development environment permissions

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

Table 2-8 Project-level test environment permissions

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

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

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

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

Table 2-10 Project-level production permissions

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

Parameter group permissions

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

Table 2-11 Project-level parameter group permissions

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

- Resource-level permissions

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

  Role permissions

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

  User permissions

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

  Use project-level permissions

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

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

3 Accessing CodeArts Pipeline

This section describes how to access CodeArts Pipeline.

Accessing Through the Homepage

Step 1 Log in to the Huawei Cloud console.

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

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

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

----End

Accessing Through a Project

Step 1 Log in to the Huawei Cloud console.

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

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

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

Step 5 Click a project name to access the project.

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

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

----End

4 Creating a Pipeline

4.1 Creating a Pipeline with the GUI

Preparations

- **Create a project**.
- If you use a CodeArts Repo repository, **create a code repository**.

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 code source: Repo: provides comprehensive code hosting services for enterprises and Git-based online code hosting services for software developers. • Third-party code source – Git: After connecting to a Git repository, you can obtain its branch information. **NOTE** • If you do not need to associate the pipeline with a code repository, you can select **None**. In this case, executing a job that should be associated with a repository will result in an error. For details, see **FAQs**.
Orchestration Method	If you select **Repo** as the code source, you can choose either of the following ways to orchestrate a pipeline. • Graphical: Uses the GUI to clearly display serial and parallel jobs. • YAML: Uses YAML (One YAML file can be used for multiple pipelines). Syntax auto-completion and validation available. For details, see **Creating a Pipeline with YAML**.
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. 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}*. Enter only letters, digits, and underscores (_) with a maximum of 128 characters.
Description	Enter a maximum of 1024 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**: You can add official, third-party, and custom extensions to a pipeline template. After jobs of 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**.
- Prepare a YAML file.

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

YAML File Example

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

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

jobs: # Define jobs included in the pipeline.
  build: # Job ID, which defines the unique identifier of the job.
    name: maven build # Job name, which is displayed on the GUI.
    steps: # Define the steps within the job.
      - name: My build step # Step name, which is displayed on the GUI.
        uses: CodeArtsBuild # Extension used for this step.
        with: # Define the extension's runtime parameters as key-value pairs. Variables defined in "env" can be referenced.
          jobId: 878b4d13cb284d9e8f33f988a902f57c # Job ID. On the job details page, copy the 32-bit string at the end of the browser URL to obtain the ID.
          artifactIdentifier: my_image
          version: ${{ env.image_version }}
  check:
    name: code check
    steps:
      - name: My check step
        uses: CodeArtsCheck
        with:
          jobId: 43885d46e13d4bf583d3a648e9b39d1e
```



```
      checkMode: full
deploy:
  name: cce deploy
  needs: # Specify that this job should run only after the listed jobs have completed.
    - build
    - check
  if: ${{ completed() }} # Specify the condition under which this job should run.
  steps:
    - name: My deploy step
      uses: CodeArtsDeploy
      with:
        jobId: 9c5a5cda6ffa4ab583380f5a014b2b31
        version: ${{ env.image_version }}
```

Creating a 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 select **Repo** to create a YAML-based pipeline.
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 write a YAML file, see **YAML Syntax**.

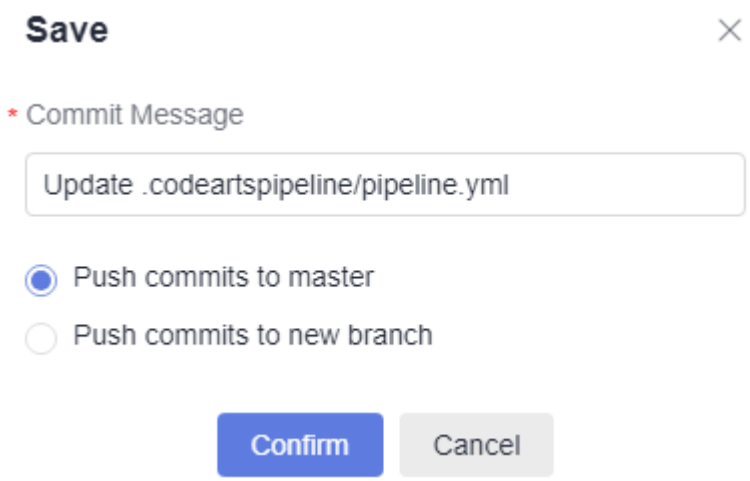
Parameter	Description
YAML File	This parameter is mandatory when **Configuration File** is set to **Existing**. Select a branch and enter the relative path of the YAML file.
Repo Endpoint	Configure an endpoint to enhance permissions for Repo. Endpoints are used for change-triggered pipelines and repository operation extensions. 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 1024 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

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

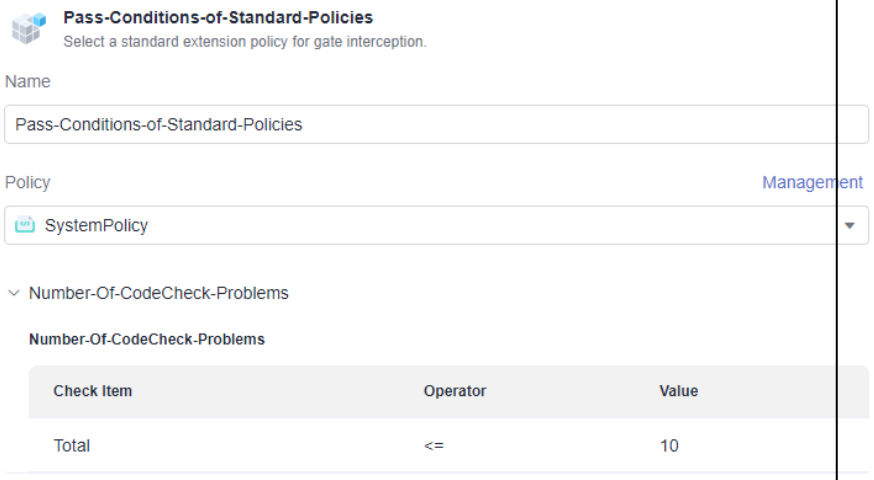
Step 1 Access the CodeArts Pipeline homepage.

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

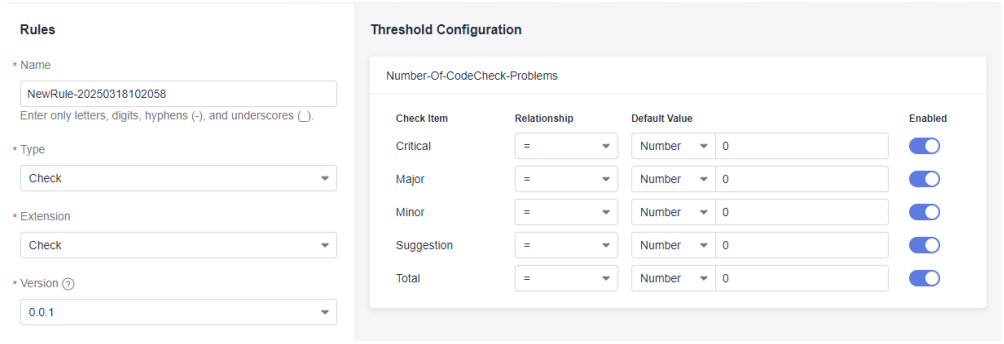


Table 5-2 Rule parameters

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

Step 1 Access the CodeArts Pipeline homepage.

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

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

Step 4 Click **Create Policy** 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.

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

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.

Configuring Pipeline Jobs

Step 1 Access the CodeArts Pipeline homepage.

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

Step 3 On the **Task Orchestration** page, click **New Job** under a stage, and select extensions from the displayed window.

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

Step 4 Configure extensions 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 task to be called. You can search for a task. If no proper task is available, create one 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.
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.

Operation	Description
Sorting extensions	Click, hold, and move an extension card to adjust the extension sequence.

Operation	Description
Configuring jobs	Configure the following information: • **ID**: The job ID must be unique. Enter only letters, digits, hyphens (-), and underscores (_) with a maximum of 128 characters. • **Execution Host**: You can use the built-in executors or create custom ones. – **Built-in executor**: provided by CodeArts Pipeline with out-of-the-box availability. – **Custom executor**: allows you to configure tools and running environments as needed. Before using custom executors, add an agent pool. For details, see **Agent Pools**. **NOTE** You only need to configure executors for non-job-level extensions. • Select Job – **Always**: The job will always be selected for execution and cannot be canceled. – **Disabled**: The job cannot be selected for execution. – **Selected by default**: The job is selected for execution by default and can be modified. – **Not selected by default**: Job is not selected for execution by default. • **Execute**: Execution conditions are the triggers for executing jobs in a pipeline. – **Even when previous job is not selected**: The current job is executed 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 *${{value}}* and can be any combination of contexts, operators, functions, or literals. Example: If the current job is executed regardless of whether the previous job (ID: **job_1**) succeeded or failed, the expression can be as follows: `${{ jobs.job_1.status == 'COMPLETED' || jobs.job_1.status == 'FAILED' }}` For details, see **Configuring Expressions**.

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

Configuring an Execution Plan

You can configure an execution plan in advance to improve the execution efficiency. You can select an execution plan during scheduled task execution or manual execution.

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

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

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

Step 4 Click **Save**. You can edit, clone, or delete the execution plan.

----End

Configuring a Job Template

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

Step 2 Click **Create Template**. In the displayed window, set parameters.

Step 3 Configure jobs for the template by referring to the following table.

Table 5-7 Configuring a job template

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 task to be called. You can search for a task. If no proper task is available, create one 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.
Deleting an extension	Move the cursor to an extension card, click ⋮, and select **Delete** to delete the extension.
Replacing an extension	Move the cursor to an extension card, click ⋮, and select **Replace** to replace the extension. Or, click **Replace 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	Configure the following information: • Basic information: – **Name**: name of the job template. – **Description**: description of the job template. A maximum of 1,024 characters are allowed. • Job ID: The job ID should be unique. Enter only letters, digits, hyphens (-), and underscores (_) with a maximum of 128 characters. **NOTE** During pipeline orchestration, you can only edit the job ID in a job template. • **Execution Host**: You can use the built-in executors or create custom ones. – **Built-in executor**: provided by CodeArts Pipeline with out-of-the-box availability. – **Custom executor**: allows you to configure tools and running environments as needed. Before using custom executors, add an agent pool. For details, see **Agent Pools**. **NOTE** You only need to configure executors for non-job-level extensions. • Select Job – **Always**: The job will always be selected for execution and cannot be canceled. – **Disabled**: The job cannot be selected for execution. – **Selected by default**: The job is selected for execution by default and can be modified. – **Not selected by default**: Job is not selected for execution by default. • **Execute**: Execution conditions are the triggers for executing jobs in a pipeline. – **Even when previous job is not selected**: The current job is executed 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.

Operation	Description
	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' }}` For details, see **Configuring Expressions**.

Step 4 After the configuration is complete, click **OK**.

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

----End

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. You can create a maximum of 100 custom parameters.
- **Parameter Groups**: You can associate all pipelines in the project with a parameter group. You can create a maximum of 5 parameter groups and add a maximum of 20 custom parameters to a 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 parameter with the same name exists, the parameter priority is as follows: predefined parameters > custom parameters > parameter groups.
- If a pipeline is associated with multiple parameter groups 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, the predefined and custom parameter list will be displayed.

Predefined Parameters

Table 5-8 Predefined parameters

Parameter	Description
PROJECT_ID	ID of the project to which the current pipeline belongs.
PIPELINE_ID	ID of the current pipeline.
PIPELINE_NUMBER	Pipeline number.
COMMIT_ID	The last commit ID before execution.
COMMIT_ID_SHORT	The last short commit ID before execution.
TIMESTAMP	Pipeline execution timestamp. For example, **20211222124301**.
PIPELINE_TRIGGER_TYPE	Pipeline trigger type, which includes Manual, Scheduler, Rollback, and Webhook (CreateTag, Note, Issue, MR, and Push).
PIPELINE_NAME	Pipeline name.
REPO_URL	Code repository address (HTTPS).
EXECUTE_USER	The user who executes the pipeline.
EXECUTE_USER_ID	Executor ID.
EXECUTE_USER_NAME	Executor name.
EXECUTE_USER_NICKNAME	Executor alias.
PASS_CONDITIONS_LINK	Pipeline execution details link.

Parameter	Description
PIPELINE_RUN_ID	Pipeline run ID.
MERGE_ID	Merge request ID.
WEBHOOK_PAYLOAD	Webhook request payload information.
Repo01_REPOSITORY_NAME	Repository name.
Repo01_SOURCE_BRANCH	The source branch name used by the repository when a merge request triggers the pipeline.
Repo01_TARGET_BRANCH	Name of the target branch for repository operations.
Repo01_TAG	Repository tag name.
Repo01_COMMIT_ID	The last commit ID before 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 5-9 Custom parameters

Parameter	Description
Name	The specified name cannot be the same as that of a predefined parameter. Enter only letters, digits, and underscores (_) with a maximum of 128 characters.
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. It can be left blank. • **Auto-Increment**: The value contains no more than 8,192 characters. 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 for you to set optional values. After that, click the **Default** drop-down list box, select or search for a 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

Click + in the **Operation** column to add a parameter. Click in the **Operation** column 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 5-2 Creating a parameter group

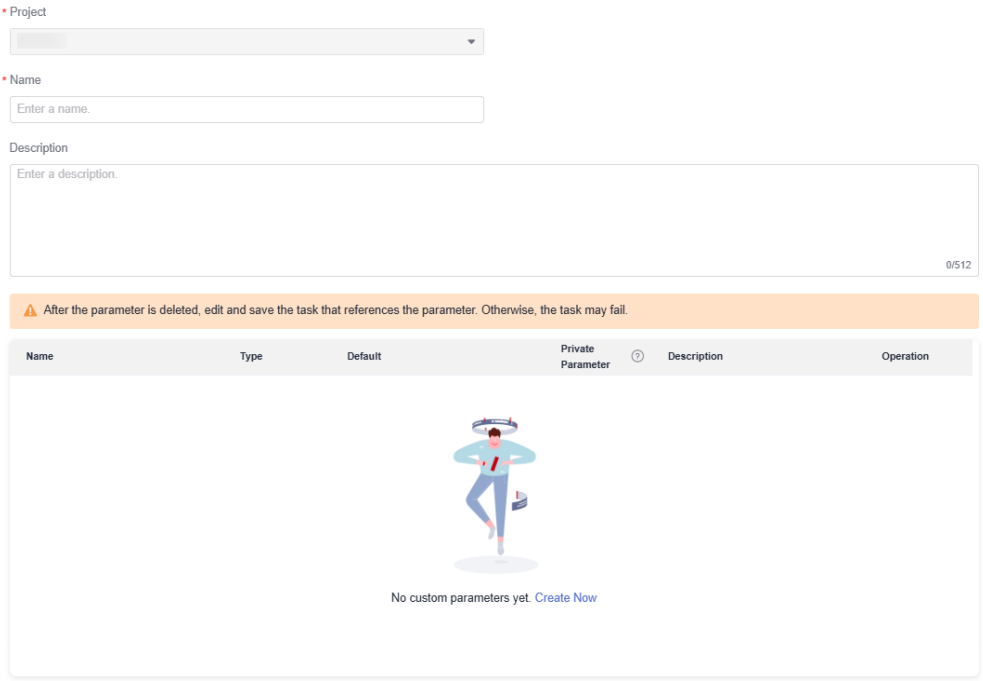


Table 5-10 Parameter group description

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**.

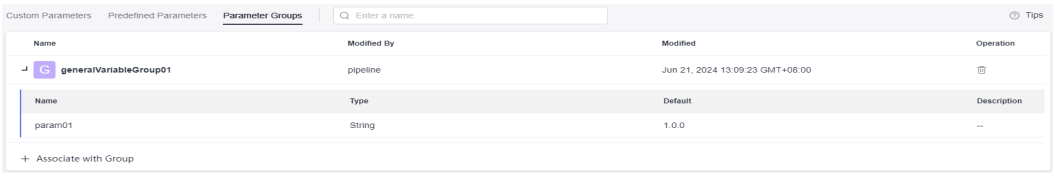
Step 4 Click **OK** to create a parameter group.

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 5-3 Associating with a parameter group



Step 7 After the configuration, save the pipeline.

----End

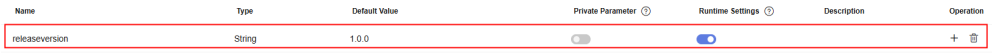
Using a Parameter in a Pipeline

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.

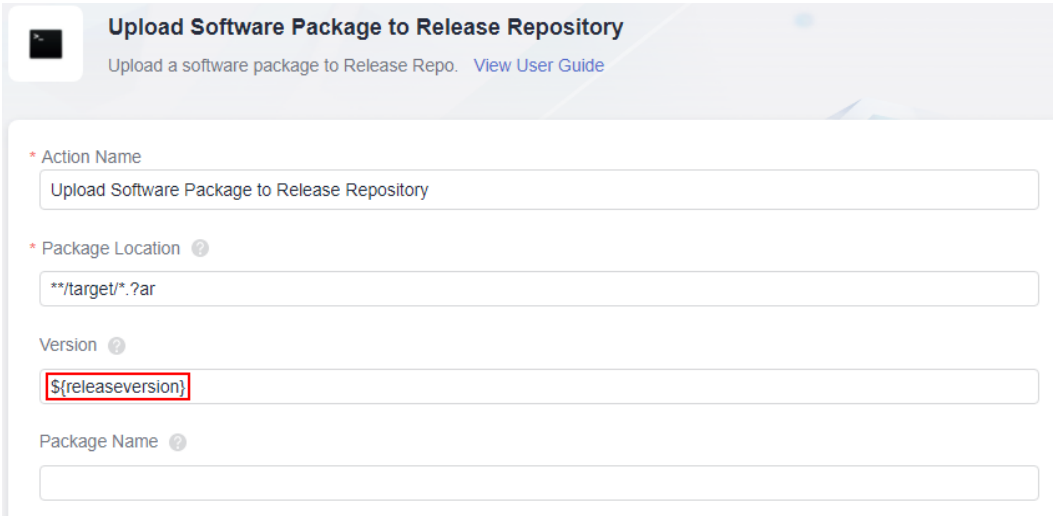
Step 2 On the **Parameters** tab page, add the **releaseversion** parameter, set the default value, and enable **Runtime Settings**.

Figure 5-4 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 5-5 Referencing a build task parameter



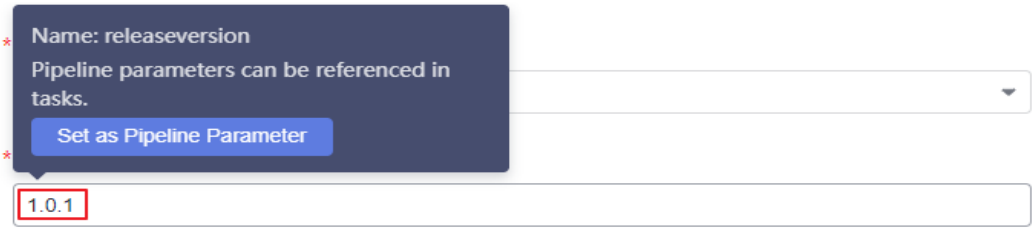
NOTE

$ will not trigger the display of parameter groups.

Step 4 Save the build job.

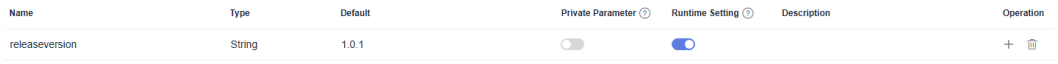
Step 5 Create a pipeline using a blank template, add the **Build** extension and select the created build job. The parameter **releaseversion** is displayed.

Figure 5-6 Configuring a build task parameter



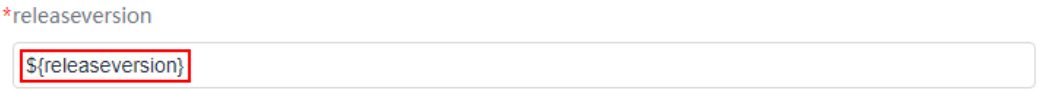
Step 6 Move the cursor to the **releaseversion** parameter to set it as a pipeline parameter. Alternatively, click **OK**, switch to the **Parameter Configuration** tab page, create the pipeline parameter **releaseversion**, set **Type** to **Auto-increment** or **String**, set a default value, and enable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

Figure 5-7 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 5-8 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, scheduled tasks, webhooks, parallel execution policies, and preemption policies for pipelines. These policies automate executions, trigger executions through third parties, allocate granular resources, and forcibly stop all running instances triggered by the same event.

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.

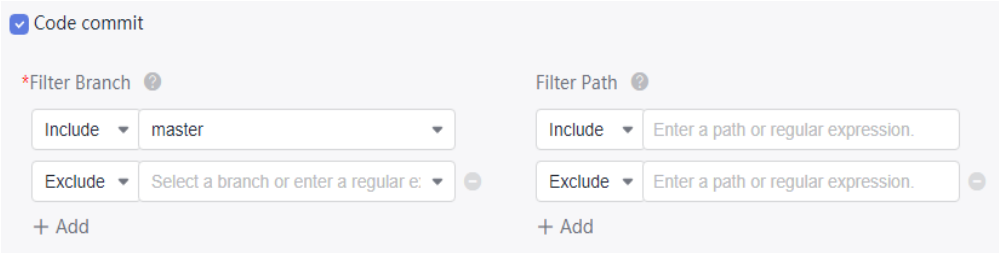
- Triggered upon code commits (Repo supported)

  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 5-9 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.

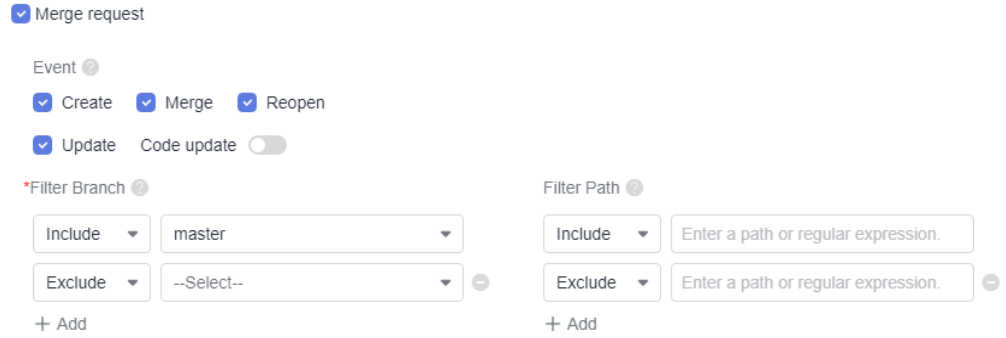
  – **Update**: triggered upon MR content, setting, or source code update. If you enable **Code update** at the same time, the pipeline will be triggered only upon source code update.

  Branch description:

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

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

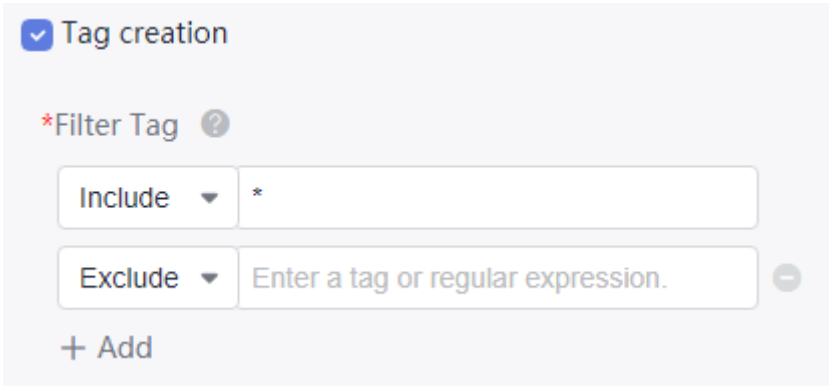
Figure 5-10 Configuring merge request trigger



- 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 5-11 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.
- Path matching covers the first 300 updated files per submission. To match beyond the 300 files, split them.

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 5-12 Configuring a scheduled task

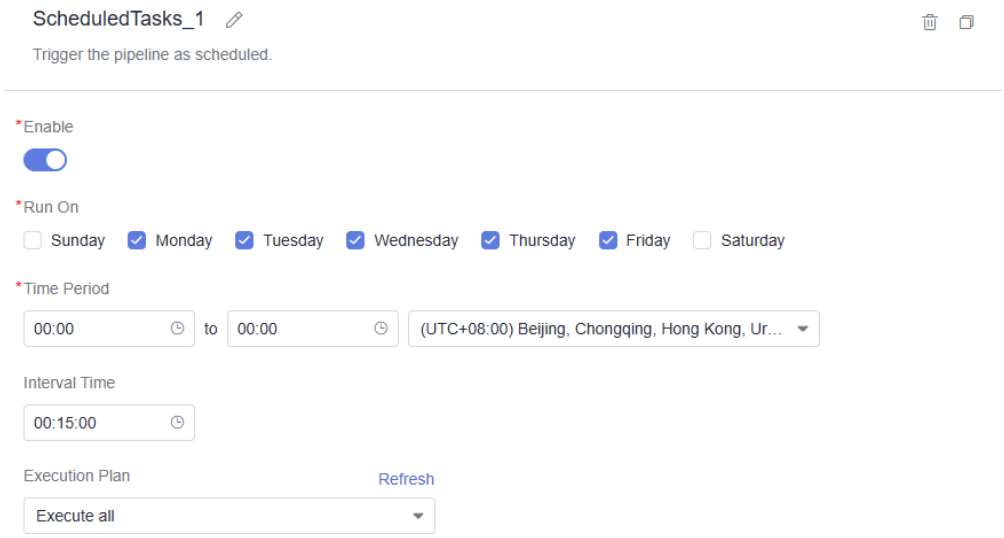


Table 5-11 Scheduled task

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

NOTE

- 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 Webhooks

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

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

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

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

Step 4 Enable **Webhook**, set parameters as shown in **Table 5-12**, and save the pipeline for the setting to take effect.

Figure 5-13 Configuring a webhook trigger

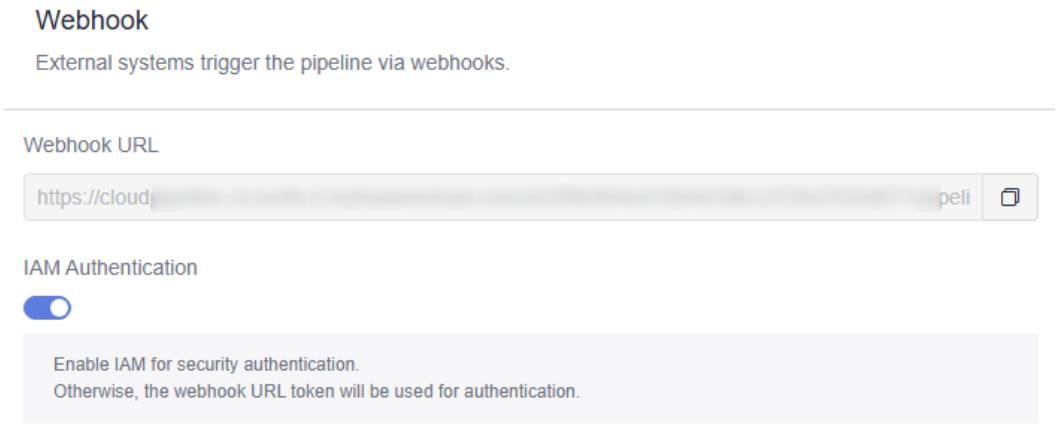


Table 5-12 Webhook parameters

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

----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

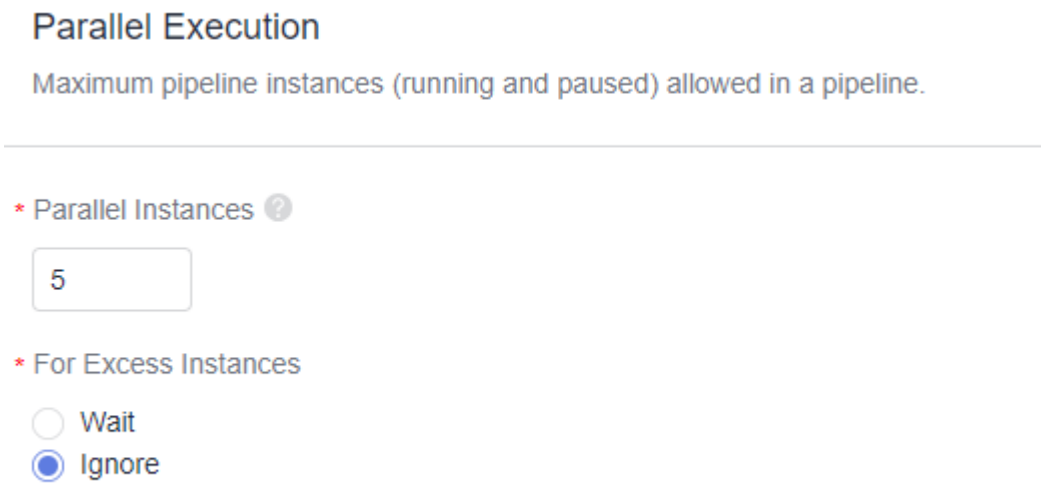


Table 5-13 Parallel execution parameters

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

Configuring Preemption

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

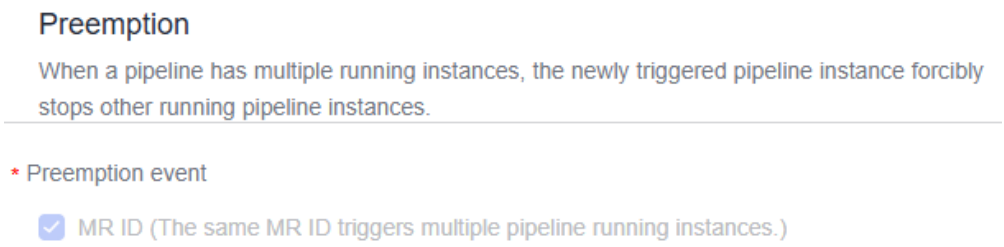
Step 1 **Access the CodeArts Pipeline homepage**.

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

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

Step 4 Enable **Preemption**, select a preemption event, and save the pipeline for the setting to take effect.

Figure 5-15 Configuring a preemption policy



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

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

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 **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 users who favorite pipelines of pipeline activities (deleted, failed, succeeded, updated).

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

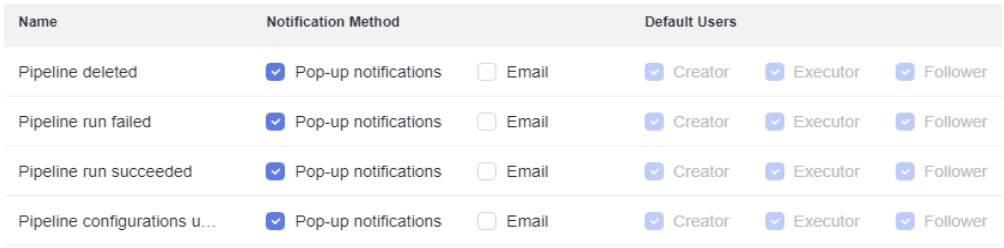
Step 2 On the pipeline list page, search for the target pipeline, click ··· in the **Operation** column, and click **Edit**.

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 5-16 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).

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.

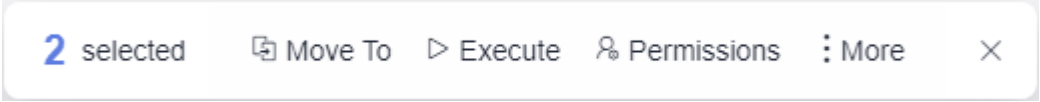
NOTE

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.

  NOTE

  On the **Permissions** tab page, only the project admin and project manager can modify pipeline permissions in batches.
- Choose **More** > **Set Tag**. In the displayed dialog box, set tags for selected pipelines.
- Choose **More** > **Delete**. In the displayed dialog box, enter the prompt information and click **OK**. A maximum of 20 pipelines can be deleted at a time.

----End

7 Executing a Pipeline

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

Executing a Pipeline

Step 1 Access the CodeArts Pipeline homepage.

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

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

- **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 executed.

  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.

- Click **Stop** in the upper right corner to stop the execution.
- Click **Edit** to change the pipeline configurations.
- A pipeline can have multiple concurrent executions. You can click **Execute** to start a new execution of the pipeline.

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.

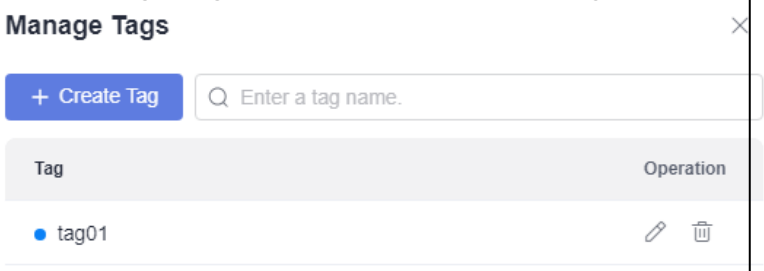
Checking a Pipeline

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.

Table 8-1 Pipeline information

Parameter	Description
Name	Pipeline name and the project to which the pipeline belongs. **NOTE** If you access CodeArts Pipeline through a project, the project name will not be displayed here.
Last Executed	Information about the most recently executed pipeline, including the execution mode, branch, latest code commit ID, and executor.
Tag	Pipeline tag.
Workflow	Scheduling process and execution status (completed, failed, running, or stopped) of the pipeline.
Started & Lasted	Start time and duration of the last execution.
Execution Time	Start time of the last execution.
Execution Duration	Duration of the pipeline execution.

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 ••• and select **Edit** to edit a pipeline. • Click ••• and select **Clone** to quickly create a pipeline based on the current pipeline. • Click ••• and select **Preview** to preview a pipeline. • Click ••• and select **Operation History** to view historical operation records (creation, editing, and failure). • Click ••• and select **Set Tag**. In the displayed dialog box, perform the following operations. – Click + to add tags for the pipeline. Max. five tags for each pipeline. – Click **Manage Tags** to create, edit, or delete tags.  **NOTE** A tenant can create a maximum of 100 tags. • Click ••• and select **Disable** to disable a pipeline. • Click ••• and select **Delete** to delete a pipeline. **NOTE** By default, only project managers, project creators, and system engineers can delete pipelines. You can configure permissions for different roles. • Click ••• and select **Execution Plan** to **configure a plan**.

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

- Click **Set** in the upper right corner to customize the pipeline display information.

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

NOTE

Execution records are generated only after the first execution.

Table 8-2 Execution history

Parameter	Description
Execution Message	Displays the execution sequence number, execution branch, latest commit information, and latest commit ID of the branch.
Status	Pipeline execution status, including completed, running, failed, stopped, paused, suspended, and ignored.
Execution Type	Pipeline triggers, including manually, scheduled task, MR, push, webhook, and sub-pipeline.
Workflow	Scheduling process and execution status (completed, failed, running, or stopped) of the pipeline.
Execution Description	Description of the pipeline execution.
Job Details	Pipeline execution details.
Execution Time	Time when the pipeline starts to be executed.
Execution Duration	Duration of the pipeline execution.

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

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

Table 8-3 Operations on the pipeline details page

Operation	Description
Retry	If the execution fails, you can click **Retry** in the upper right corner to continue the execution.
Edit	You can click **Edit** to orchestrate the pipeline.

Operation	Description
Execute	You can click **Execute** to execute the pipeline with the latest configurations. An execution record will be generated.
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**. • You can click the failed job card to check the failing reason.
More operations	Click ... in the upper right corner of the page to clone, preview, disable, and delete the pipeline, and check the operation history.

Step 4 Click the **Queued** tab.

This page displays the instances to be executed.

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

----End

9 Checking the Dashboard

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

Checking the Dashboard

Step 1 Access the CodeArts Pipeline homepage.

Step 2 The dashboard displays pipeline statistics.

Table 9-1 Pipeline statistics

Item	Description
Pipelines	Total number of pipelines.
Executions	Execution success rate and execution times. You can check executions during a specified period.
Parallel Executions	Number of queuing and running parallel instances. Click on the right of the queuing instances to check the queuing pipelines and triggers. Click on the right of the running parallel instances to check the names, execution information, triggers, and execution time of running pipelines.
Pipelines per Project	Number of pipelines in each project. You can click a project name to access pipelines in this project.
Executions per Project	Execution success rate and execution times in each project. You can check executions during a specified period. You can also click a project name to access pipelines in this project.
Parallel Executions per Project	Number of queuing and running parallel instances in each project.

----End

10 Configuring a Change-triggered Pipeline

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 10-1 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 (_).
Pipeline Source	Code repository source. 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 code repository 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 10-2 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. 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 10-3 Pipeline parameters

Parameter	Description
Name	Pipeline name. It is generated based on the creation time by default. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 128 characters.
Project	Project to which the microservice belongs.
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.
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.

Parameter	Description
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 Based on Changes. **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 10-4 Change parameters

Parameter	Description
Change Subject	Name of the change. Enter a maximum of 256 characters.

Parameter	Description
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.

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 the execution configuration.

- **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.

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 code 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

11 Managing Pipeline Extensions

11.1 Extensions Overview

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

Accessing the Extension Platform

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

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

Scenarios

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

11.2 Pipeline Official Extensions

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

Table 11-1 Pipeline Official Extensions

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

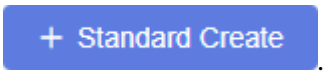
Type	Name	Description
Check	Check	Calls CodeArts Check capabilities. CodeArts Check is a cloud-based management service that checks code quality. Developers can easily perform static code and security checks in multiple languages and obtain comprehensive quality reports. CodeArts Check also provides bug fixing suggestions and trend analysis to control code quality and reduce costs. **Learn more.**
	Check-Template	This extension can be configured only in a pipeline template. When a pipeline is generated based on the template, the extension automatically creates a Check job and configures the job in the generated pipeline.
	Pass-Conditions-of-Standard-Policies	A standard extension policy for access control.
Normal	CreateTag	Creates and pushes tags for code repositories.
	Subpipeline	Configures and calls other pipelines in a project.
	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.
	CreateReleaseBranch	Creates a release branch based on the default branch of a microservice. This extension is automatically configured by a change-triggered pipeline.
	MergeReleaseBranch	Merges a feature branch into a release branch. This extension is automatically configured by a change-triggered pipeline.
	MergeDefaultBranch	Merges a release branch into the default branch of a microservice. This extension is automatically configured by a change-triggered pipeline.

11.3 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 

.

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

Table 11-2 Extension information

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

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

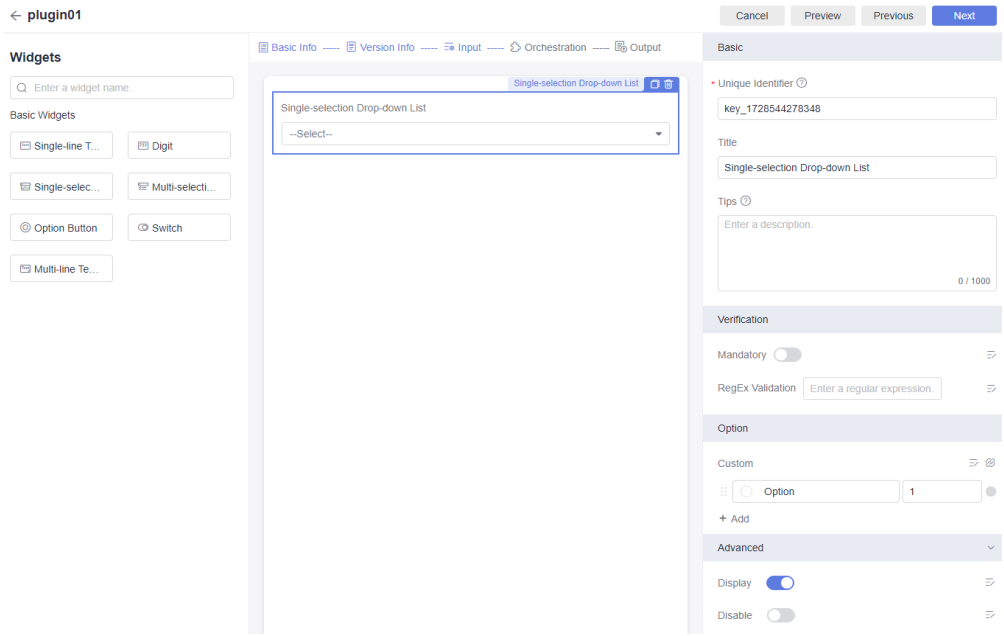
NOTE

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

Figure 11-1 Orchestrating widgets



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

Table 11-3 Widget parameters

Category	Parameter	Description	Widget
Basic	Unique Identifier	Unique ID of the widget. The ID is used to obtain widget input. Enter only letters, digits, underscores (_), and hyphens (-) with a maximum of 200 characters.	All
	Title	Name of the widget. The name will be displayed on the pipeline job orchestration page. Enter no more than 140 characters.	All
	Tips	Tooltip of the widget. Enter no more than 1,000 characters.	All
	Placeholder	Informative message displayed in the text box. For example, what value should be input.	Single-line Text Box
	Accuracy	Number of decimal places allowed in a widget value: **0** to **4**.	Digits

Category	Parameter	Description	Widget
	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
Option	Custom	Options available for the widget. Click + Add to add an option. Click ⊖ to delete an option. • Option name: option displayed on the extension configuration page. • Value: value delivered when the extension is running. In addition to manual configuration, set options by: • APIs: Set options by configuring web APIs. Click ⚙ on the right. On the displayed dialog box, you can configure parameters after enabling the function. For details, see **Table 11-4**. • Context: Configure data source to obtain the URL of the pipeline source or IDs of build jobs. Click 🔗 next to **Custom**. The context dialog box is displayed. You can configure parameters after enabling the function.	Single-selection Drop-down List, Multi-selection Drop-down List, and Option Button
Advanced	Display	Whether the widget is visible. You can click ⚙ on the right to set display conditions.	All

Category	Parameter	Description	Widget
	Disable	Whether the widget is disabled (not disabled by default). You can click on the right to set conditions.	All

Table 11-4 API parameters

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

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.

Parameter	Description
Remote Search	Enable this function to add a remote search field. For extension search, the entered value will be used as the value of the remote search field to call the API again. • Params parameter: The parameter type of the search field is the Params parameters of the API request body. • Body parameter: The parameter type of the search field is the Body parameters of the API request body.
Search Tip	Describes the search function and is displayed below the search bar. Max. 100 characters.

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

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

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

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

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

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

- Draft release

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

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

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

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

- Official release

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

----End

11.4 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
  |    | -- 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
```

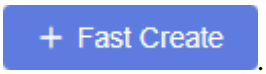
Notes:

- 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 11-5**.

Table 11-5 Extension information

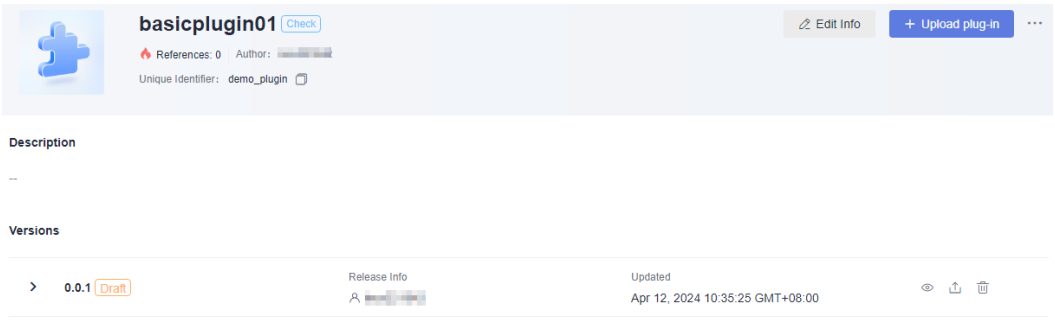
Parameter	Description
Icon	Icon of the extension. Upload an image in PNG, JPEG, or JPG format, with a file size no more than 512 KB (recommended: 128 x 128 pixels). If no image is uploaded, the system generates an icon.
Name	The extension name displayed in the extension platform. Enter only spaces, letters,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

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

Step 5 Click **OK**.

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

Figure 11-2 Uploading an extension



Step 7 Using an extension in a pipeline

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

Step 8 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": [{
         "name": "okey",                      # Output name.
         "type": "output",                    # Output type: output or metrics.
         "description": "Description",
         "prop": {
            "defaultValue": "123"           # Default value
         }
      },
      {
         "name": "mkey",
         "type": "metrics",
         "description": "Description",
         "prop": {
            "defaultValue": "213",
            "group": "213"                # Corresponding to the group name in pass conditions
         }
      }
   ]
}
```

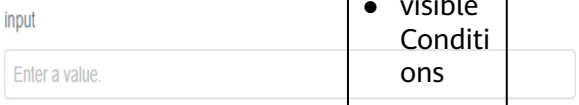
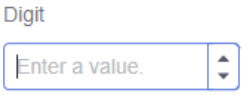
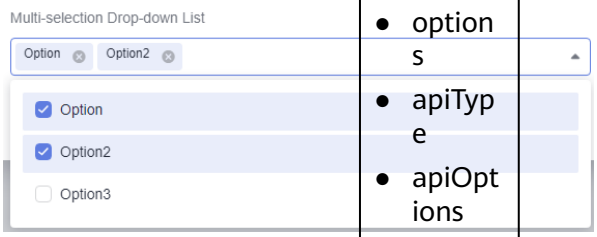
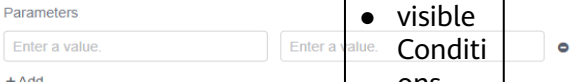
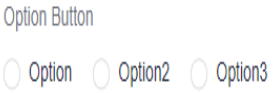
The parameters of **codearts-extension.json** are described in the following table.

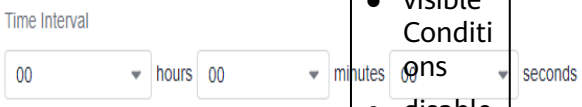
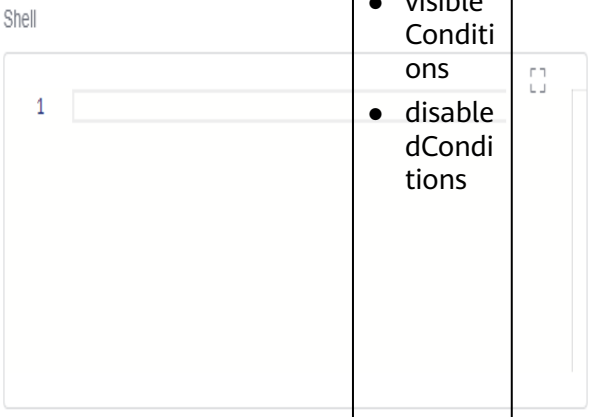
Table 11-6 Parameters

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 11-7 inputs

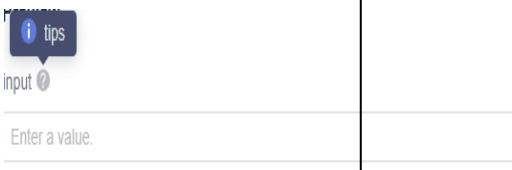
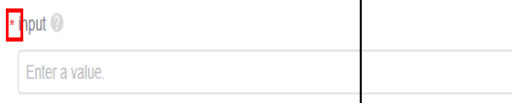
Type	Widget	Example	extendProp
input	Single-line Text Box		• visibleConditions • disabledConditions
inputNumber	Digit		• visibleConditions • disabledConditions
switch	Switch		• visibleConditions • disabledConditions
singleSelect	Single-selection Drop-down List		• options • apiType • apiOptions
multipleSelect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

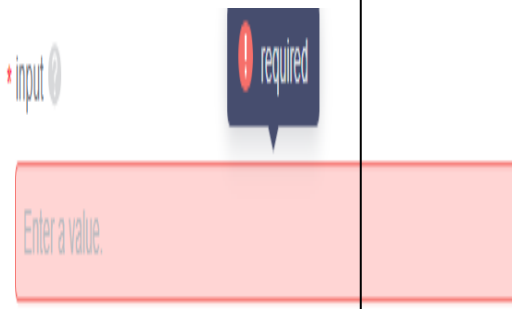
Type	Widget	Example	extendProp
timeInterval	Time Interval		• visibleConditions • disabledConditions
shell	Shell		• visibleConditions • disabledConditions
endpoint:*${module_id}*	Endpoint		• visibleConditions • disabledConditions

inputs fields are listed in the following table.

Table 11-8 inputs fields

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

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

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

extendProp	Extension field
	{
	visibleConditions: [],
	disabledConditions: []
	...
	}
		No	For details about extendProp, see **Table 11-9**.

extendProp functions are listed in the following table.

Table 11-9 extendProp functions

Field	Description	Mandatory	Remarks

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

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

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

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

Field	Description	Mandatory	Remarks

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

11.5 Executing Images

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

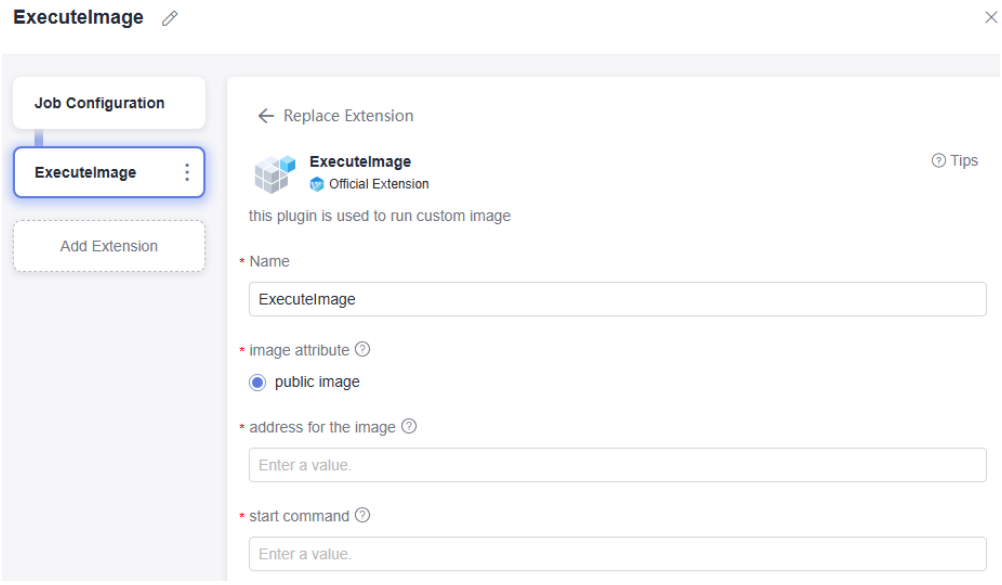
Constraints

This extension is only available for custom executors.

Configuration Method

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

Figure 11-3 Extension for executing images



Step 2 Set parameters as shown in the following table.

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

----End

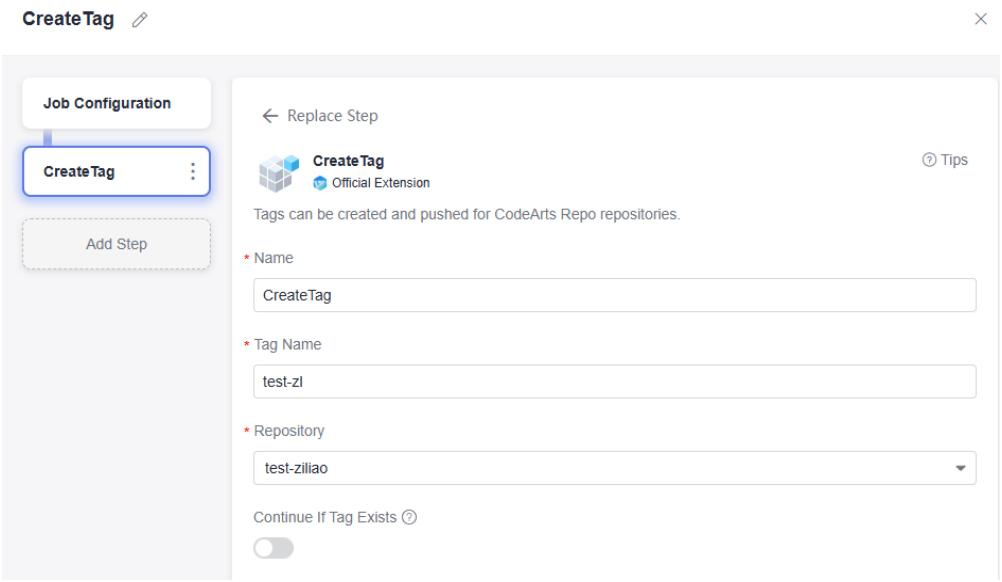
11.6 Pushing Tags for Code Repositories

You can create and push tags for code repositories.

Configuration Method

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

Figure 11-4 Adding the CreateTag extension



Step 2 Set parameters as shown in the following table.

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

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

----End

12 Creating Service Endpoints

Scenario

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

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

The following table lists the endpoints supported by CodeArts.

Table 12-1 Service endpoints

Type	Scenario
Docker repository	Connect to a third-party Docker image repository. After the connection is successful, CodeArts Deploy can obtain Docker images from the repository.
Kubernetes	Connect to a Kubernetes cluster. After the connection is successful, you can deploy applications to the relevant Kubernetes cluster.
IAM user	Delegate your AK/SK to an IAM user so that the user can obtain a token to perform tasks that require higher permissions.

Prerequisites

- You must have the **DevMarket** > **Endpoint** > **create** permission to create service endpoints. For details, see
- The third-party service to connect can be accessed from the public network without restrictions.

Creating a Service Endpoint

Step 1 Go to the CodeArts homepage.

Log in to the **CodeArts console**, and click **Access Service**.

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

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

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

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

The new endpoint is displayed.

----End

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

Docker Repository

Table 12-2 Creating a Docker repository service endpoint

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

Kubernetes

Table 12-3 Creating a Kubernetes service endpoint

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

IAM User

Table 12-4 Creating an IAM user service endpoint

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

Managing a Service Endpoint

Step 1 Go to the CodeArts homepage.

Log in to the **CodeArts console**, and click **Access Service**.

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

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

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

Table 12-5 Managing a service endpoint

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

----End

13 Reference

13.1 Pipeline Contexts

13.1.1 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 13-1 Pipeline contexts

Context	Type	Description
pipeline	object	Information about the pipeline run.
sources	object	Information about the pipeline sources in each pipeline run.
env	object	Information about the custom parameters in each pipeline run.
jobs	object	Information about jobs that have reached the final states in each pipeline run.

Context Reference Format

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

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

Contexts Attributes

Table 13-2 Context attributes

Context	Attribute	Type	Description	Example

pipeline context	pipeline	object	Information about the pipeline run. This object contains the following attributes: **project_id**, **pipeline_id**, **run_number**, **timestamp**, **trigger_type**, and **run_id**.	• Content example
				The following example shows the pipeline context information contained in a manually executed pipeline.
				`{ "project_id": "6428c2e2b4b64affa14ec80896695c49", "pipeline_id": "f9981060660249a3856f46c2c402f244", "run_number": "168", "timestamp": "20231016000004", "trigger_type": "Manual", "run_id": "c2f507f93510459190b543e47f6c9bec" }`
				• Usage example
				To obtain the 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).	
	sources.<alias>.tag	string	Tag name when the tag is triggered.	

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

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

	value	string	Value of a custom parameter.	

Context	Attribute	Type	Description	Example

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

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

Related Information

The following are context scenarios:

- **Configuring Expressions**.
- **Obtaining Artifact Information Using the Pipeline Context**.

13.1.2 Configuring Expressions

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

Step 1 **Create a pipeline**.

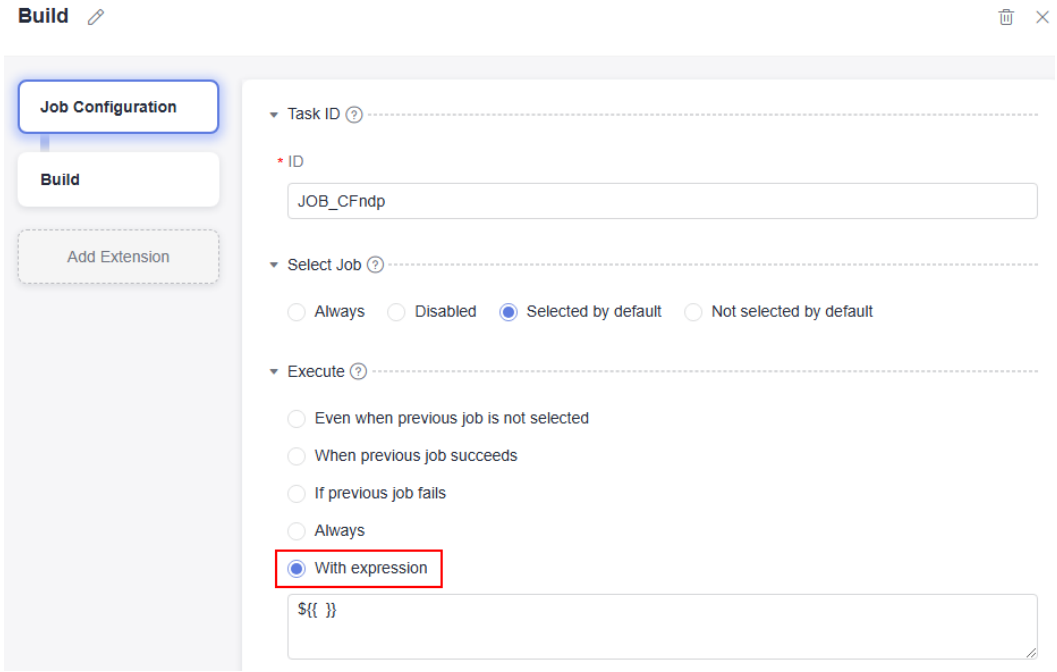
Step 2 Add stage jobs or edit existing jobs.

Step 3 Click **Job Configuration** and select **With expression** to configure the expression for job execution.

NOTE

For a new stage and job, add an extension first and then click **Job Configuration**.

Figure 13-1 Expressions



Example:

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

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

----End

References

- **Operator**

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

  Table 13-3 Operators

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

- **Function**

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

Table 13-4 Built-in functions

Function	Description
contains	• Format contains(search, item) • Description If *search* contains *item*, this function returns **true**. – If *search* is an array and *item* is an element in the *array*, this function returns **true**. – If *search* is a string and *item* is a substring of *search*, the function returns **true**. • Example **contains('abc', 'bc')** returns **true**.
startsWith	• Format startsWith(searchString, searchValue) • Description If *searchString* starts with *searchValue*, this function returns **true**. • Example **startsWith('abc', 'ab')** returns **true**.
endsWith	• Format endsWith(searchString, searchValue) • Description If *searchString* ends with *searchValue*, this function returns **true**. • Example **endsWith('abc', 'bc')** returns **true**.
format	• Format format(string, replaceValue0, replaceValue1, ..., replaceValueN) • Description Replaces the value in *string* with the value of *replaceValueN*. • Example **format('Hello {0} {1}', 'a', 'b')** returns **Hello a b**.
substring	• Format substring(string, beginIndex, endIndex) • Description Returns the characters in *string* from beginIndex to endIndex-1. If endIndex is not configured, the characters in *string* from beginIndex to the end are returned. • Example **substring('202412091101', 0, 8)** returns **20241209**. **substring ('202412091101', 8)** returns **1101**.

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

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

13.1.3 Obtaining Artifact Information Using the Pipeline Context

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

Step 1 Create a pipeline.

Step 2 Add the **Build** extension to **Stage_1**, obtain the task ID as shown in Figure 13-2, and set the artifact identifier to **demo** as shown in Figure 13-3.

Figure 13-2 Obtaining the task ID

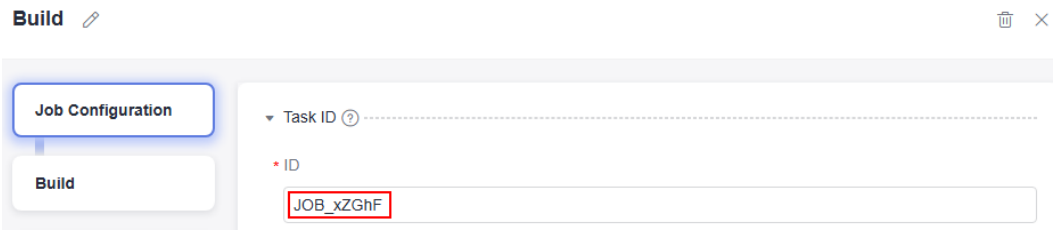
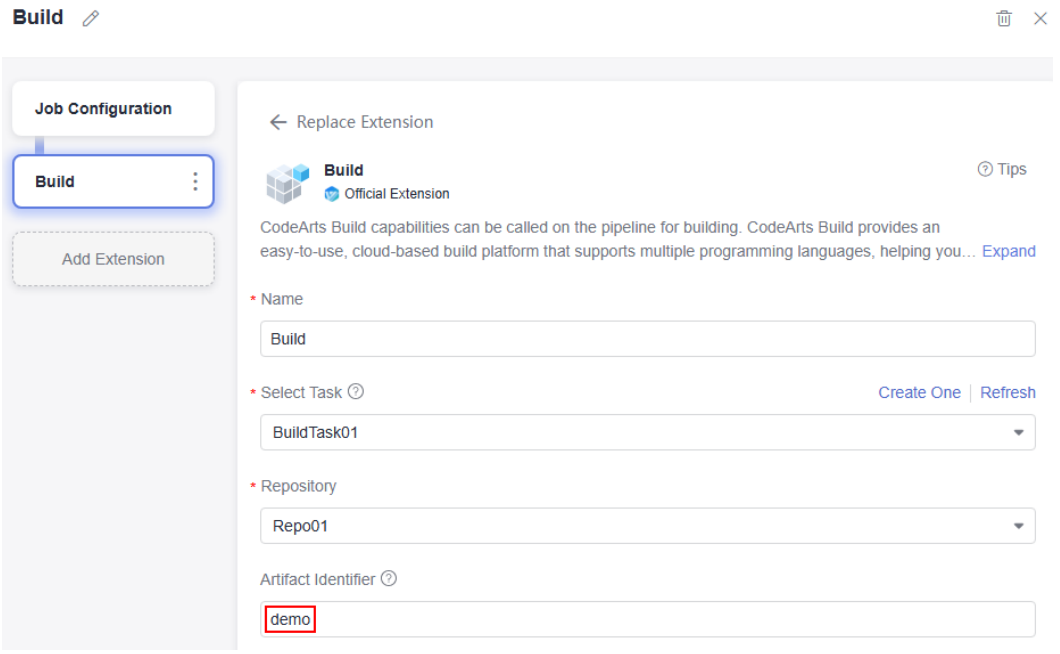


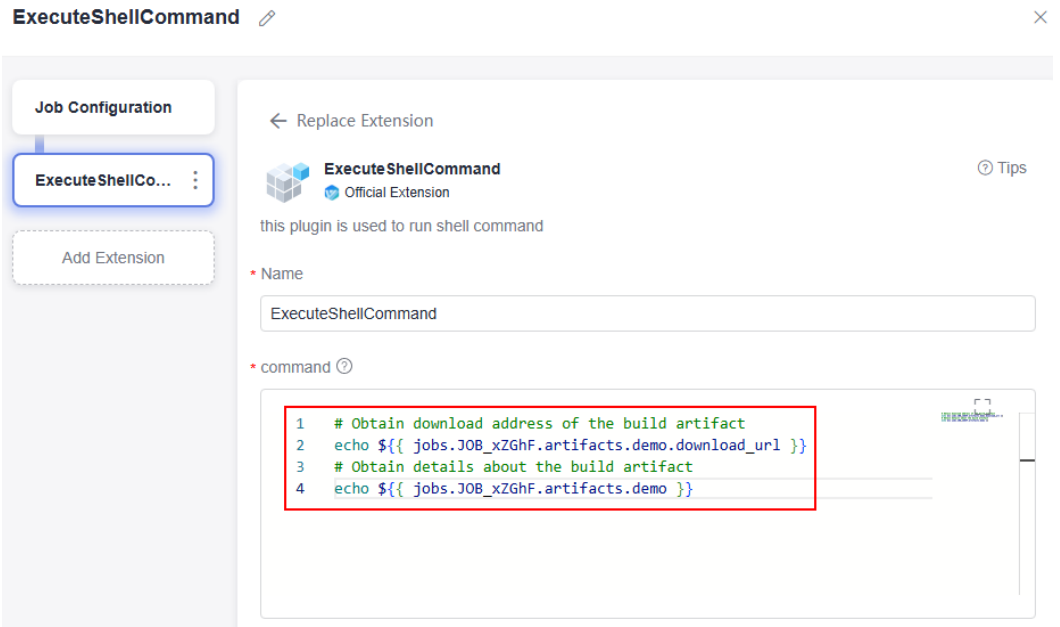
Figure 13-3 Adding the Build extension



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

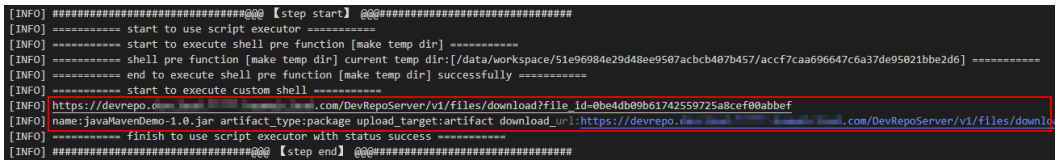
```
# Obtain the download address of the artifact.
echo ${{ jobs.JOB_xZGhF.artifacts.demo.download_url }}
# Obtain all information about the artifact.
echo ${{ jobs.JOB_xZGhF.artifacts.demo }}
```

Figure 13-4 Adding the ExecuteShellCommand extension



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

Figure 13-5 Artifact information



----End

13.2 YAML Syntax

13.2.1 on

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

on.<event_name>.types

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

Table 13-5 Merge request types

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

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

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

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

Use the **pull_request** event to define a pipeline to run for MRs that target specific branches.

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

NOTE

Do not use **branches** and **branches-ignore** for a **pull_request** event at the same time.

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

Table 13-6 Matching characters

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

Example

- **Including branches**

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

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

  - a branch named **master**, or;
  - a branch whose name starts with **release**, for example, **release**, **release-1.0.0**, and **release/1.0.0**

- **Excluding branches**

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

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

  - a branch named **test**, or;
  - a branch whose name starts with **dev**, for example, **dev**, **develop-1.0.0**, and **develop/1.0.0**

- **Both including and excluding branches**

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

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

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

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

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

Use the **push** event to define a pipeline to run on specific branches or tags.

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

NOTE

- Do not use **branches** and **branches-ignore** for a **push** event at the same time.
- Do not use **tags** and **tags-ignore** for a **push** event at the same time.

Example

- **Including branches/tags**

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

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

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

- **Excluding branches/tags**

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

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

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

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

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

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

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

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

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

Use the **push** and **pull_request** events to define a pipeline to run when specified files change.

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

NOTE

Do not use **paths** and **paths-ignore** for **push** and **pull_request** events at the same time.

Example

- **Including files**

```
on:
  push:
```

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

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

- **Excluding files**

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

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

- **Both including and excluding files**

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

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

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

on.schedule

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

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

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

13.2.2 env

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

Example

```
env:
  version: 1.0.0
```

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

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

13.2.3 jobs

A pipeline can consist of multiple jobs.

jobs.<job_id>

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

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

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

jobs.<job_id>.name

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

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

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

jobs.<job_id>.needs

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

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

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

jobs.<job_id>.if

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

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

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

jobs.<job_id>.steps

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

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

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

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

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

```
jobs:
  demo_job:
    name: simple demo job
    steps:
      - name: simple custom step
        uses: custom_plugin@1.0.0
```

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

YAML Syntax for the Pipeline Official Extensions

- **Build**

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

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

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

- **TestPlan**

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

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

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

- **Deploy**

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

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


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

- **Check**

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

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

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

- **CreateTag**

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

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

  tagName: Tag name.

- **Subpipeline**

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

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

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

- **JenkinsTask**

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

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


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

- **PipelineSuspension**

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

```
uses: SuspendPipeline
```

- **DelayedExecution**

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

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

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

Table 13-7 Time zone

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

Available Value	Time Zone
Saudi Arabia Standard Time	GMT+03:00

- **ManualReview**

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

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

  - **mode**: Review mode. **members** indicates that the review is performed by member, and **roles** indicates that the review is performed by role.
  - **approvers**: User IDs of the approvers when **mode** is set to **members**. Use commas (,) to separate multiple user IDs.
  - **role**: Roles when the **mode** is set to **roles**. For details about the options, see **Table 13-8**. Use commas (,) to separate multiple roles.
  - **checkStrategy**: Strategy used when the **mode** is set to **members**. **all** indicates that the application can be approved only by all users. **any** indicates that the application can be approved by any user.
  - **timeout**: Review timeout, in seconds.
  - **timeoutStrategy**: Strategy used when the review times out. **reject** indicates that the pipeline is stopped to run. **pass** indicates that the pipeline can continue to run.
  - **comment**: Review description.

Table 13-8 Review roles

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

Role	YAML Identifier
Committer	'12'

