CodeArts Pipeline

Best Practices

Issue	01
Date	2025-08-27

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2025. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI 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 Best Practices

This document includes best practices for using CodeArts Pipeline in common scenarios. Each practice is given a description and procedure.

Table 1-1 CodeArts Pipeline best practices

Practice	Description
Fixing a Bug for Quick Release Through a Change-triggered Pipeline	Use a change-triggered pipeline to fix a bug and quickly release the fix.
Configuring Pass Conditions for Automated Code Checks	Configure code check thresholds and apply pass conditions to a stage for automated checks.
Transferring CodeArts Pipeline Parameters to CodeArts Build and CodeArts Deploy	Transfer a pipeline version number parameter to a CodeArts Build task and a CodeArts Deploy task.
Creating Tags for Code Repositories Through Pipelines	Create a repository tag through the pipeline contexts.
Configuring a Pipeline Gate for a Code Repository Merge Request	Configure an automated pipeline gate for merge requests to improve code quality and team collaboration efficiency.
Managing Pipeline Permissions	You can manage CodeArts Pipeline's project-level and resource-level permissions. This practice describes how to configure resource-level permissions.
HE2E DevOps Practice: Configuring a Pipeline	This practice describes how to connect code check, build, and deployment tasks in a **DevOps Full-Process Sample Project** for continuous delivery.

2 Fixing a Bug for Quick Release Through a Change-triggered Pipeline

Overview

CodeArts Pipeline provides the microservice model for enterprises. Each microservice is independently developed, verified, deployed, and rolled out, accelerating requirement release. This model also lets enterprises organize teams by function, optimize management models, and improve operation efficiency.

Using this model, you can create change-triggered pipelines to associate them with change resources and release changes for quick project delivery.

Constraints

Change-triggered pipelines can only be triggered by changes. A microservice can only have one change-triggered pipeline.

Procedure

The following describes how to use a change-triggered pipeline to fix a bug for quick release.

Table 2-1 Procedure

Step	Description
Create a microservice	Manage a specific service function.
Create a change	Associate a change with bug fixing work items.
Create a change-triggered pipeline	Release a change in a microservice.

Step	Description
Execute a change-triggered pipeline	Release the updated code.

Preparations

- **You have enabled and authorized CodeArts Pipeline**.
- **You have created a project**. The following uses a Scrum project name **Project_Test** as an example. **You have created a work item** in the project. The following uses a bug work item named **BUGFIX** as an example.
- **You have created a code repository**. The following uses a repository named **Repo_Test** (created using the **Java Maven Demo** template) as an example.
- **You have created a CodeArts Repo HTTPS service endpoint**. The following uses an endpoint named **HttpsEndpoint01** as an example.

Step 1: Create a Microservice

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**.

Step 4 Click **Homepage** from the top navigation pane. Search for the project created in **Preparations** and access the project.

Step 5 In the navigation pane on the left, choose **CICD** > **Pipeline**.

Step 6 Click the **Microservices** tab.

Step 7 Click **Create Microservice**. On the displayed page, configure parameters.

Table 2-2 Microservice parameters

Parameter	Description
Project	Keep the default value, which is the project of the microservice.
Microservice Name	Enter **Microservice01**.
Code Source	Code source associated with the microservice. Select **Repo**.
Repository	Select the repository **Repo_Test** created in **Preparations**.
Default Branch	Select **master**.
Language	Development language for the microservice. Select **Java**.

Parameter	Description
Description	Optional. Describe the microservice with a maximum of 1,024 characters.

Step 8 Click **OK**.

----End

Step 2: Create a Change

Step 1 Click the created microservice name. The **Overview** page is displayed.

Step 2 Click the **Changes** tab.

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

Table 2-3 Change parameters

Parameter	Description
Change Subject	Enter **fix-a-bug**.
Repository	Keep the default value, which is the same as that of the microservice.
Branch	The development branch for the change. After the change is successfully released through the pipeline, the branch will be automatically merged to the default branch of the microservice. Select **Pull new from default** and enter the branch name **bugfix**.
Associated Work Item	Select the work item **BUGFIX** created in **Preparations**.

Step 4 Click **OK**.

After the change is created, the system creates a feature branch based on the microservice default branch. You can commit code to this feature branch. For details about how to commit code to a branch, see **Editing and Creating a Merge Request**.

----End

Step 3: Create a Change-triggered Pipeline

Step 1 In the microservice list, click a microservice name. The **Overview** page is displayed.

Step 2 Switch to the **Pipelines** tab.

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

Table 2-4 Pipeline parameters

Parameter	Description
Name	Enter **Pipeline-Test**.
Project	Keep the default value, which is the project of the pipeline.
Pipeline Source	Keep the default value, which is the same as that of the microservice.
Repository	Keep the default value, which is the same as that of the microservice.
Default Branch	Keep the default value, which is the same as that of the microservice.
Repo Endpoint	This is mandatory if you enabled **Change-based Trigger**. Select the authorization endpoint **HttpsEndpoint01** created in **Preparations**.
Alias	Repository alias. If you set an alias, system parameters will be generated. It is not set in this example.
Change-based Trigger	Enable it to set current pipeline to a change-triggered one. It is enabled in this example.
Description	(Optional.) Describe the pipeline with a maximum of 1,024 characters.

Step 4 Click **Next** and select the **Maven-Build** template. Stages and jobs will be generated. You can retain the default settings.

Step 5 Click **Save**.

----End

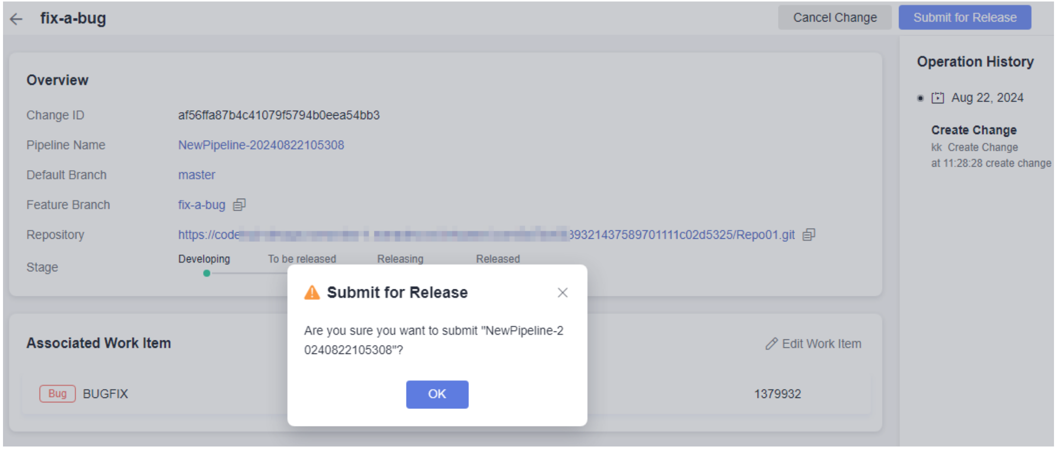
Step 4: Execute a Change-triggered Pipeline

After the code is updated, you can execute the change-triggered pipeline.

Step 1 On the change list page, click a change name.

Step 2 Click **Submit for Release** in the upper right corner. In the displayed dialog box, confirm the release.

Figure 2-1 Submitting for release



Overview

Change ID af56ffa87b4c41079f5794b0eea54bb3
Pipeline Name NewPipeline-20240822105308
Default Branch master
Feature Branch fix-a-bug
Repository https://code[redacted]39321437589701111c02d5325/Repo01.git
Stage Developing To be released Releasing Released

Associated Work Item
Bug BUGFIX 1379932

Operation History
Aug 22, 2024
Create Change
kk Create Change
at 11:28:28 create change

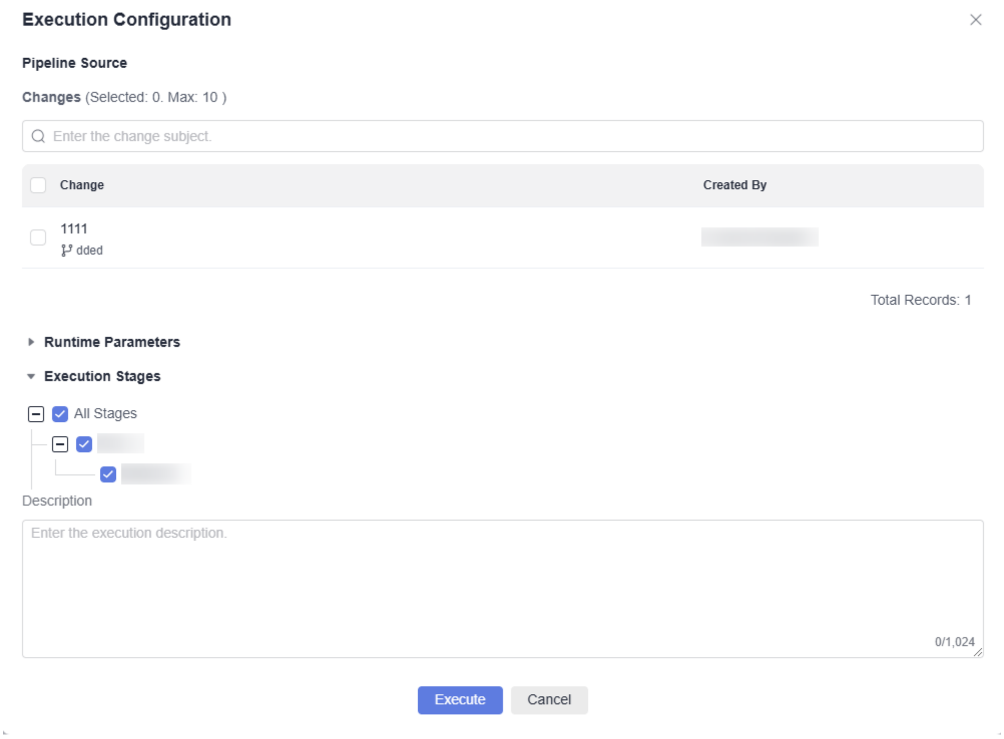
Submit for Release

Are you sure you want to submit "NewPipeline-20240822105308"?

Step 3 Click **OK**. The release list page is displayed.

Step 4 Click **Execute** in the upper right corner. In the displayed window, select the submitted change, and retain the default settings.

Figure 2-2 Execution configuration



Execution Configuration

Pipeline Source

Changes (Selected: 0. Max: 10)

Enter the change subject.

□ Change	Created By
□ 1111 dded	[redacted]

Total Records: 1

▸ Runtime Parameters
▾ Execution Stages
☑ All Stages
☑ [redacted]
☑ [redacted]

Description

Enter the execution description.

0/1,024

Step 5 Click **Execute**.

During pipeline running, the **MergeReleaseBranch** and **MergeDefaultBranch** stages are automatically generated. The newly pulled feature branch is merged to the integration branch.

After the code check and build jobs are successfully executed, the pipeline proceeds to the **MergeDefaultBranch** stage with a confirmation dialog box displayed.

Step 6 Click **Continue**. After the **MergeDefaultBranch** stage is executed, the system:

- Updates the change status to **Released**.
- Updates the status of the **BUGFIX** work item to **Closed**.
- Merges the code on the release branch to the default branch.

A change release has been completed.

If the pipeline fails to be executed, rectify the fault based on the error message. For details, see **FAQ**.

----End

3 Configuring Pass Conditions for Automated Code Checks

Overview

Traditional software quality relies mainly on manual tests, leading to low efficiency.

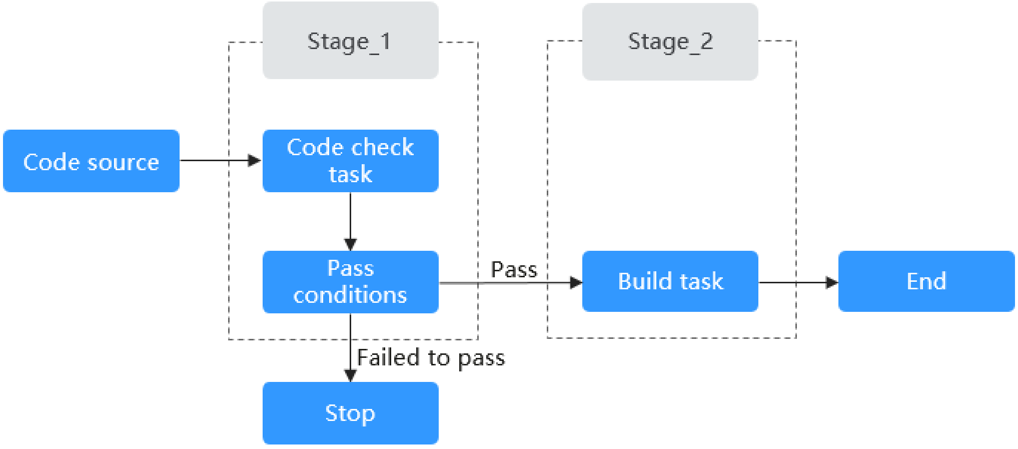
CodeArts Pipeline uses pass conditions to control whether a pipeline can proceed to the next stage. You can apply policies to pipelines as pass conditions for efficient project management and high-quality delivery.

With CodeArts Pipeline, more than 70% issues can be intercepted through automated code checks. This improves test efficiency and software quality.

Procedure

The following describes how to configure code check thresholds and apply pass conditions to a stage for automated check.

Figure 3-1 Pipeline workflow



Perform the following procedure.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Table 3-1 Procedure

Step	Description	Billing
Create a rule and configure thresholds	Create a rule of the code check type and configure thresholds for the rule.	Upgrade the CodeArts package to the basic edition. For details, see **CodeArts Billing Modes**.
Create a policy and add the rule to the policy	Add the preceding code check rule to the created policy.	Upgrade the CodeArts package to the basic edition. For details, see **CodeArts Billing Modes**.
Configure a pipeline	Add the preceding policy to the pass conditions.	No billing involved.
Execute the pipeline	Execute the pipeline: ● If the code check job meets the pass conditions, the pipeline will continue to run. ● If the code check job does not meet the pass conditions, the pipeline will stop running.	No billing involved.

Preparations

- **You have enabled and authorized CodeArts Pipeline**.
- **You have created a project**. The following uses a Scrum project named **Project_Test** as an example.
- **You have created a code repository**. The following uses a repository named **Repo_Test** (created using the **Java Maven Demo** template) as an example.

  A code check task with the same name as the code repository is automatically created. Change the task name to **CheckTask01** by referring to **Creating a Task**.
- **You have created a build task** with the **Repo_Test** repository. The following uses a build task (created using the **Maven** template) named **BuildTask01** as an example.
- **You have created a pipeline** with the **Repo01** repository. The following uses a pipeline named **Pipeline-Test** (created using the blank template) as an example.

Step 1: Create a Rule and Configure Thresholds

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.

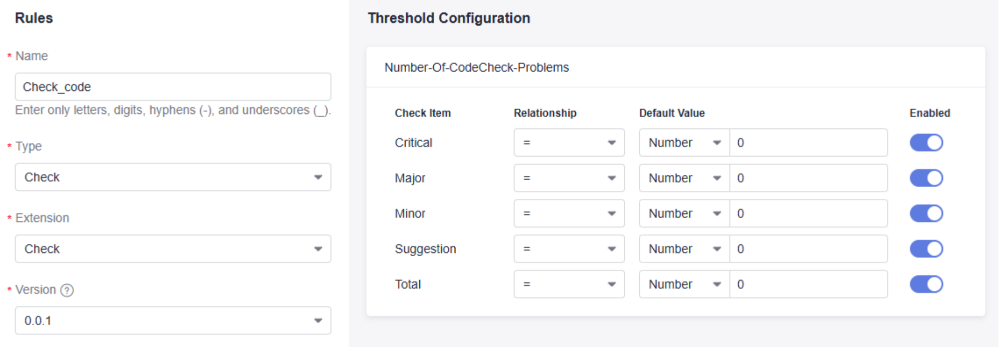
Step 3 Click **Access Service**.

Step 4 Click the avatar icon in the upper right corner and choose **All Account Settings** from the drop-down list.

Step 5 In the navigation pane on the left, choose **Policy Management** > **Rules**.

Step 6 Click **Create Rule**. On the displayed page, configure parameters.

Figure 3-2 Creating a rule



Rules

* Name

Check_code

Enter only letters, digits, hyphens (-), and underscores (_).

* Type

Check

* Extension

Check

* Version

0.0.1

Threshold Configuration

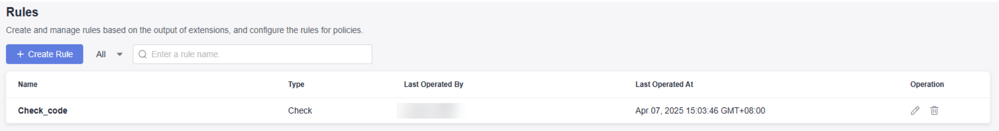
Number-Of-CodeCheck-Problems

Check Item	Relationship	Default Value		Enabled
Critical	=	Number	0	●
Major	=	Number	0	●
Minor	=	Number	0	●
Suggestion	=	Number	0	●
Total	=	Number	0	●

Table 3-2 Rule parameters

Parameter	Description
Name	Enter a rule name, such as **Check_code**.
Type	Select the rule type **Check**.
Extension	Select the extension **Check**.
Version	Select the version **0.0.1**.
Threshold Configuration	The extension thresholds are automatically filled based on the selected extension version. You can use the default values.

Step 7 Click **OK**.



Rules

Create and manage rules based on the output of extensions, and configure the rules for policies.

Name	Type	Last Operated By	Last Operated At	Operation
Check_code	Check	[redacted]	Apr 07, 2025 15:03:46 GMT+08:00	✎ 🗑

----End

Step 2: Create a Policy and Add a Rule to the Policy

There are tenant-level policies and project-level policies. Tenant-level policies can be applied to pipelines of all projects under the current tenant, while project-level policies can be applied to all pipelines under the current project. The following uses a tenant-level policy as an example.

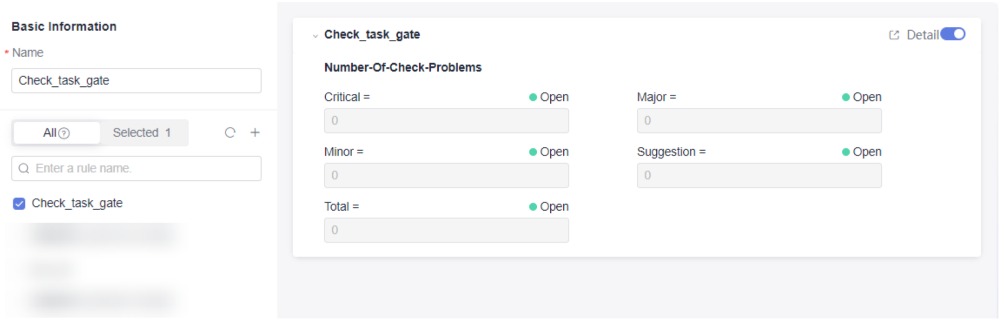
Step 1 In the navigation pane on the left, choose **Policies**.

NOTE

A system policy exists by default. You can view and use the policy, but cannot edit or delete it.

Step 2 Click **Create Policy**. On the displayed page, enter a policy name and select the rule created in **Step 1: Create a Rule and Configure Thresholds**.

Figure 3-3 Creating a policy



Basic Information

* Name

Check_task_gate

All | Selected 1

Enter a rule name.

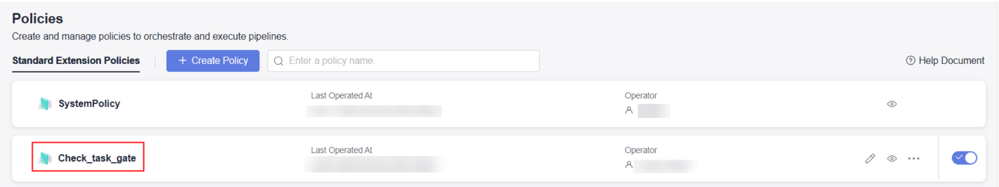
☑ Check_task_gate

Check_task_gate | Detail

Number-Of-Check-Problems

Critical =	● Open	Major =	● Open
0		0	
Minor =	● Open	Suggestion =	● Open
0		0	
Total =	● Open		
0			

Step 3 Click **OK**.



Policies

Create and manage policies to orchestrate and execute pipelines.

Standard Extension Policies | + Create Policy | Enter a policy name. | Help Document

SystemPolicy	Last Operated At	Operator	
Check_task_gate	Last Operated At	Operator	

----End

Step 3: Configure a Pipeline

Step 1 On the top navigation bar, click **Homepage**.

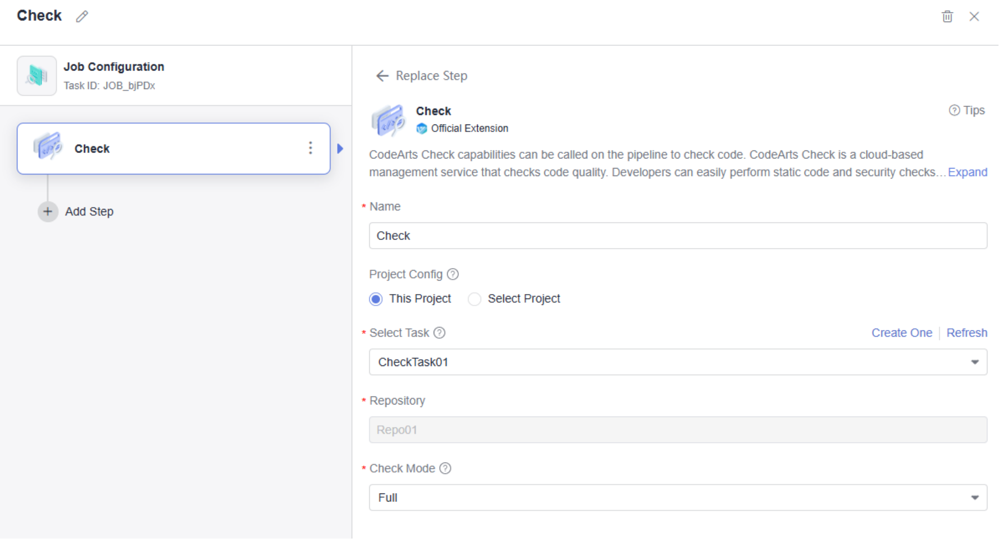
Step 2 Search for the project created in **Preparations** and access the project.

Step 3 In the navigation pane on the left, choose **CICD** > **Pipeline**.

Step 4 Search for the pipeline created in **Preparations**, click ··· in the **Operation** column, and select **Edit**. The **Task Orchestration** page is displayed.

Step 5 Click + Job under **Stage_1**, select **From empty**, add the code check task created in **Preparations**, set **Check Mode** to **Full**, and click **OK**.

Figure 3-4 Adding a code check task



Check

CodeArts Check capabilities can be called on the pipeline to check code. CodeArts Check is a cloud-based management service that checks code quality. Developers can easily perform static code and security checks...

* Name: Check

Project Config: ● This Project ○ Select Project

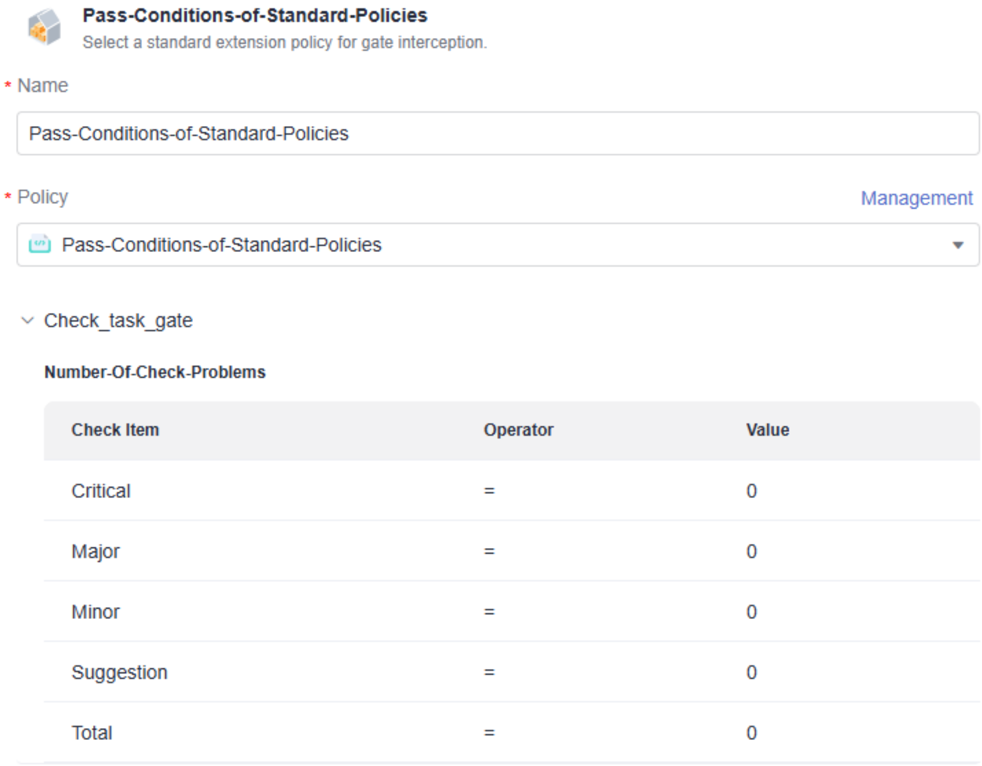
* Select Task: CheckTask01

* Repository: Repo01

* Check Mode: Full

Step 6 Click Pass Conditions under **Stage_1**, on the displayed window, add **Pass-Conditions-of-Standard-Policies**, and select the policy created in **Step 2: Create a Policy and Add a Rule to the Policy**.

Figure 3-5 Adding pass conditions



Pass-Conditions-of-Standard-Policies

Select a standard extension policy for gate interception.

* Name: Pass-Conditions-of-Standard-Policies

* Policy: Pass-Conditions-of-Standard-Policies

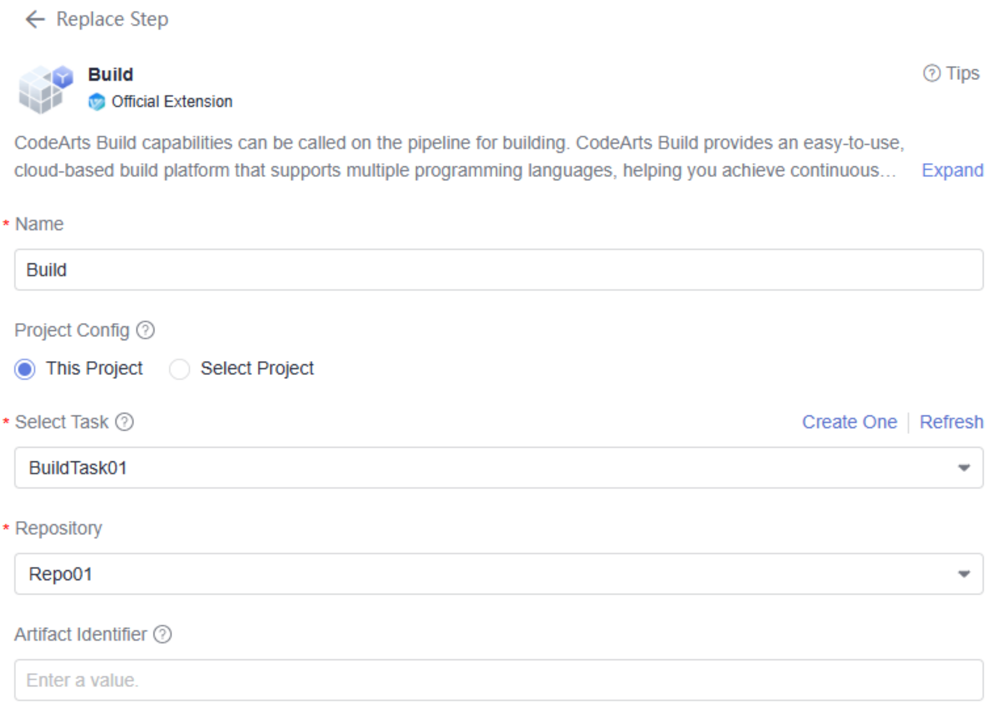
Check_task_gate

Number-Of-Check-Problems

Check Item	Operator	Value
Critical	=	0
Major	=	0
Minor	=	0
Suggestion	=	0
Total	=	0

Step 7 Click ⊕ or + **Stage** to add a new stage for the pipeline, add the build task created in **Preparations**, select the associated repository for the build task, and click **OK**.

Figure 3-6 Adding a build task



← Replace Step

Build
Official Extension

ⓘ Tips

CodeArts Build capabilities can be called on the pipeline for building. CodeArts Build provides an easy-to-use, cloud-based build platform that supports multiple programming languages, helping you achieve continuous... Expand

* Name

Build

Project Config ⓘ

● This Project ○ Select Project

* Select Task ⓘ Create One | Refresh

BuildTask01

* Repository

Repo01

Artifact Identifier ⓘ

Enter a value.

----End

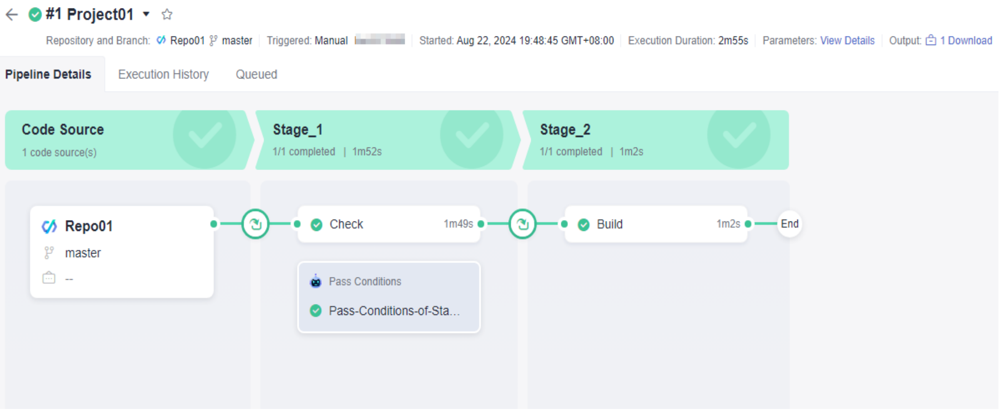
Step 4: Execute the Pipeline

Step 1 After configuring the pipeline, click **Save and Execute**.

Step 2 Check the execution result.

- If the code check job meets the pass conditions, the pipeline will proceed to the next stage, as shown in the following figure.

Figure 3-7 Executing a pipeline



← #1 Project01 ▾ ☆

Repository and Branch: Repo01 master | Triggered: Manual [redacted] | Started: Aug 22, 2024 19:48:45 GMT+08:00 | Execution Duration: 2m55s | Parameters: View Details | Output: 1 Download

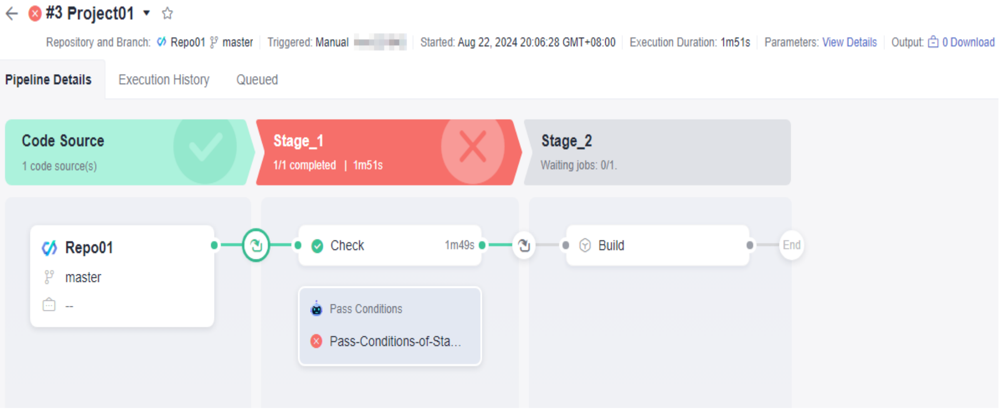
Pipeline Details | Execution History | Queued

Code Source — 1 code source(s)
Stage_1 — 1/1 completed | 1m52s
Stage_2 — 1/1 completed | 1m2s

Repo01 — master — --
✓ Check 1m49s
Pass Conditions
✓ Pass-Conditions-of-Sta...
✓ Build 1m2s
End

- If the code check task does not meet the pass conditions, the pipeline will stop running, as shown in the following figure. You can click the pass conditions card to check details.

Figure 3-8 Executing a pipeline



#3 Project01

Repository and Branch: Repo01 master | Triggered: Manual [redacted] | Started: Aug 22, 2024 20:06:28 GMT+08:00 | Execution Duration: 1m51s | Parameters: View Details | Output: 0 Download

Pipeline Details | Execution History | Queued

Code Source
1 code source(s)

Stage_1
1/1 completed | 1m51s

Stage_2
Waiting jobs: 0/1.

Repo01
master
--

Check 1m49s

Pass Conditions
Pass-Conditions-of-Sta...

Build

End

----End

4 Transferring CodeArts Pipeline Parameters to CodeArts Build and CodeArts Deploy

Overview

Pipeline parameters can be transferred among different services (such as CodeArts Build and CodeArts Deploy). By creating a CI/CD pipeline, you can streamline data of build and deployment.

Procedure

The following describes how to transfer a pipeline version number parameter to a build and a deployment job.

Table 4-1 Procedure

Step	Description
Create a build task	Create a build task, add the version number parameter, and reference it in the build step.
Create an application	Create an application, add a software package parameter, and reference it in the deployment step.
Create and execute a pipeline	Create a pipeline, add the version number parameter, and add the created build task and application to the pipeline. ● In the build task, reference the pipeline version number parameter. ● In the application, reference the pipeline version number parameter.
Check the build and deployment results	Check whether: ● The build package version number is a dynamic parameter transferred by the pipeline. ● The software package has been obtained by the deployment job.

Preparations

- You have enabled and authorized CodeArts Pipeline.
- You have created a project. The following uses a Scrum project named **Project_Test** as an example.
- You have created a code repository. The following uses a repository named **Repo_Test** (created using the **Java Maven Demo** template) as an example.
- You need to prepare a host with an EIP. You can use an existing host or purchase a Huawei Cloud ECS.

Step 1: Create a Build Task

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**.

Step 4 Click **Homepage** from the top navigation pane. Search for the project created in Preparations and access the project.

Step 5 In the left navigation pane, choose **CICD** > **Build**.

Step 6 Click **Create Task** and enter basic information.

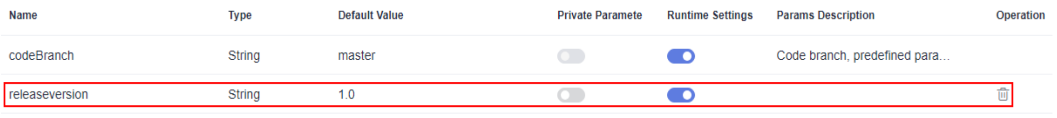
Table 4-2 Basic information

Parameter	Description
Name	Build task name. Enter **BuildTask01**.
Project	Keep the default value, which is the project of the build task.
Code Source	Code source associated with the build task. Select **Repo**.
Repository	Code repository associated with the build task. Select the code repository **Repo_Test** created in Preparations.
Default Branch	Select **master**.

Step 7 Click **Next**, select the **Maven** template, and then click **OK**.

Step 8 On the **Parameters** tab page, add the **releaseversion** parameter, and toggle on **Runtime Settings**.

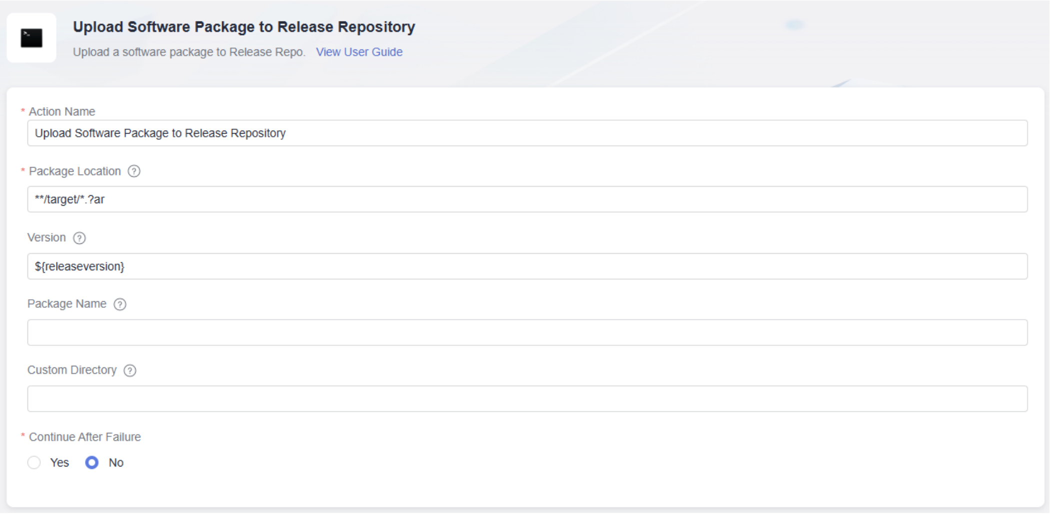
Figure 4-1 Creating a build parameter



Name	Type	Default Value	Private Paramete	Runtime Settings	Params Description	Operation
codeBranch	String	master	○	●	Code branch, predefined para...	
releaseversion	String	1.0	○	●		🗑

Step 9 On the **Build Actions** page, click **Upload Software Package to Release Repository**. For the **Version** field, enter **${releaseversion}**, and retain the default values for other fields.

Figure 4-2 Configuring build actions



Upload Software Package to Release Repository

Upload a software package to Release Repo. View User Guide

* Action Name: Upload Software Package to Release Repository

* Package Location: **/target/*.?ar

Version: ${releaseversion}

Package Name:

Custom Directory:

* Continue After Failure: ○ Yes ● No

Step 10 Click **Save**.

----End

Step 2: Create an application

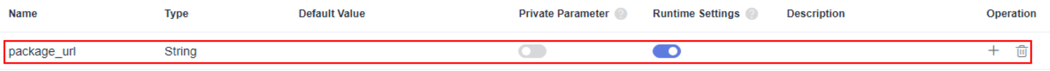
Step 1 In the left navigation pane, choose **Settings** > **General** > **Basic Resources**. On the displayed page, click **Create Host cluster** and add the purchased host to the cluster.

Step 2 In the left navigation pane, choose **CICD** > **Deploy**.

Step 3 Click **Create Application**. On the displayed page, enter an application name **DeployTask01**, click **Next**, select **Blank Template**, and click **OK**.

Step 4 Click the **Parameters** tab. On the displayed page, create a custom parameter **package_url**, and toggle on **Runtime Settings**.

Figure 4-3 Creating a deployment parameter

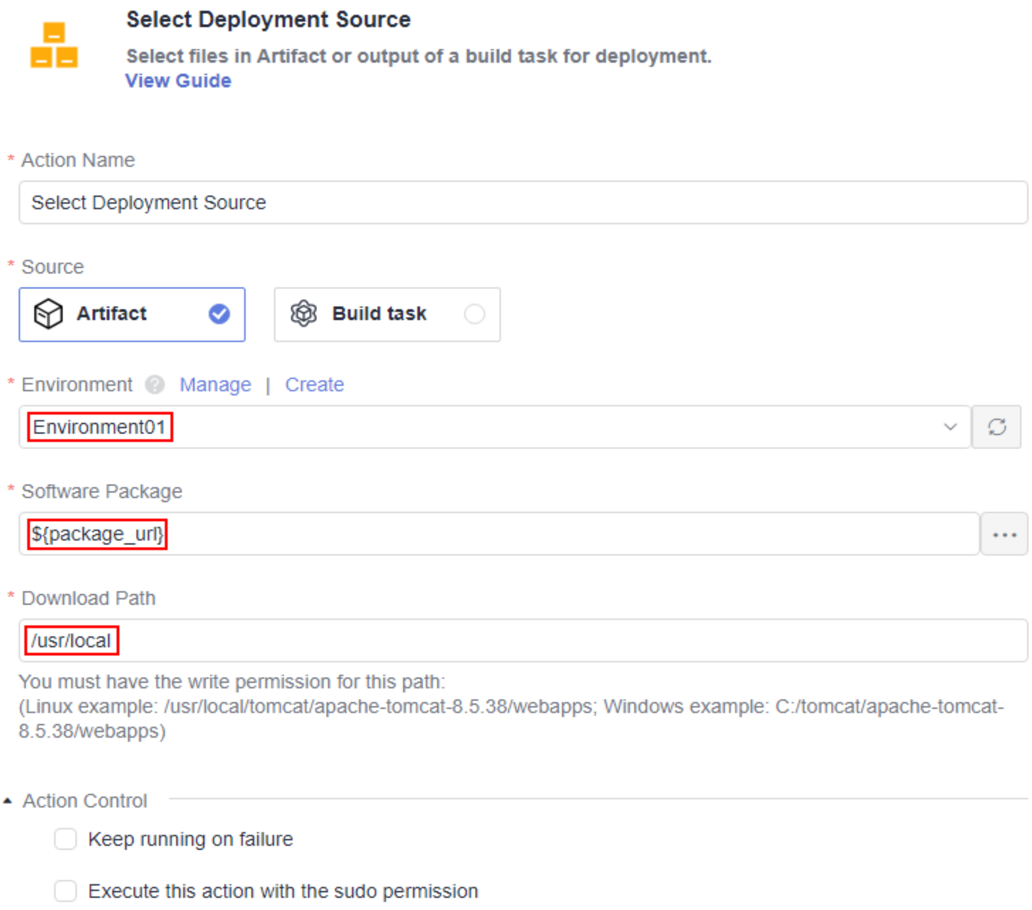


Name	Type	Default Value	Private Parameter	Runtime Settings	Description	Operation
package_url	String		○	●		+ 🗑

Step 5 Click the **Environment Management** tab. On the displayed page, click **Create Environment**, enter an environment name **Environment01**, and click **Save**. On the displayed **Resources** page, click **Import Host** to import the host to the environment.

Step 6 Click the **Deployment Actions** tab. On the displayed page, configure the **Select Deployment Source** action as shown in **Table 4-3**.

Figure 4-4 Configuring deployment actions



Select Deployment Source

Select files in Artifact or output of a build task for deployment.
View Guide

* Action Name: Select Deployment Source

* Source: Artifact ● | Build task ○

* Environment (Manage | Create): Environment01

* Software Package: ${package_url}

* Download Path: /usr/local

You must have the write permission for this path:
(Linux example: /usr/local/tomcat/apache-tomcat-8.5.38/webapps; Windows example: C:/tomcat/apache-tomcat-8.5.38/webapps)

Action Control

□ Keep running on failure

□ Execute this action with the sudo permission

Table 4-3 Configuring deployment actions

Parameter	Description
Action Name	Retain the default value.
Source	Software package source. Select **Artifact**.
Environment	Environment for deployment. Select **Environment01**.
Software Package	Software package to be deployed. Obtain the build package uploaded by the build task to Release Repos. Set this parameter to **${package_url}** to reference the **package_url** parameter.
Download Path	Path for downloading the software package to the target host. Enter **/usr/local**.
Action Control	Retain the default setting.

Step 7 Click **Save**.

----End

Step 3: Create and Execute a Pipeline

Step 1 In the navigation pane on the left, choose **CICD** > **Pipeline**.

Step 2 Click **Create Pipeline** and configure the pipeline information as shown in **Table 4-4**.

Table 4-4 Pipeline information

Parameter	Description
Name	Enter **Pipeline-Test**.
Code Source	Code source associated with the pipeline. Select **Repo**.
Repository	Code repository associated with the pipeline. Select the code repository **Repo_Test** created in **Preparations**.
Default Branch	Select **master**.

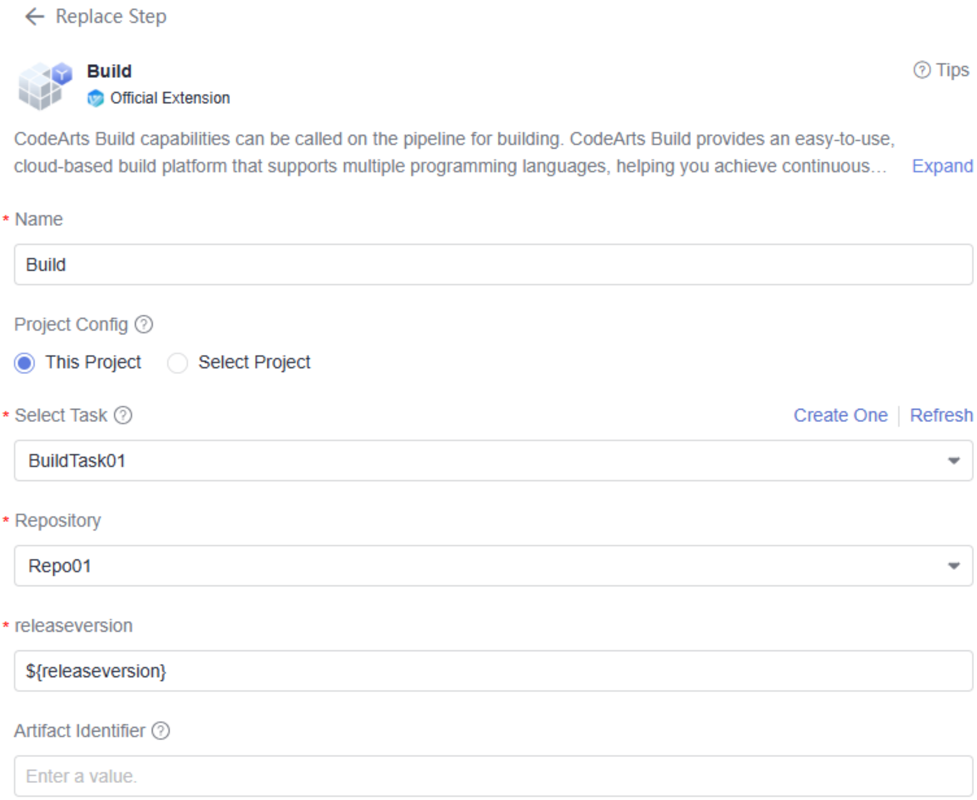
Step 3 After configuring the basic information, click **Next**. On the displayed **Select Template** page, select **Blank Template** and click **OK**. The **Task Orchestration** page is displayed.

Step 4 Click the **Parameter Configuration** tab. On the displayed page, create a custom parameter **releaseversion**, set its default value to **${TIMESTAMP}**, and click the toggle to enable **Runtime Setting**.

Step 5 Click the **Task Orchestration** page. On the displayed page, two stages (**Pipeline Source** and **Stage_1**) are generated by default. Click **Stage** to add a new stage (**Stage_2**).

Step 6 Click **Job** under the **Stage_1** and select **From empty**. Search for the **Build** extension, move the cursor to this extension, click **Add**, select the **created build task**, select the repository associated with the build task, and set **releaseversion** to **${releaseversion}** to reference the **releaseversion** parameter of the pipeline.

Figure 4-5 Adding a build task



CodeArts Build capabilities can be called on the pipeline for building. CodeArts Build provides an easy-to-use, cloud-based build platform that supports multiple programming languages, helping you achieve continuous...

* Name: Build

Project Config: ● This Project ○ Select Project

* Select Task: BuildTask01

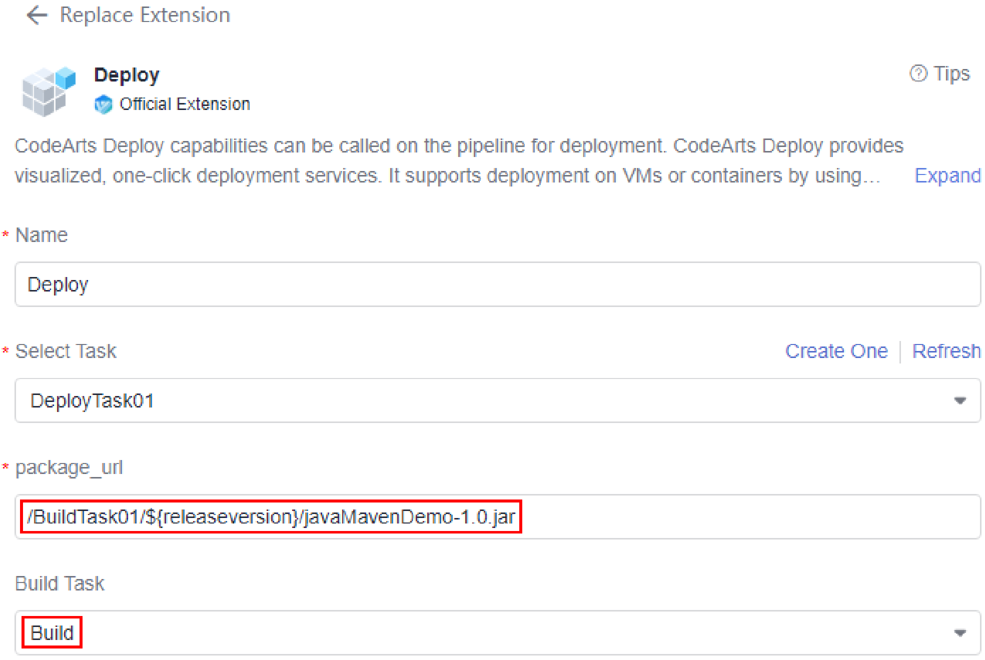
* Repository: Repo01

* releaseversion: ${releaseversion}

Artifact Identifier: Enter a value.

Step 7 Click **Job** under the **Stage_2** stage and select **From empty**. In the displayed window, search for the **Deploy** extension, move the cursor to this extension, click **Add**, select the **created application**, enter the **package_url** path by referring to **Figure 4-6**, and associate with the build task in **Step 6**.

Figure 4-6 Add an application



Deploy
Official Extension

CodeArts Deploy capabilities can be called on the pipeline for deployment. CodeArts Deploy provides visualized, one-click deployment services. It supports deployment on VMs or containers by using...

* Name: Deploy

* Select Task: DeployTask01

* package_url: /BuildTask01/${releaseversion}/javaMavenDemo-1.0.jar

Build Task: Build

NOTE

package_url is the relative path of the build package in **Release Repos**. The path includes the build task name, version number, and package name. In this section, the pipeline **releaseversion** parameter indicates the version number.

Step 8 Click **Save and Execute**.

----End

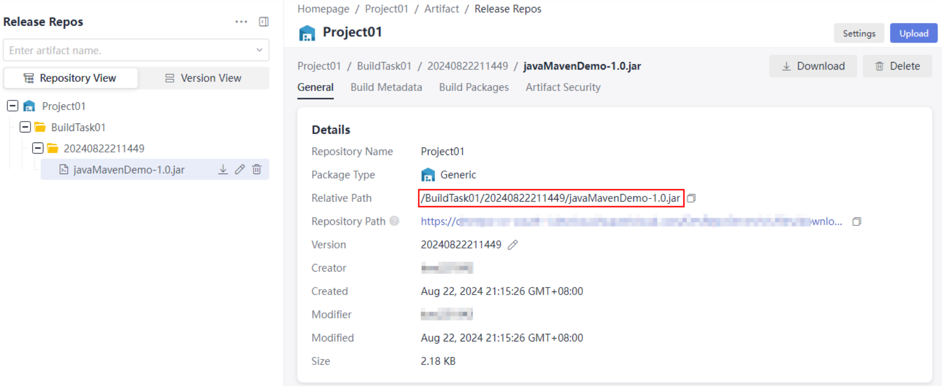
Step 4: Check Build and Deployment Results

After the pipeline is successfully executed, check whether the pipeline parameter has been transferred to the build and deployment jobs.

- Check the build result
  a. In the navigation pane on the left, choose **Artifact** > **Release Repos**.
  b. Expand the project navigation tree on the left to check the uploaded build package.

     As shown in the following figure, the version number in the relative path is the timestamp transferred by the pipeline **releaseversion** parameter.

Figure 4-7 Checking the software package



Details

Repository Name Project01

Package Type Generic

Relative Path /BuildTask01/20240822211449/javaMavenDemo-1.0.jar

Repository Path https://c[redacted]wnlo...

Version 20240822211449

Creator [redacted]

Created Aug 22, 2024 21:15:26 GMT+08:00

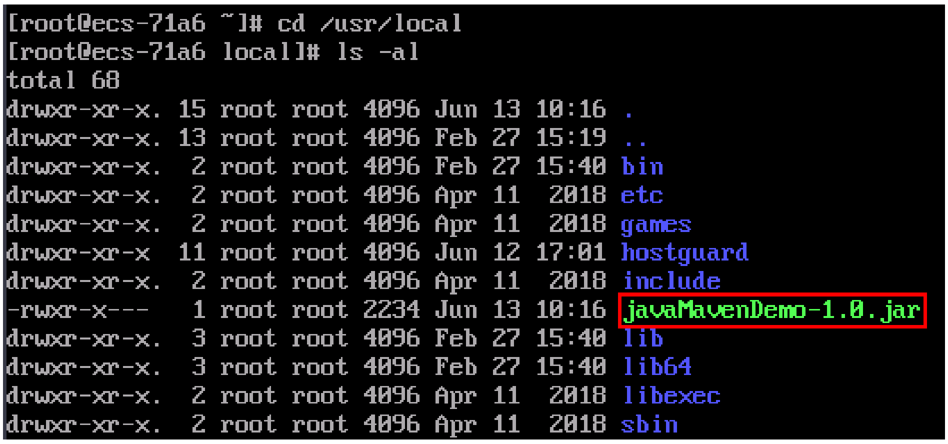
Modifier [redacted]

Modified Aug 22, 2024 21:15:26 GMT+08:00

Size 2.18 KB

- Check the deployment result
  a. Click the username in the upper right corner and choose **CodeArts Console** to access the console.
  b. Click ☰ in the upper left corner, search for **Elastic Cloud Server**, and click it to access its console.
  c. Locate the ECS used for deployment, click **Remote Login** in the **Operation** column.
  d. In the **Other Login Modes** area, select **Log in using Remote Login on the management console** and click **Log In**.
  e. Enter the username and password for purchasing the ECS. Press **Enter**.
  f. Enter the following command and press **Enter** to go to the directory **/usr/local** configured during the **Create an Application** step.
     ```
     cd /usr/local
     ```
  g. Enter the following command and press **Enter**. The deployed build package is displayed as shown in the following figure, which indicates that the pipeline parameter has been successfully transferred.
     ```
     ls -al
     ```

Figure 4-8 Checking the deployment result



```
[root@ecs-71a6 ~]# cd /usr/local
[root@ecs-71a6 local]# ls -al
total 68
drwxr-xr-x. 15 root root 4096 Jun 13 10:16 .
drwxr-xr-x. 13 root root 4096 Feb 27 15:19 ..
drwxr-xr-x.  2 root root 4096 Feb 27 15:40 bin
drwxr-xr-x.  2 root root 4096 Apr 11  2018 etc
drwxr-xr-x.  2 root root 4096 Apr 11  2018 games
drwxr-xr-x  11 root root 4096 Jun 12 17:01 hostguard
drwxr-xr-x.  2 root root 4096 Apr 11  2018 include
-rwxr-x---   1 root root 2234 Jun 13 10:16 javaMavenDemo-1.0.jar
drwxr-xr-x.  3 root root 4096 Feb 27 15:40 lib
drwxr-xr-x.  3 root root 4096 Feb 27 15:40 lib64
drwxr-xr-x.  2 root root 4096 Apr 11  2018 libexec
drwxr-xr-x.  2 root root 4096 Apr 11  2018 sbin
```

5 Creating Tags for Code Repositories Through Pipelines

Overview

Contexts are a way to access information about pipeline runs, sources, variables, and jobs. Each context is an object that contains various attributes. You can use pipeline contexts to transfer information among jobs to streamline a pipeline.

The following describes how to create a repository tag through the pipeline contexts.

Preparations

- **You have enabled and authorized CodeArts Pipeline**.
- **You have created a project**. The following uses a Scrum project named **Project_Test** as an example.
- **You have created a code repository** and created a branch. The following uses a repository named **Repo_Test** (created using the **Java Maven Demo** template) and a branch named **release-1.0.0** as an example.

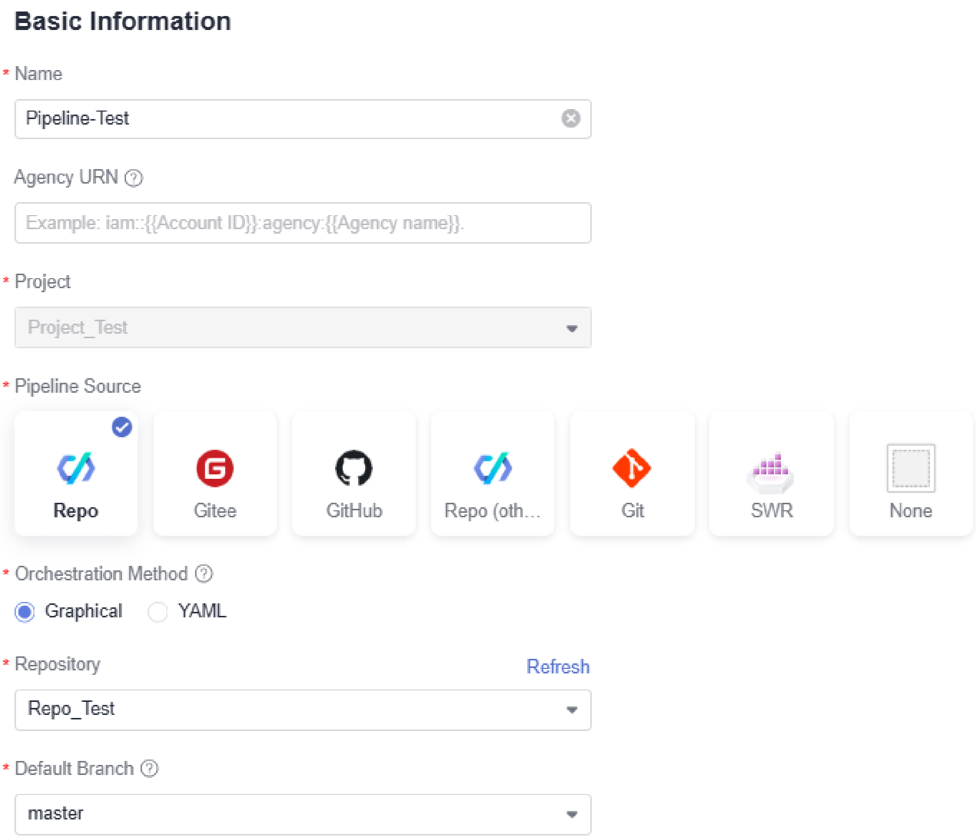
Procedure

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**.

Step 4 Click **Create Pipeline** and configure parameters by referring to **Table 5-1**.



Basic Information

* Name

Pipeline-Test

Agency URN

Example: iam::{{Account ID}}:agency:{{Agency name}}.

* Project

Project_Test

* Pipeline Source

* Orchestration Method

● Graphical ○ YAML

* Repository

Repo_Test

* Default Branch

master

Table 5-1 Pipeline basic information

Parameter	Description
Name	Pipeline name. Enter a maximum of 128 characters, including letters, digits, underscores (_), and hyphens (-). For example, **Pipeline-Test**.
Project	Project to which the pipeline belongs. Select the project **Project_Test** created in **Preparations**.
Code Source	Code source associated with the pipeline. Select **Repo**.
Repository	Code repository associated with the pipeline. Select the code repository **Repo_Test** created in **Preparations**.
Default Branch	Select the branch **release-1.0.0** created in **Preparations**.

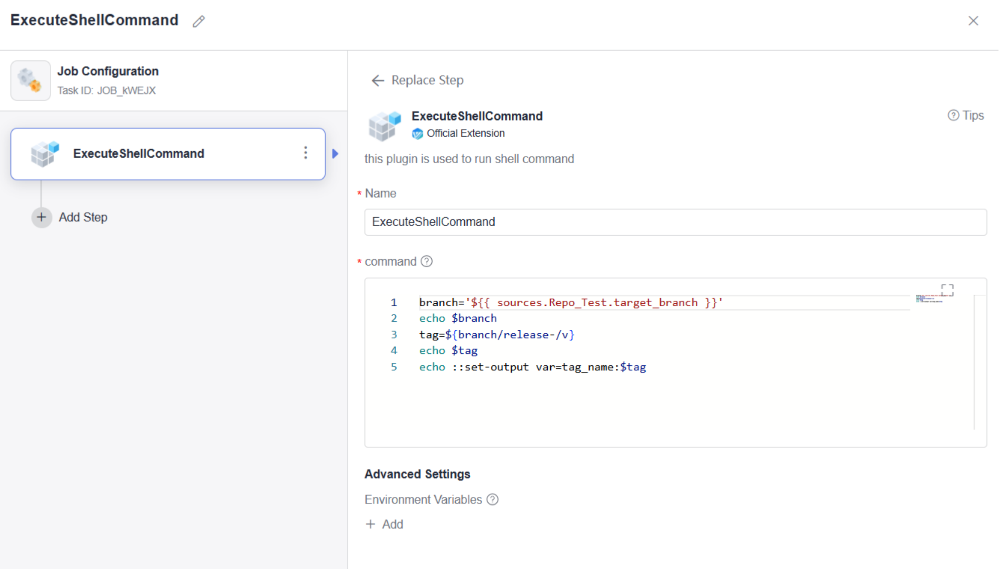
Step 5 After configuring the basic information, click **Next**. On the displayed **Select Template** page, select **Blank Template** and click **OK**.

Step 6 Access the **Task Orchestration** page. Two stages (**Pipeline Source** and **Stage_1**) are generated by default. Click **Stage** to add a new stage (**Stage_2**).

Step 7 Click **New Job** under **Stage_1** and select **From empty**. In the displayed window, search for the **ExecuteShellCommand** extension, move the cursor to this extension, click **Add**, retain the default name, and enter the following shell command to generate a tag name:

```
branch='${{ sources.Repo_Test.target_branch }}'    //Obtain the name of the running branch.
echo $branch                                        //Print the branch name.
tag=${branch/release-/v}                            //Rename the branch. (Here we customize the branch name release-1.0.0 as v1.0.0.)
echo $tag                                           //Print the tag name.
echo ::set-output var=tag_name:$tag                 //Generate an output tag_name and set it as a context for future use.
```

Figure 5-1 Generating a tag name



ExecuteShellCommand

Job Configuration
Task ID: JOB_kWEJX

ExecuteShellCommand

Add Step

Replace Step

ExecuteShellCommand
Official Extension

Tips

this plugin is used to run shell command

* Name
ExecuteShellCommand

* command

```
branch='${{ sources.Repo_Test.target_branch }}'
echo $branch
tag=${branch/release-/v}
echo $tag
echo ::set-output var=tag_name:$tag
```

Advanced Settings

Environment Variables

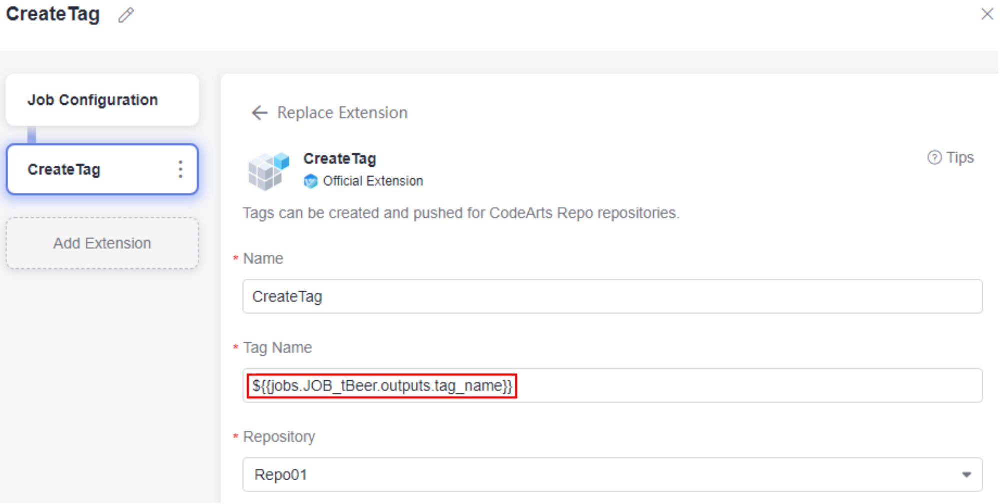
+ Add

Step 8 Click **Job** under **Stage_2** and select **From empty**. In the displayed window, search for the **CreateTag** extension, move the cursor to this extension, click **Add**, and set parameters by referring to **Table 5-2**.

Table 5-2 Parameter description

Parameter	Description
Name	Retain the default name.
Tag Name	Enter **${{jobs.JOB_tBeer.outputs.tag_name}}**, where **JOB_tBeer** indicates the task ID of the added **ExecuteShellCommand** extension.
Repository	Select the code repository associated with the pipeline.

Figure 5-2 Creating a repository tag



Tags can be created and pushed for CodeArts Repo repositories.

* Name: CreateTag

* Tag Name: ${{jobs.JOB_tBeer.outputs.tag_name}}

* Repository: Repo01

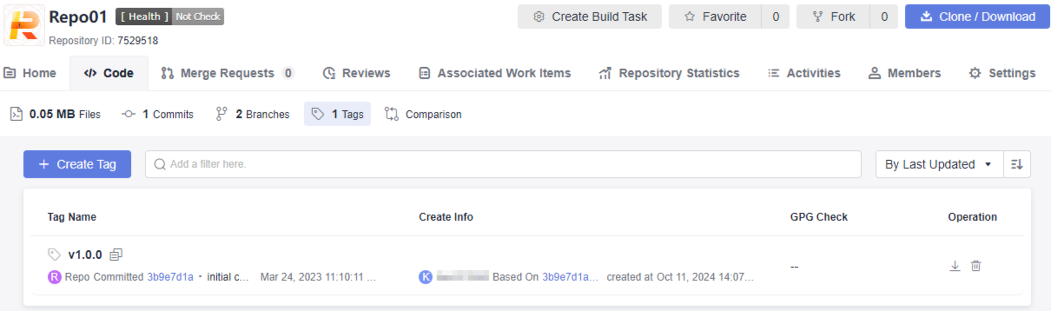
Step 9 After the configuration is complete, click **Save and Execute**. In the displayed window, retain the default settings and click **Execute**.

Step 10 After the pipeline execution is complete, choose **Code** > **Repo** from the left navigation pane.

Step 11 Click the repository associated with the pipeline.

Step 12 On the displayed **Code** page, click the **Tags** tab. The tag **v1.0.0** is displayed.

Figure 5-3 Checking a tag



Repository ID: 7529518

Tag Name	Create Info	GPG Check	Operation
v1.0.0 Repo Committed 3b9e7d1a · initial c... Mar 24, 2023 11:10:11 ...	[redacted] Based On 3b9e7d1a... created at Oct 11, 2024 14:07...	--	

----End

6 Configuring a Pipeline Gate for a Code Repository Merge Request

Overview

You can set an automated pipeline gate for merge requests in code repositories. By doing so, every merge request that triggers a pipeline will go through code checks, builds, or tests in the pipeline. Only code that has passed strict code review can be merged into the main branch. This mechanism ensures code stability and reliability, and effectively reduces production environment issues caused by code defects. Overall, this solution improves code quality and CI/CD efficiency.

Preparations

- **You have enabled and authorized CodeArts Pipeline**.
- **You have created a project**. The following uses a Scrum project named **Project_Test** as an example.
- **You have created a code repository** and created a branch. The following uses a repository named **Repo_Test** (created using the **Java Maven Demo** template) and a branch named **release-1.0.0** as an example.

Process

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**.

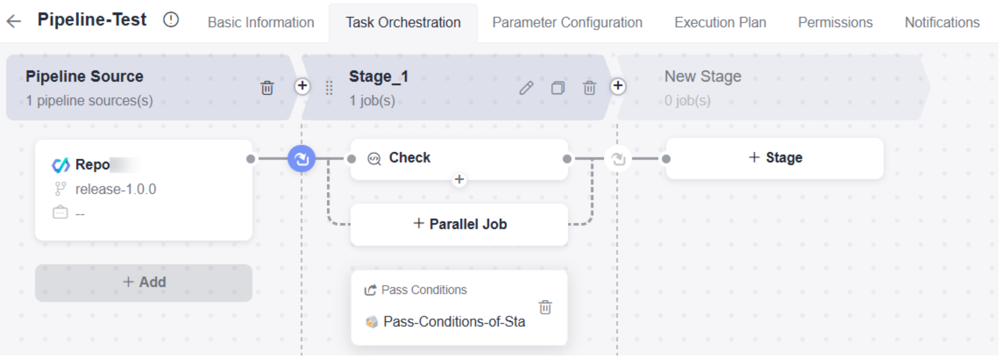
Step 4 Click **Create Pipeline** and configure pipeline information as shown in **Table 6-1**.

Table 6-1 Parameters

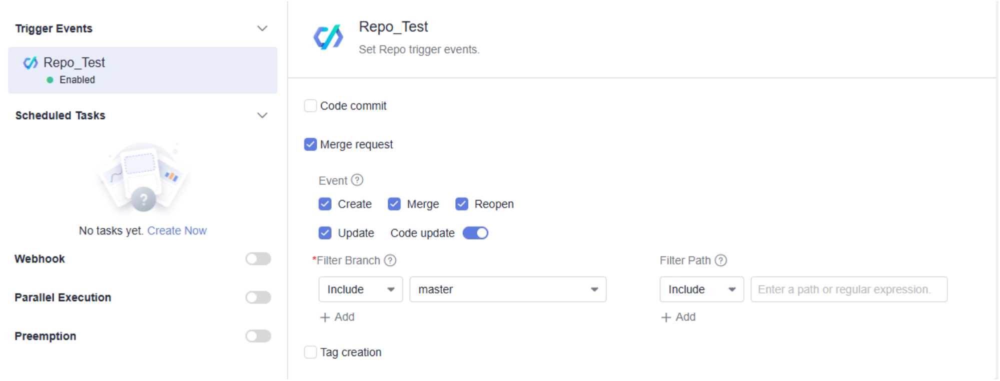
Parameter	Example Value	Description
Name	Pipeline-Test	Pipeline name.
Project	Project_Test	Project that a pipeline belongs to.
Pipeline Source	Repo	Pipeline source associated with the pipeline.
Repository	Repo_Test	Code repository associated with the pipeline.
Default Branch	release-1.0.0	Default branch of the repository.

Step 5 After configuring the basic information, click **Next**. On the displayed **Select Template** page, select **Blank Template** and click **OK**.

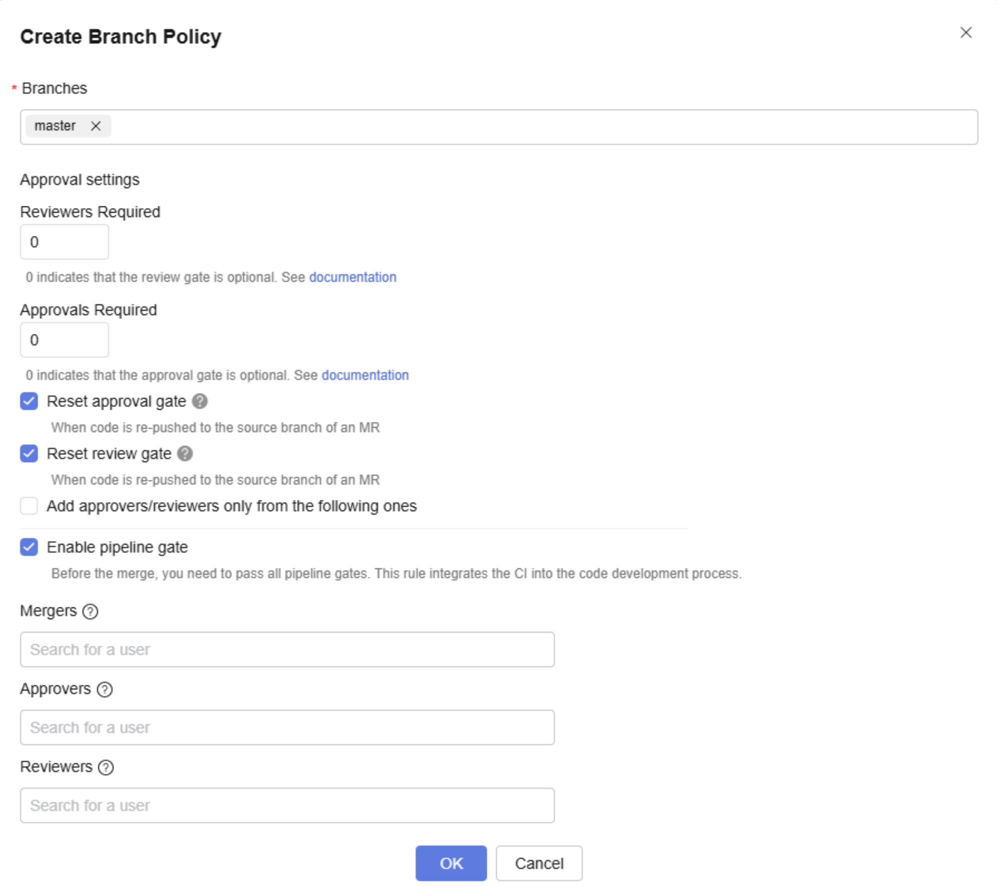
Step 6 Go to the **Task Orchestration** page. Two stages (**Pipeline Source** and **Stage_1**) are generated by default. You can add code check or build jobs as needed.



Step 7 Click the **Execution Plan** tab and configure Repo trigger events. Select **Merge request**, set events, filter branches, and filter paths. For details, see **Configuring Event Triggers**.



Step 8 Access the **Repo_Test** repository created in **Preparations**, choose **Settings > Policy Settings > Merge Requests**. In the **Branch Policies** area, click **Create Branch Policy**, select or enter a branch, select **Enable pipeline gate**, and click **OK**.



Create Branch Policy

* Branches

master ×

Approval settings

Reviewers Required

0

0 indicates that the review gate is optional. See documentation

Approvals Required

0

0 indicates that the approval gate is optional. See documentation

☑ Reset approval gate

When code is re-pushed to the source branch of an MR

☑ Reset review gate

When code is re-pushed to the source branch of an MR

□ Add approvers/reviewers only from the following ones

☑ Enable pipeline gate

Before the merge, you need to pass all pipeline gates. This rule integrates the CI into the code development process.

Mergers

Search for a user

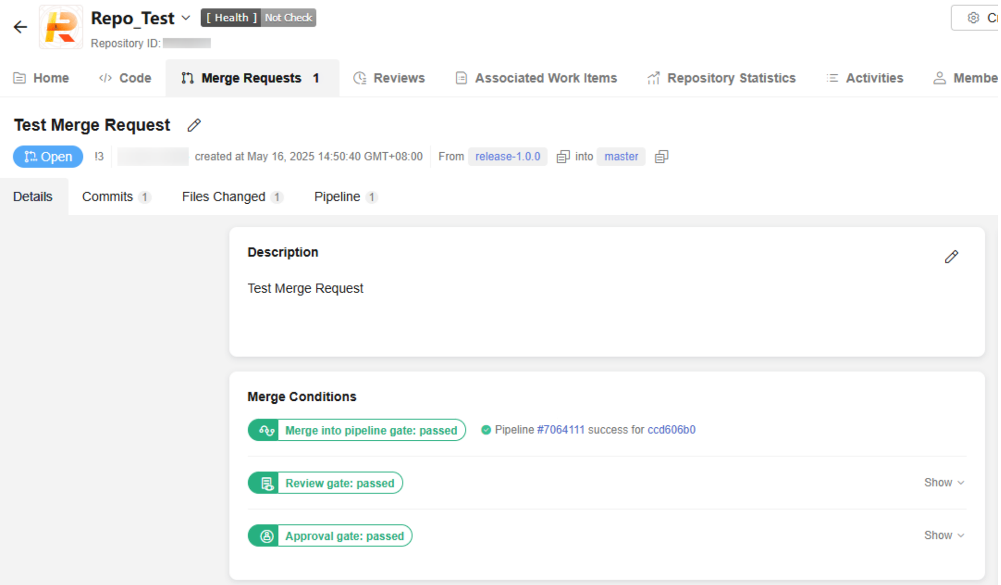
Approvers

Search for a user

Reviewers

Search for a user

Step 9 Click the **Merge Requests** tab. On the displayed page, create a merge request in **Repo_Test** to trigger the pipeline. The pipeline running status will be displayed in the **Merge Conditions** area.



Test Merge Request

Open !3 [redacted] created at May 16, 2025 14:50:40 GMT+08:00 From release-1.0.0 into master

Description

Test Merge Request

Merge Conditions

Merge into pipeline gate: passed — Pipeline #7064111 success for ccd606b0

Review gate: passed — Show

Approval gate: passed — Show

----End

7 Managing Pipeline Permissions

Overview

You can manage CodeArts Pipeline's project-level and resource-level permissions. For details, see **Authorizing CodeArts Pipeline**. The following describes how to configure resource-level permissions for a single pipeline or multiple pipelines at once.

Preparations

- **You have enabled and authorized CodeArts Pipeline**.
- **You have created a project**. The following uses a Scrum project named **Project01** as an example.
- **You have created a code repository**. The following uses a repository named **Repo01** (created using the **Java Maven Demo** template) as an example.

Procedure

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

Step 2 Click **Create Pipeline** and configure parameters by referring to **Table 7-1**.

Table 7-1 Pipeline information

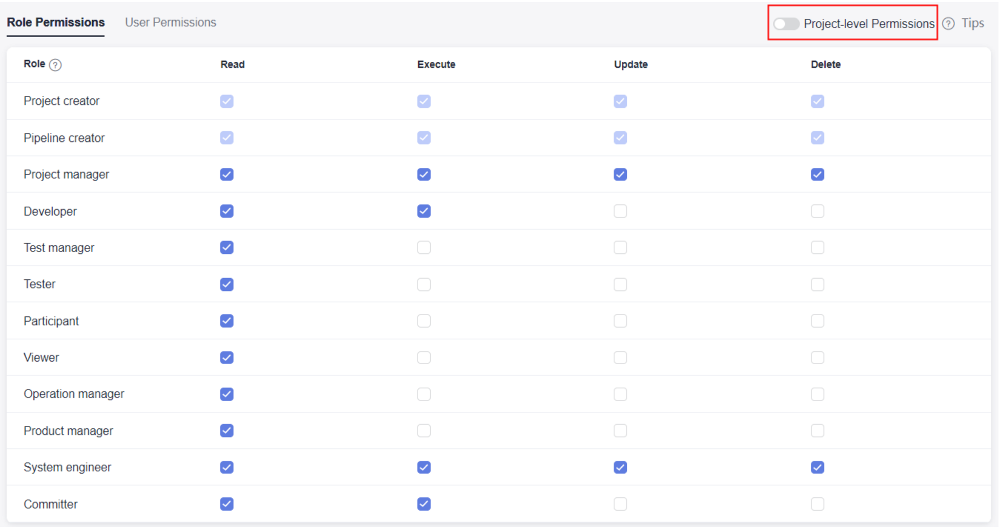
Parameter	Example Value	Description
Name	Pipeline01	Pipeline name. Enter a maximum of 128 characters, including letters, digits, underscores (_), and hyphens (-).
Project	Project01	Project that a pipeline belongs to.
Pipeline Source	Repo	Code source associated with the pipeline.
Orchestration Method	Graphical	Method of orchestrating a pipeline.

Parameter	Example Value	Description
Repository	Repo01	Code repository associated with the pipeline.
Default Branch	master	Default branch of the repository.

Step 3 After configuring the basic information, click **Next**. On the displayed **Select Template** page, select **Blank Template** and click **OK**. The **Task Orchestration** page is displayed.

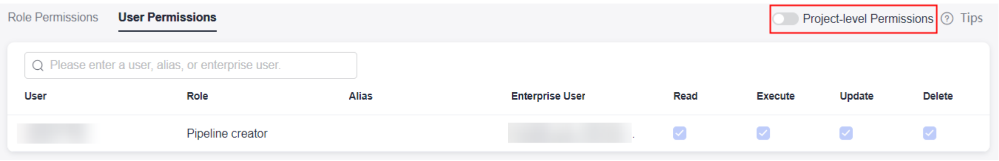
Step 4 Click the **Permissions** tab. On the displayed page, **Project-level Permissions** is enabled by default, which means that project-level permissions will be synchronized to the pipeline. Click ●. In the displayed dialog box, click **OK** to disable **Project-level Permissions**. Then you can customize role and user permissions of the pipeline.

Figure 7-1 Configuring role permissions



Role	Read	Execute	Update	Delete
Project creator	☑	☑	☑	☑
Pipeline creator	☑	☑	☑	☑
Project manager	☑	☑	☑	☑
Developer	☑	☑	□	□
Test manager	☑	□	□	□
Tester	☑	□	□	□
Participant	☑	□	□	□
Viewer	☑	□	□	□
Operation manager	☑	□	□	□
Product manager	☑	□	□	□
System engineer	☑	☑	☑	☑
Committer	☑	☑	□	□

Figure 7-2 Configuring user permissions

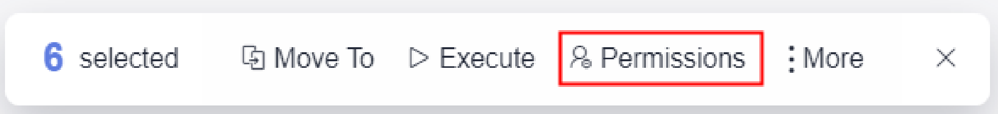


Role Permissions
User Permissions
Project-level Permissions
Tips
Please enter a user, alias, or enterprise user.

User	Role	Alias	Enterprise User	Read	Execute	Update	Delete
[redacted]	Pipeline creator		[redacted]	☑	☑	☑	☑

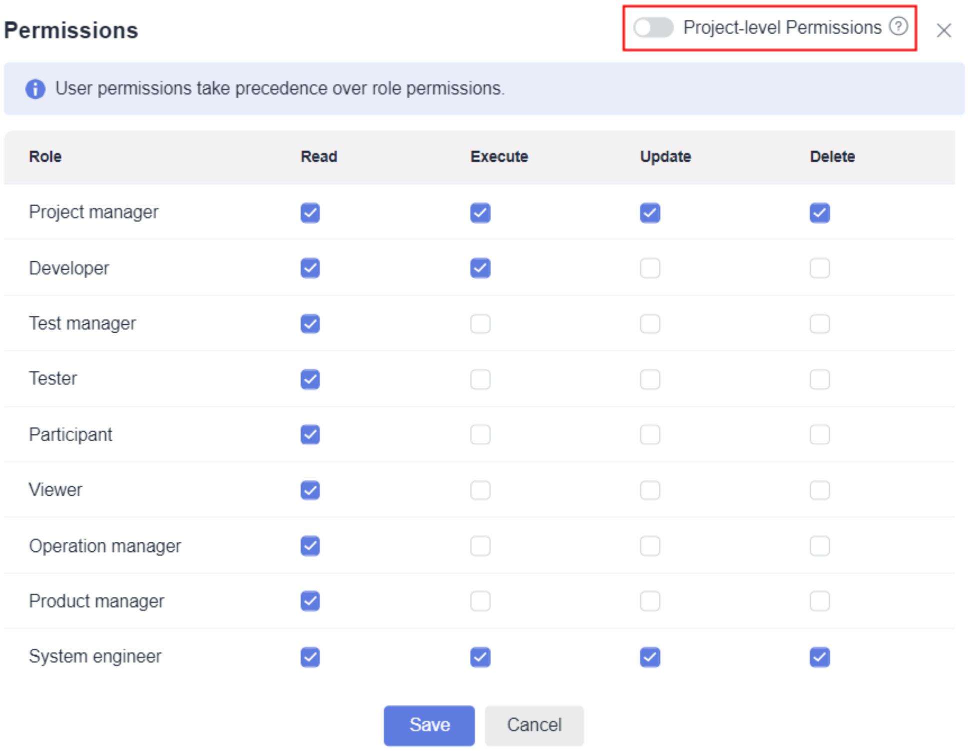
Step 5 Click **Save** to save the configuration. Then click ← in the upper left corner to return to the pipeline list page.

Step 6 Select desired pipelines and click **Permissions** in the lower part of the page. In the displayed window, set permissions for selected pipelines.



Step 7 Click to enable **Project-level Permissions** to inherit project-level permissions. You can also disable **Project-level Permissions** to set role permissions for selected pipelines.

Figure 7-3 Managing permissions



User permissions take precedence over role permissions.

Role	Read	Execute	Update	Delete
Project manager	☑	☑	☑	☑
Developer	☑	☑	□	□
Test manager	☑	□	□	□
Tester	☑	□	□	□
Participant	☑	□	□	□
Viewer	☑	□	□	□
Operation manager	☑	□	□	□
Product manager	☑	□	□	□
System engineer	☑	☑	☑	☑

Step 8 Click **Save**.

----End

8 HE2E DevOps Practice: Configuring a Pipeline

This section describes how to connect code check, build, and deployment tasks in **DevOps Full-Process Sample Project** for continuous delivery.

Before the practice, perform the **deployment**.

Introduction to Preset Pipelines

There are five pipeline tasks preset in the sample project. You can view and use them as needed.

Table 8-1 Preset pipeline tasks

Preset Pipeline Task	Description
phoenix-workflow	Implements basic functions.
phoenix-workflow-test	Runs in the test environment.
phoenix-workflow-work	Implements the Worker function.
phoenix-workflow-result	Implements the Result function.
phoenix-workflow-vote	Implements the Vote function.

Configuring and Executing a Pipeline

Assume that you use the **phoenix-cd-cce** application for deployment. Clear the workloads in the cluster before executing a pipeline.

Step 1 Go to the **Phoenix Mall** project, and choose **CICD** > **Pipeline** from the navigation pane.

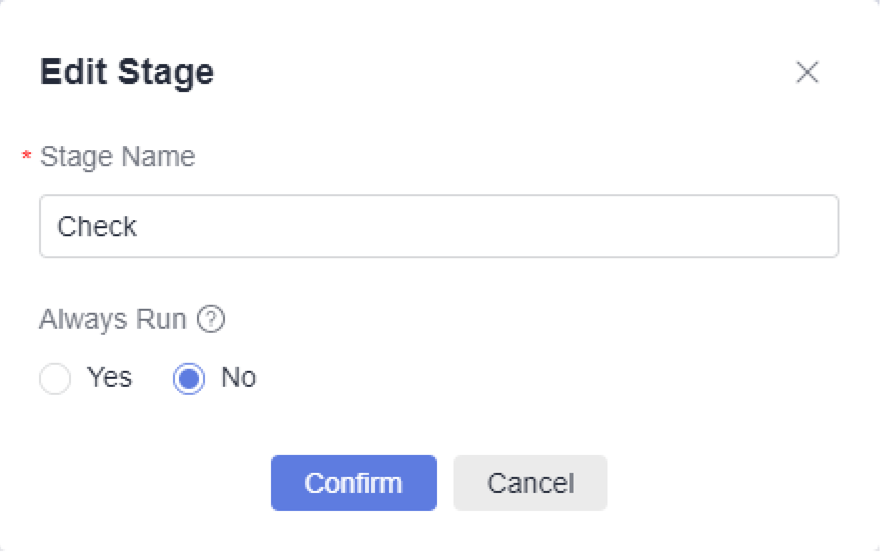
Step 2 In the **Operation** column of the **phoenix-workflow** pipeline, click ··· and choose **Edit**.

Step 3 Click the **Parameters** tab, and verify that the default values of **dockerOrg** and **dockerServer** are the same as those of the **phoenix-sample-ci** build task.

Step 4 Add a code check stage.

1. Click the **Task Orchestration** tab and click ⊕ between **Pipeline Source** and **Build**. A new stage **Stage_1** is displayed.
2. Click ✎ next to **Stage_1**. In the **Edit Stage** window, enter the stage name **Check** and click **Confirm**.

Figure 8-1 Editing the stage name



Edit Stage

* Stage Name

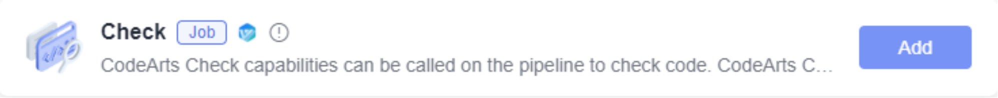
Check

Always Run

○ Yes ● No

3. Click **Job**, and select **From empty**. The job creation window is displayed on the right.
4. Find the **Check** job in the list, and click **Add**.

Figure 8-2 Adding the Check job



Check Job

CodeArts Check capabilities can be called on the pipeline to check code. CodeArts C...

5. Select the **phoenix-codecheck-worker** task and click **OK**.

   The **Check** job is displayed.

Step 5 Configure a deployment task.

Click the deployment task name, select the associated build task **phoenix-sample-ci**, and check the values of configuration items.

- The configurations of task **phoenix-sample-standalone** must be the same as those on the **Parameters** page of the task with the same name in CodeArts Deploy.
- The configurations of task **phoenix-cd-cce** must be the same as those on the **Parameters** page of the task with the same name in CodeArts Deploy.

NOTE

Two deployment tasks are added in this example. If you selected only one deployment mode in preceding steps, keep the corresponding task and delete the other one.

Step 6 Click **Save and Execute**. In the displayed dialog box, click **Execute** to start the pipeline.

If ✓ is displayed, the pipeline is successfully executed.

If the pipeline fails, click the cause to view logs. Then rectify the fault by referring to **CodeArts Pipeline FAQs**.

----End

Configuring Pass Conditions

To control the code quality, the code must be scanned and the number of errors must be within a reasonable range before being released. By adding quality gates, you can effectively automate the control process.

Step 1 Go to the **Phoenix Mall** project, and choose **CICD** > **Pipeline** from the navigation pane.

Step 2 In the **Operation** column of the **phoenix-workflow** pipeline, click ··· and choose **Edit**.

Step 3 Click the **Task Orchestration** tab. In the **Check** stage, click **Pass Conditions**. The **Pass Conditions** window is displayed on the right.

Step 4 Click **Add** next to **Pass-Conditions-of-Standard-Policies**.

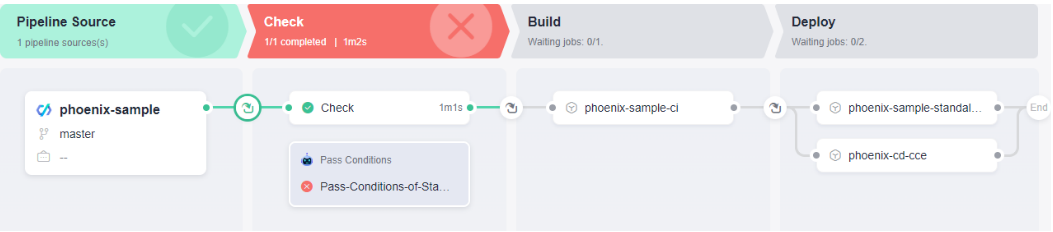
Step 5 Select **SystemPolicy** and click **OK**.

The new pass condition is displayed.

Step 6 Click **Save and Execute**.

If the number of check issues does not meet the pass conditions, the pipeline will fail.

Figure 8-3 Executing a pipeline



----End

Configuring Code Changes to Automatically Trigger a Pipeline

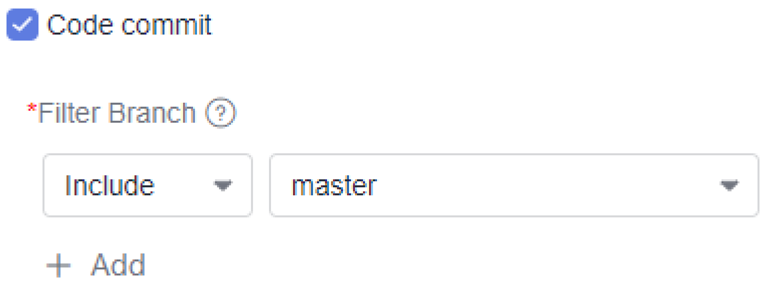
Through the following configuration, code changes can automatically trigger pipeline execution, implementing continuous project delivery.

Step 1 Go to the **Phoenix Mall** project, and choose **CICD** > **Pipeline** from the navigation pane.

Step 2 In the **Operation** column of the **phoenix-workflow** pipeline, click ··· and choose **Edit**.

Step 3 Click the **Execution Plan** tab, select **Code commit** under **Trigger Events**, select the **master** branch, and click **Save**.

Figure 8-4 Configuring the execution plan



☑ Code commit

*Filter Branch ?

Include ▾ master ▾

+ Add

The modified execution plan is displayed.

Step 4 Modify the code and push it to the **master** branch. Then check whether the pipeline is automatically executed.

----End