CodeArts Pipeline

Best Practices

Issue 01

Date 2024-11-08

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

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 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.

Procedure

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

Step 1: Create a Microservice

Step 2: Create a Change-triggered Pipeline

Step 3: Create a Change

Step 4: Execute a Change-triggered Pipeline

Table 1-1 Procedure

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

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

Preparations

- **You have created a project**. The following uses a Scrum project name **Project01** 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 **Repo01** (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 1-2 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 **Repo01** created in **Preparations**.
Default Branch	Select **master**.
Language	Development language for the microservice. Select **Java**.
Description	(Optional) Enter a microservice description.

Step 8 Click **OK**.

----End

Step 2: 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 1-3 Pipeline parameters

Parameter	Description
Project	Keep the default value, which is the project of the pipeline.
Name	Use the default name.
Code 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	(Optional) If an alias is set, the system parameters are generated for the repository.
Change-based Trigger	Enable it to set current pipeline to a change-triggered one. It is enabled in this example.
Description	(Optional) Enter a pipeline description.

NOTE

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

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 3: Create a Change

Step 1 Access the created microservice.

Step 2 Click the **Changes** tab.

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

Table 1-4 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.

----End

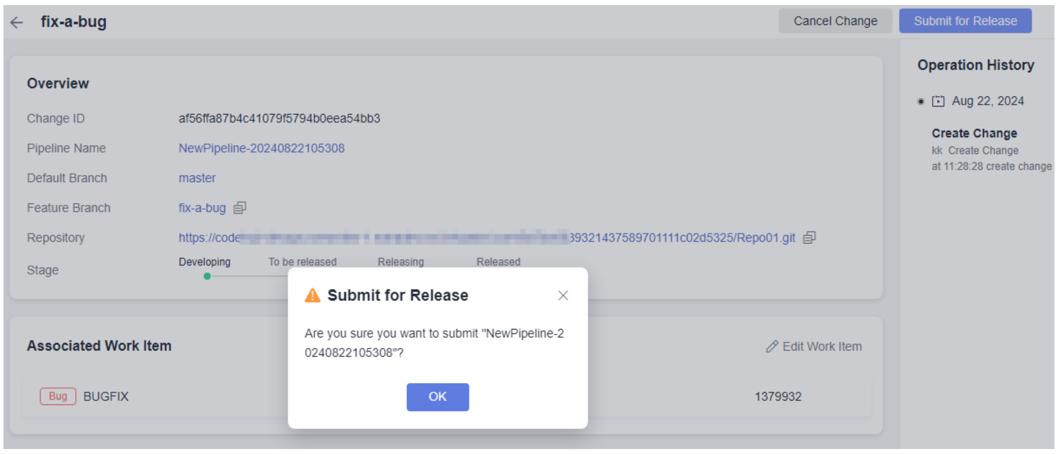
Step 4: Execute a Change-triggered Pipeline

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

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

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

Figure 1-1 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

Aug 22, 2024

Create Change
kk Create Change
at 11:28:28 create change

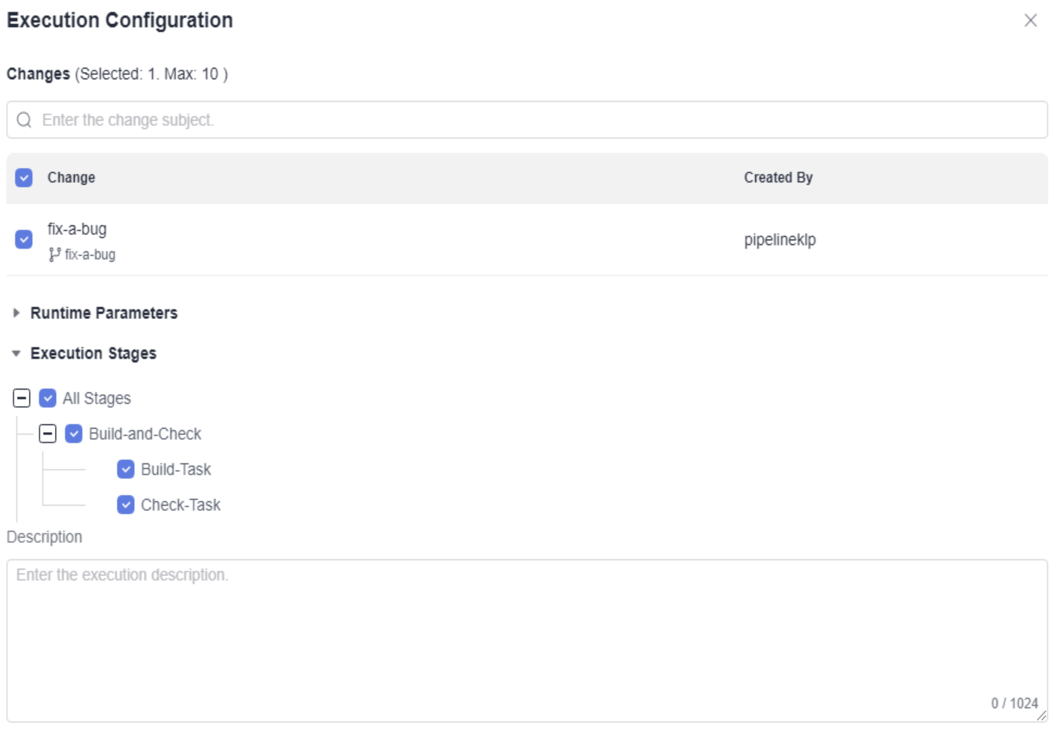
Submit for Release

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 dialog box, select the submitted change, and retain the default settings.

Figure 1-2 Execution configuration



Execution Configuration

Changes (Selected: 1. Max: 10)

Enter the change subject.

✓ Change	Created By
✓ fix-a-bug fix-a-bug	pipelineklp

Runtime Parameters

Execution Stages

- ✓ All Stages
  - ✓ Build-and-Check
    - ✓ Build-Task
    - ✓ Check-Task

Description

Enter the execution description.

0 / 1024

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.

----End

2 Configuring Pass Conditions for Automated Code Checks

Overview

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

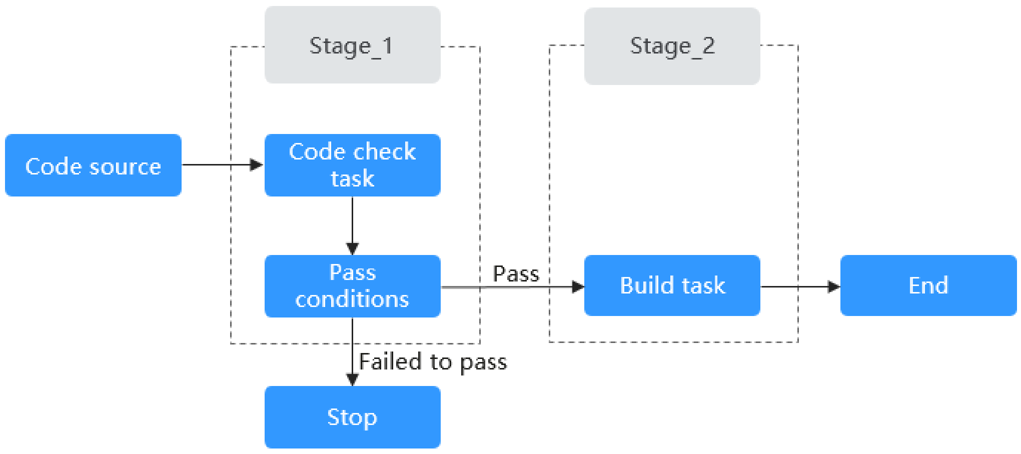
CodeArts Pipeline uses pass conditions to control whether a pipeline can proceed to the next stage. You can apply policies to pipelines as pass conditions for efficient project management and high-quality delivery.

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 2-1 Pipeline workflow



Perform the following procedure.

- **Step 1: Create a Rule and Configure Thresholds**
- **Step 2: Create a Policy and Add a Rule to the Policy**
- **Step 3: Configure a Pipeline**
- **Step 4: Execute the Pipeline**

Table 2-1 Procedure

Step	Description
Create a rule and configure thresholds	Create a rule of the code check type and configure thresholds for the rule.
Create a policy and add the rule to the policy	Add the preceding code check rule to the created policy.
Configure a pipeline	Add the preceding policy to the pass conditions.
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.

Preparations

- **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.

  A code check task with the same name as the code repository is automatically created. Change the task name to **CheckTask01** by referring to **Configuring Basic Info**.
- **You have created a build task** with the **Repo01** 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 **Pipeline01** (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.

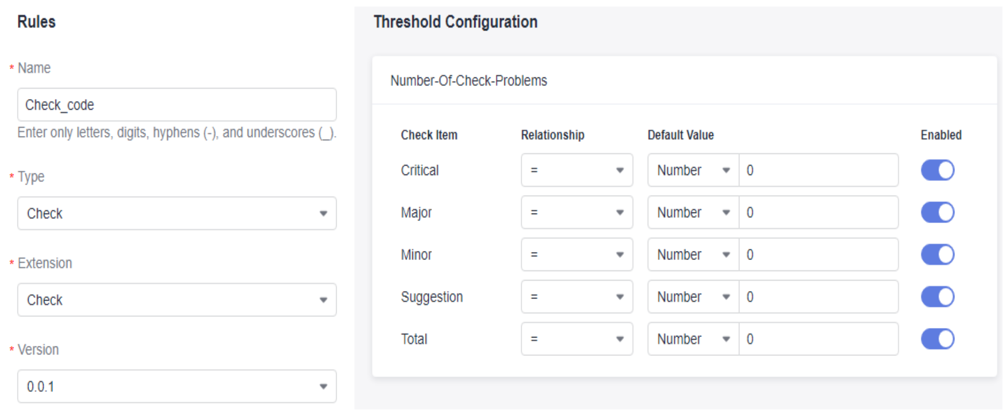
Step 3 Click **Access Service**.

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 2-2 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-Check-Problems

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

Table 2-2 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**.

----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.

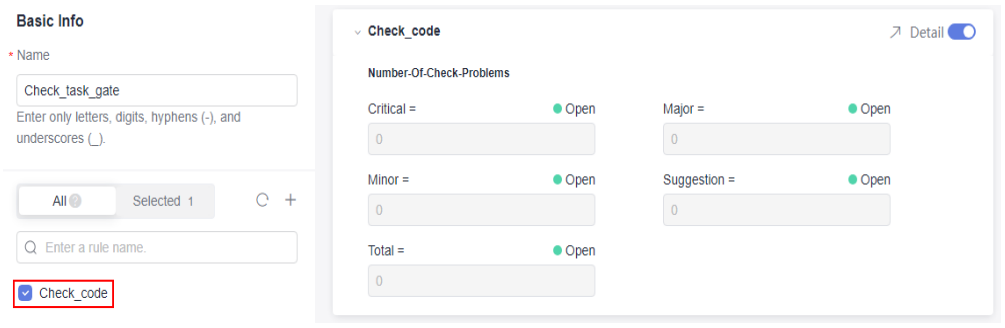
Step 1 In the navigation pane on the left, choose **Policies**.

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 2-3 Creating a policy



Basic Info
* Name
Check_task_gate
Enter only letters, digits, hyphens (-), and underscores (_).
All | Selected 1
Enter a rule name.
Check_code

Check_code | Detail
Number-Of-Check-Problems
Critical = Open 0
Major = Open 0
Minor = Open 0
Suggestion = Open 0
Total = Open 0

Step 3 Click **OK**.

----End

Step 3: Configure a Pipeline

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

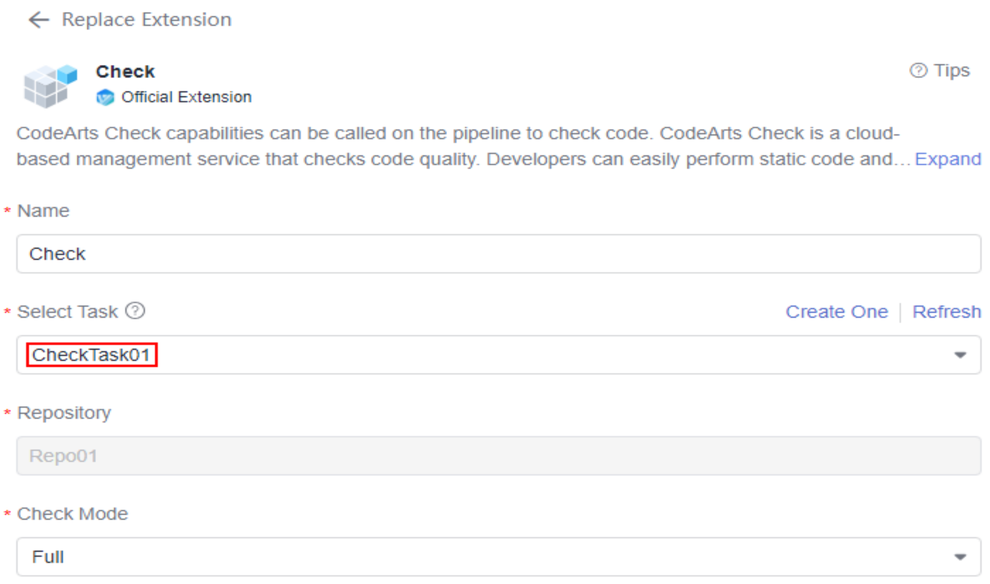
Step 2 Search for the project created in **Preparations** and access the project.

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**, add the code check job created in **Preparations**, and set **Check Mode** to **Full**.

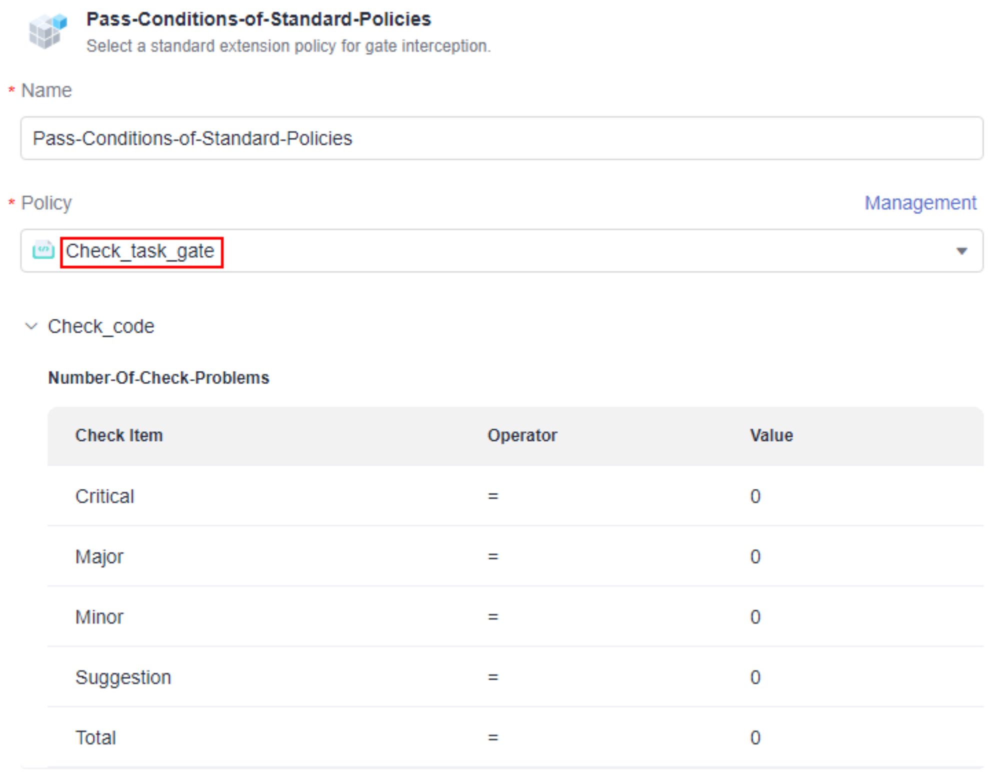
Figure 2-4 Adding a code check job



Replace Extension
Check
Official Extension
Tips
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... Expand
* Name
Check
* Select Task
Create One | Refresh
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 2-5 Adding pass conditions



Pass-Conditions-of-Standard-Policies
Select a standard extension policy for gate interception.

* Name
Pass-Conditions-of-Standard-Policies

* Policy
Check_task_gate

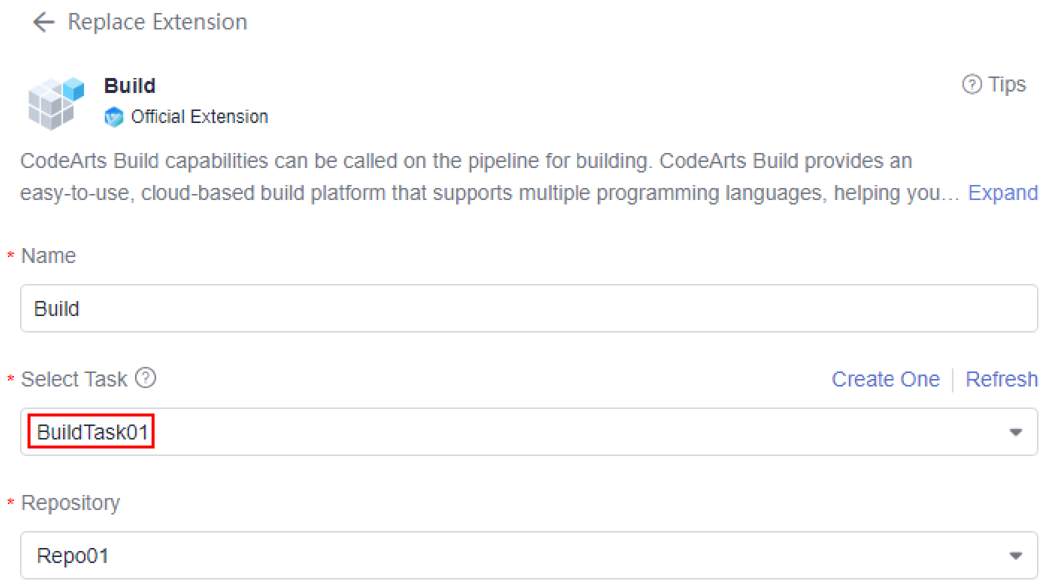
Check_code

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 job created in **Preparations**, and select the associated repository for the build job.

Figure 2-6 Adding a build job


← Replace Extension

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... Expand

* Name

Build

* Select Task

Create One | Refresh

BuildTask01

* Repository

Repo01

----End

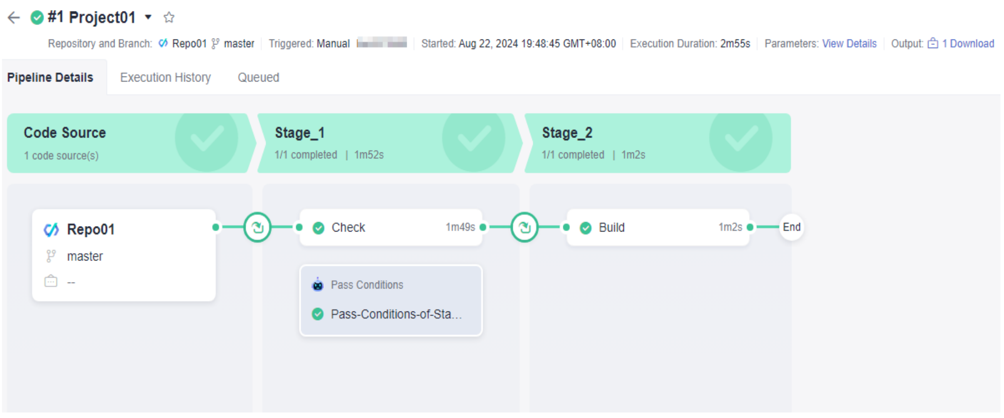
Step 4: Execute the Pipeline

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 2-7 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 — --

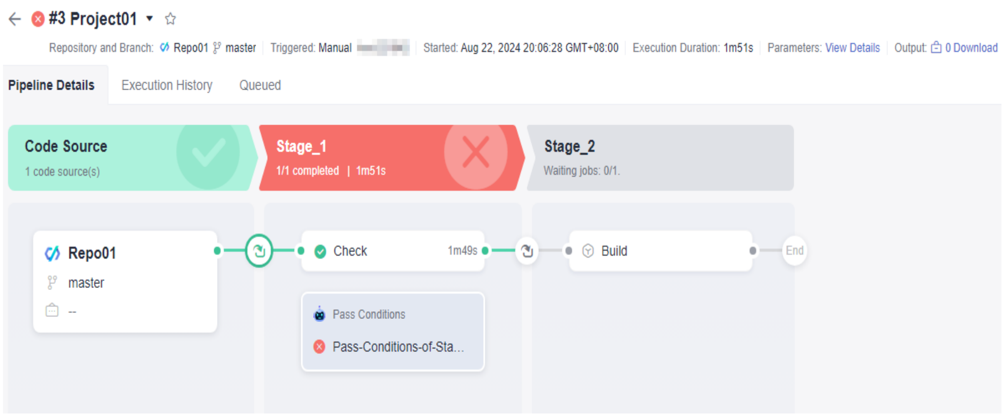
Check 1m49s

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

Build 1m2s

End

- If the code check job 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.



#3 Project01

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

----End

3 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.

Step 1: Create a Build Task

Step 2: Create an application

Step 3: Create and Execute a Pipeline

Step 4: Check Build and Deployment Results

Table 3-1 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.

Step	Description
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 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.
- 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 3-2 Basic information

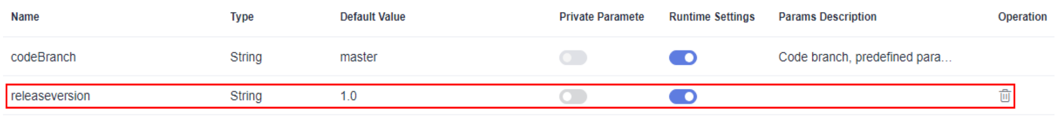
Parameter	Description
Name	Build task name. Enter **BuildTask01**.
Project	Keep the default value, which is the project of the build task.
Code Source	Code source associated with the build task. Select **Repo**.

Parameter	Description
Repository	Select the repository **Repo01** 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, set the default value, and enable **Runtime Settings**.

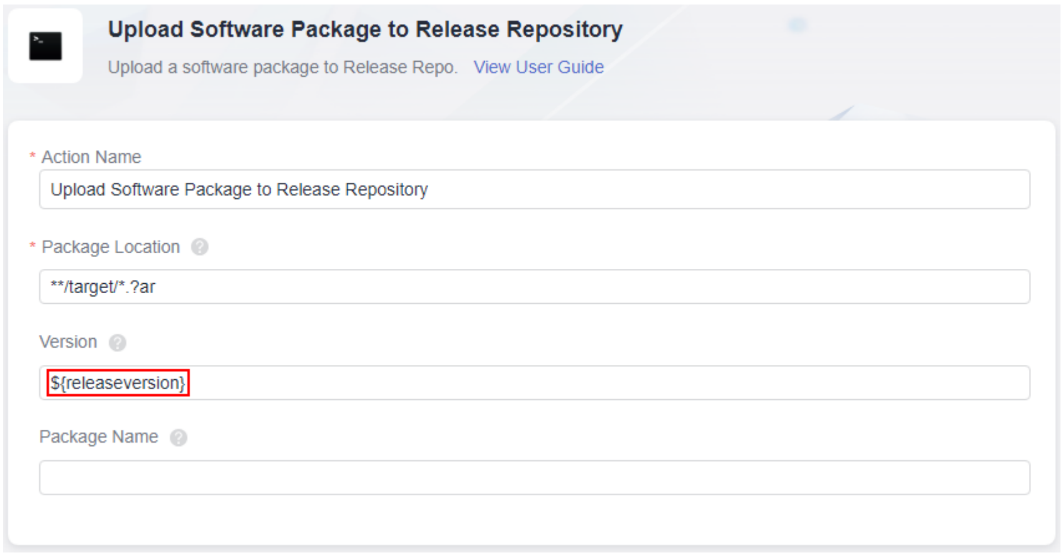
Figure 3-1 Creating a build parameter



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 3-2 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

Step 10 Click **Save**.

----End

Step 2: Create an application

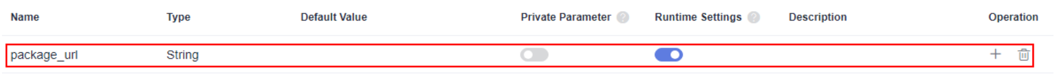
Step 1 In the left navigation pane, choose **Settings** > **General** > **Basic Resources**, create a host cluster, and add the purchased host to the cluster.

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

Figure 3-3 Creating a deployment parameter

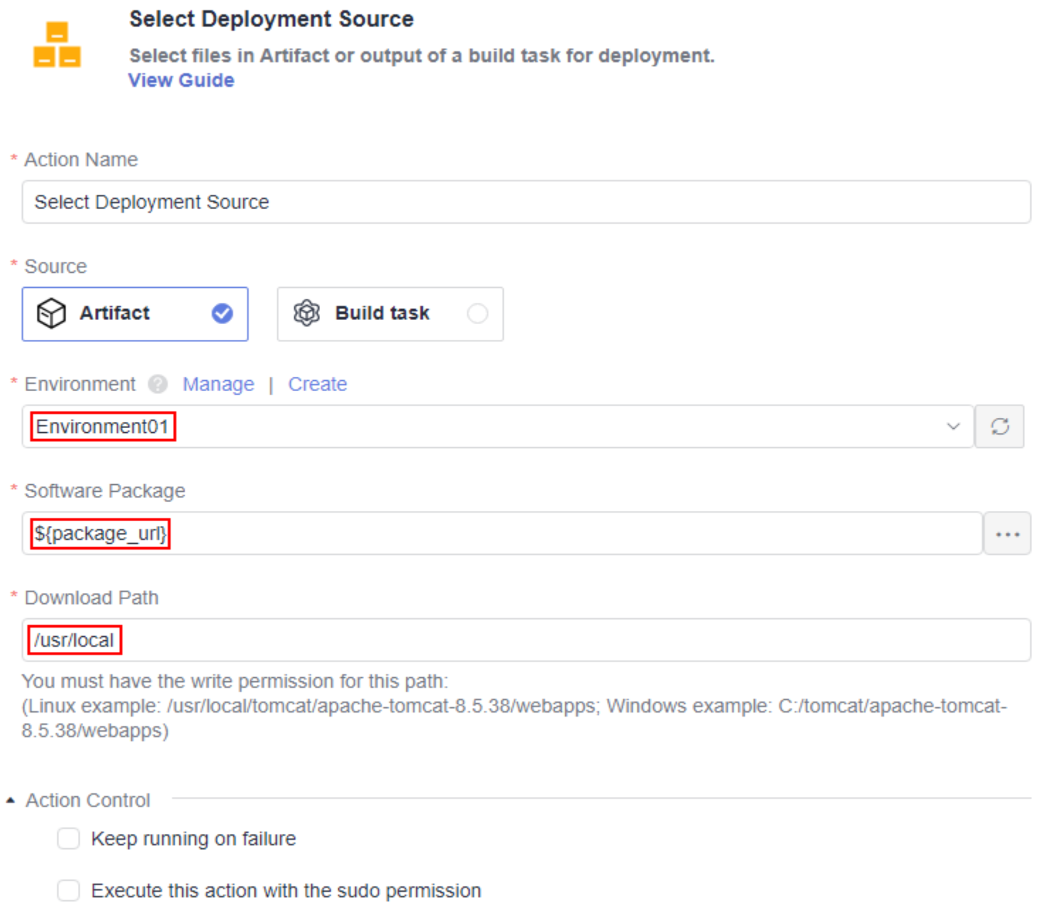


Name	Type	Default Value	Private Parameter	Runtime Settings	Description	Operation
package_url	String					

Step 5 On the **Environment Management** page, click **Create Environment**, enter the name **Environment01**, and import the host in the cluster to the environment.

Step 6 On the **Deployment Actions** tab page, add the **Select Deployment Source** action and configure the information as shown in the following table.

Figure 3-4 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 3-3 Configuring deployment actions

Parameter	Description
Action Name	Retain the default value.
Source	Software package source. Select **Artifact**.
Environment	Environment for deployment. Select **Environment01**.

Parameter	Description
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 pipeline information.

1. Configure the following basic information and click **Next**.

Table 3-4 Pipeline basic information

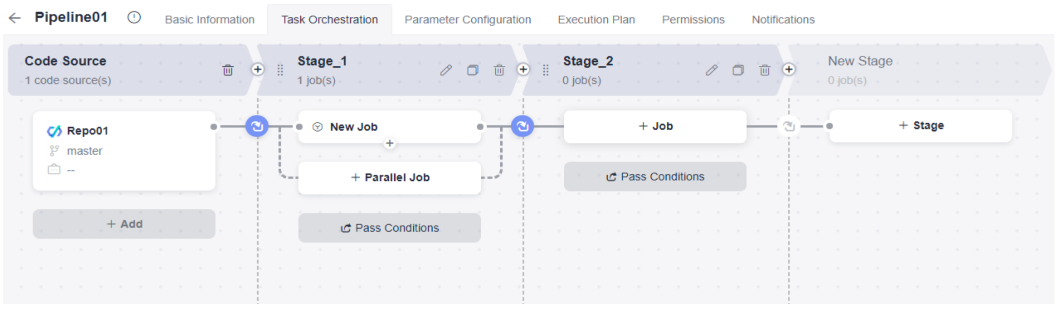
Parameter	Description
Name	Enter **Pipeline01**.
Code Source	Code source associated with the pipeline. Select **Repo**.
Repository	Select the repository **Repo01** created in **Preparations**.
Default Branch	Select **master**.

2. Select **Blank Template** and click **OK**.

Step 3 On the **Parameter Configuration** tab page, create the **releaseversion** parameter, set its default value to **${TIMESTAMP}**, and enable **Runtime Setting**.

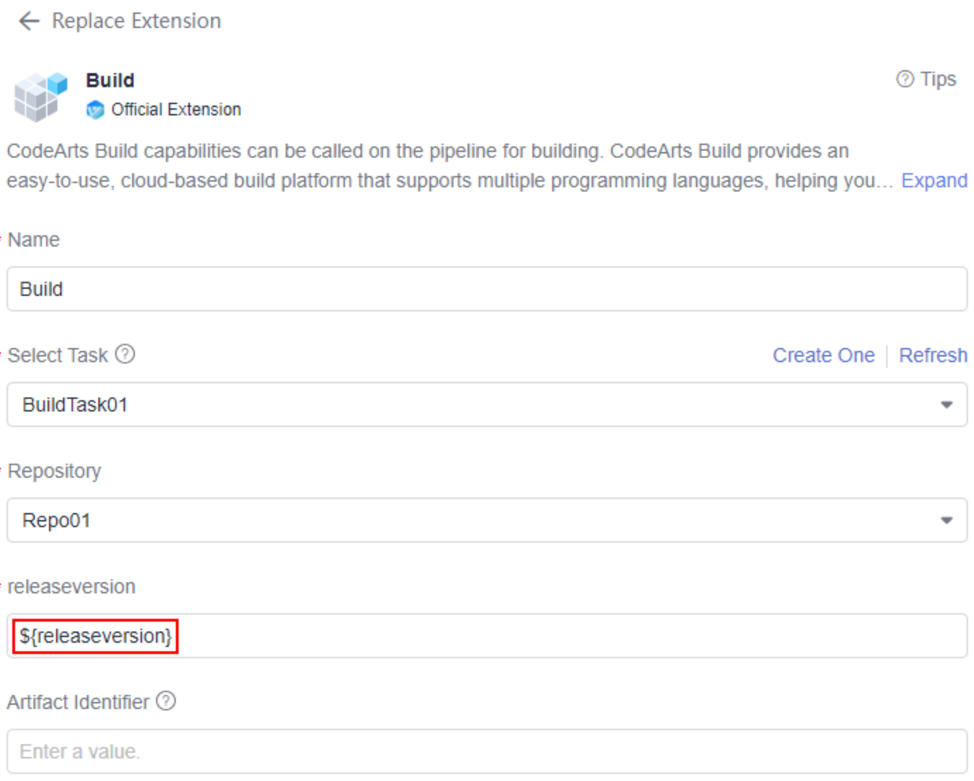
Step 4 On the **Task Orchestration** page, two stages (**Code Source** and **Stage_1**) are generated by default. Click **Stage** to add a new stage (**Stage_2**).

Figure 3-5 Task orchestration



1. Add a build job
   a. Click **New Job** under the **Stage_1**.
   b. Click the **Build** type and search for the **Build** extension.
   c. Move the cursor to the **Build** extension, click **Add**, select the **created build task**, select the repository associated with build task, and set **releaseversion** to **${releaseversion}** to reference the **releaseversion** parameter of the pipeline.

Figure 3-6 Adding a build job



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…

* Name: Build

* Select Task: BuildTask01

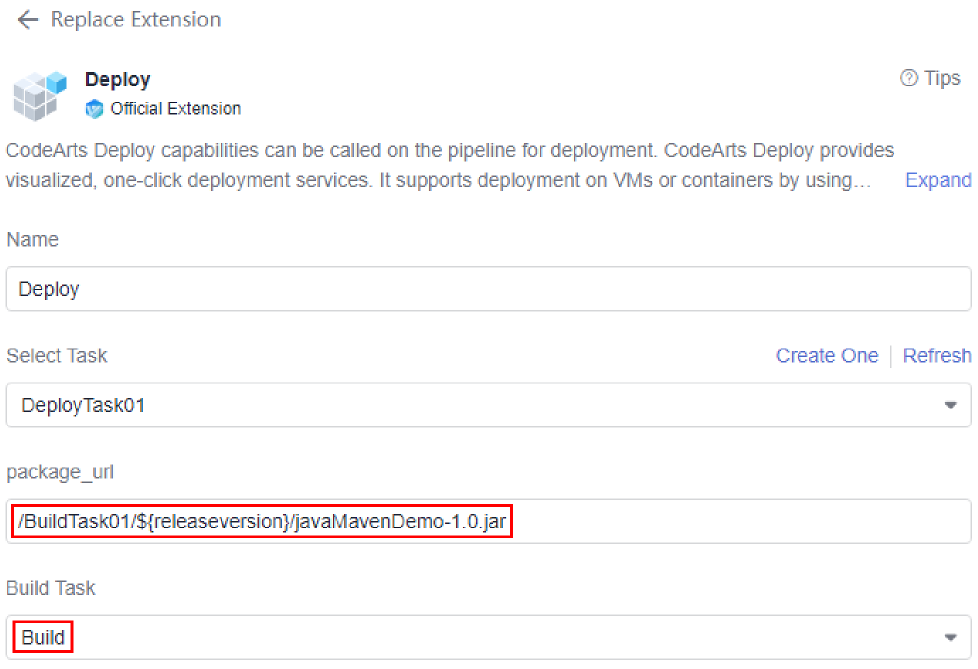
* Repository: Repo01

* releaseversion: ${releaseversion}

Artifact Identifier: Enter a value.

2. Add an application
   a. Click **Job** under **Stage_2**.
   b. In the displayed dialog box, search for the **Deploy** extension.
   c. Move the cursor to the **Deploy** extension, click **Add**, select the **created application**, enter the package path for **package_url**, and associate with the added build task, as shown in the following figure.

Figure 3-7 Adding an application


← Replace Extension

Deploy
Official Extension

Tips

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... Expand

* Name

Deploy

* Select Task — Create One | Refresh

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 5 Click **Save and Execute**.

----End

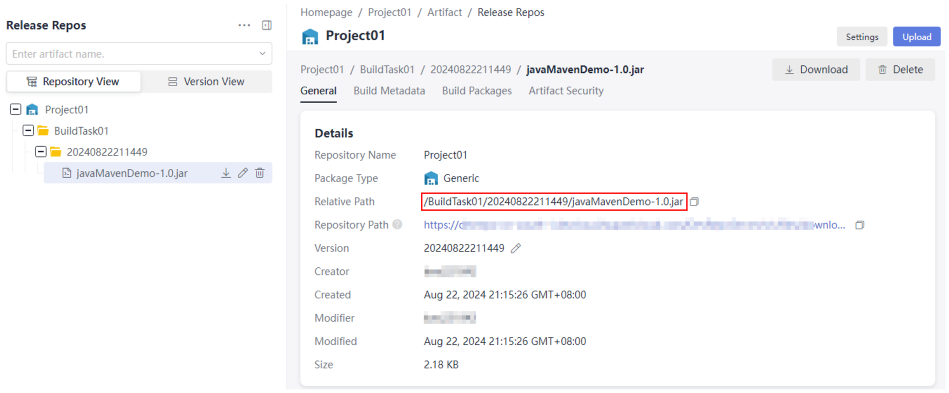
Step 4: Check Build and Deployment Results

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 3-8 Checking the software package


Release Repos

Enter artifact name.

Repository View | Version View

- Project01
  - BuildTask01
    - 20240822211449
      - javaMavenDemo-1.0.jar

Homepage / Project01 / Artifact / Release Repos

Project01

Settings | Upload

Project01 / BuildTask01 / 20240822211449 / **javaMavenDemo-1.0.jar**

Download | Delete

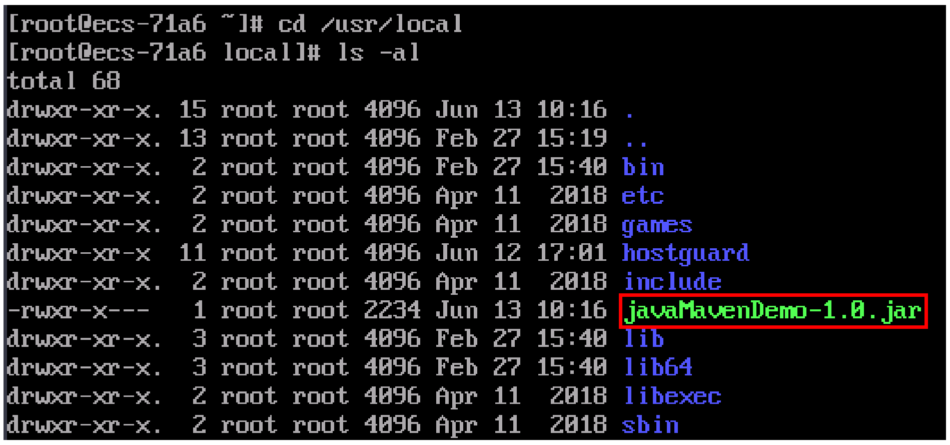
General | Build Metadata | Build Packages | Artifact Security

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 user name in the upper right corner.
  b. Click **CodeArts Console**.
  c. Click ☰ in the upper left corner and search for **Elastic Cloud Server**. Then, access the **Elastic Cloud Server** console.
  d. Locate the ECS used for deployment, click **Remote Login** in the **Operation** column.
  e. In the **Other Login Modes** area, select **Log in using Remote Login on the management console** and click **Log In**.
  f. Enter the username and password for purchasing the ECS. Press **Enter**.
  g. Enter the following command and press **Enter** to go to the directory **/usr/local** configured during the **Create an Application** step.
     ```
     cd /usr/local
     ```
  h. 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 3-9 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
```

4 Creating a Repository Tag Using the Pipeline Contexts

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 created a project**. The following uses a Scrum project named **Project01** as an example.
- **You have created a code repository** and created a branch. The following uses a repository named **Repo01** (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 pipeline information.

1. Configure the following basic information and click **Next**.

Table 4-1 Pipeline basic information

Parameter	Description
Name	Enter **Pipeline01**.

Parameter	Description
Project	Project to which the pipeline belongs. Select the project **Project01** created in **Preparations**.
Code Source	Code source associated with the pipeline. Select **Repo**.
Repository	Select the repository **Repo01** created in **Preparations**.
Default Branch	Select the branch **release-1.0.0** created in **Preparations**.

2. Select **Blank Template** and click **OK**.

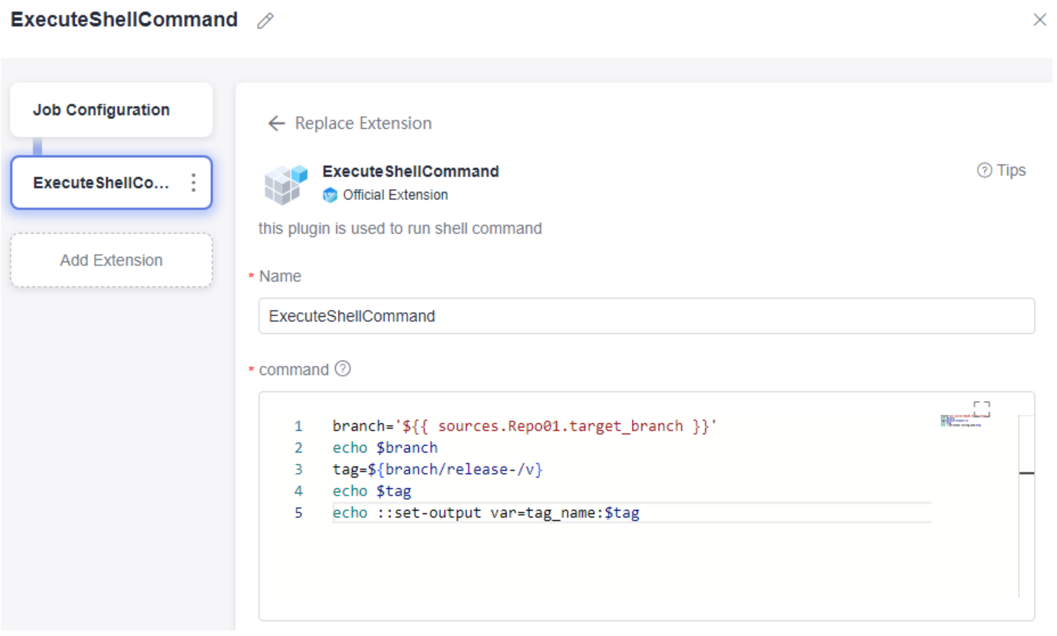
Step 5 Go to the **Task Orchestration** page. Two stages (**Code Source** and **Stage_1**) are generated by default. Click **Stage** to add a new stage (**Stage_2**).

Step 6 Add the **ExecuteShellCommand** to generate a tag name.

1. Click **Job** under the **Stage_1**.
2. Search for the extension **ExecuteShellCommand** and add it.
3. Enter a name (here we retain the default name) and enter the following shell commands:

```
branch='${{ sources.Repo01.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 4-1 Generating a tag name


ExecuteShellCommand

this plugin is used to run shell command

* Name: ExecuteShellCommand

* command

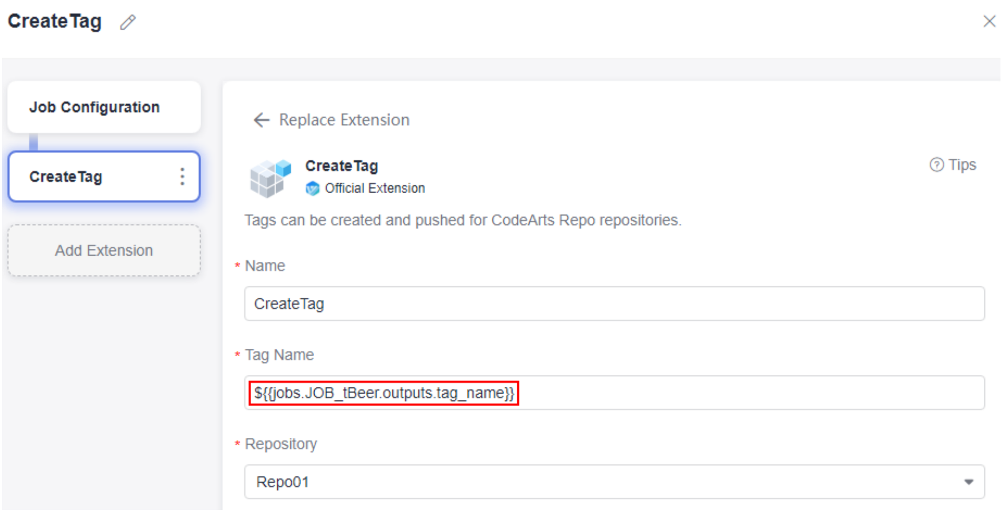
```
branch='${{ sources.Repo01.target_branch }}'
echo $branch
tag=${branch/release-/v}
echo $tag
echo ::set-output var=tag_name:$tag
```

Step 7 Add the **CreateTag** extension to create a repository tag.

1. Click **Job** under **Stage_2**.

2. Search for the **CreateTag** extension and add it, and set the following information:
   – **Name**: Extension name. Retain the default value.
   – **Tag Name**: Enter **${{jobs.JOB_tBeer.outputs.tag_name}}**, where **JOB_tBeer** indicates the ID of the **ExecuteShellCommand** job.
   – **Repository**: Select the code repository associated with the pipeline.

Figure 4-2 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 8 After the configuration, click **Save and Execute**.

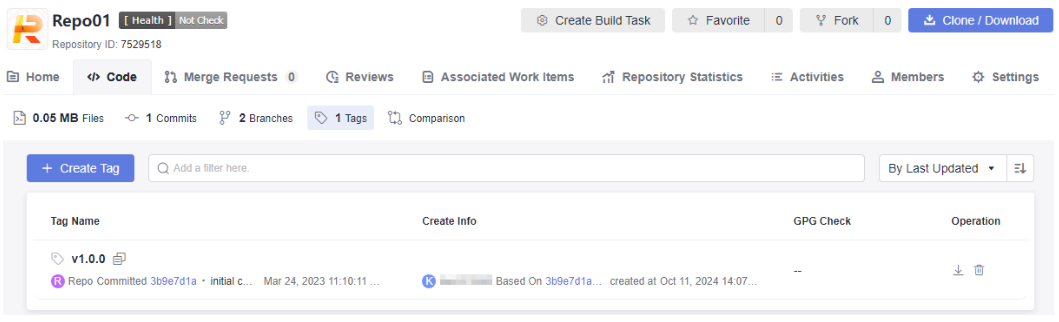
Step 9 In the displayed dialog box, retain the default settings, and click **Execute**.

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 4-3 Checking a tag



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

5 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 5-1 Preset pipeline tasks

Preset Pipeline Task	Description
phoenix-workflow	A basic pipeline task
phoenix-workflow-test	Pipeline task corresponding to the test environment
phoenix-workflow-work	Pipeline task corresponding to the Worker function
phoenix-workflow-result	Pipeline task corresponding to the Result function
phoenix-workflow-vote	Pipeline task corresponding to the Vote function

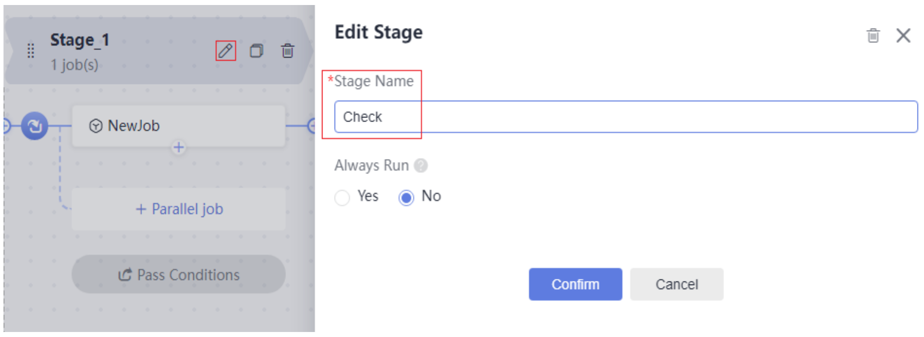
Configuring and Executing a Pipeline

A pipeline usually consists of multiple stages. You can add multiple jobs to each stage.

Step 1 Configure a pipeline.

1. Go to the **Phoenix Mall** project and choose **CICD** > **Pipeline**.
2. Find pipeline **phoenix-workflow**. Click ··· and choose **Edit**.
3. Add a code check stage.
   a. Click ⊕ between **Code Source** and **Build** to add a stage.
   b. Click ✎ next to **Stage_1**. In the **Edit Stage** window, enter the stage name **Check** and click **Confirm**.

   Figure 5-1 Editing the stage name

   *Stage Name: Check
   Always Run: Yes [✓ No]

   c. Click **Job**.
   In the **New Job** window, click **Add** next to the **Check** extension.
   d. Select the **phoenix-codecheck-worker** task and click **OK**.

   NOTE

   The check task has three modes. This procedure uses the default mode **Full**. You can change the mode as required.

   - **Full**: All files in the code repository are scanned.
   - **Incremental (last commit)**: Incremental check is performed based on the latest commit file.
   - **Incremental (last success)**: Incremental check is performed based on the changed files since the latest access control was passed.
4. 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.
5. Click **Save**.

Step 2 Go to the CCE console if you have configured deployment task **phoenix-cd-cce** in Step 1. Locate the target cluster and click its name to go to the **Overview** page.

Choose **Workloads** in the navigation pane, click the **Deployments** tab, and verify that no record exists in the list.

If there are records in the list, select all records, click **Delete**, select all options, and click **Yes** to clear the records in the list.

Step 3 Return to the pipeline list page. Click ▷ in the row where **phoenix-workflow** is located, and click **Execute** in the window that is displayed to start the pipeline.

If ✓ is displayed on the page, the task is successfully executed.

If the task fails to be executed, check the failure cause in the failed task. You can open the step details page to view the task logs and rectify the faults based on the logs.

----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 On the details page of pipeline task **phoenix-workflow**, click **Edit**.

Step 2 In the **Check** stage, click **Pass Conditions**.

Step 3 In the **Pass Conditions** dialog box, click **Add** next to **Pass-Conditions-of-Standard-Policies**.

Step 4 Select **SystemPolicy** and click **OK**.

Step 5 Click **Save and Execute**.

If the number of check issues does not meet the pass condition, the pipeline task fails to be executed.

----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 On the details page of pipeline task **phoenix-workflow**, click **Edit**.

Step 2 Click the **Execution Plan** tab, select **Code commit**, select **master** from the **Filter Branch** drop-down list box, and click **Save**.

Step 3 Modify the code and push it to the **master** branch to check whether the pipeline task is automatically executed.

----End