CodeArts Pipeline

Best Practices

Issue 01

Date 2024-06-11

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 On-demand Feature Release in Microservices....................1
2 Configuring Pass Conditions for Pipelines....................6
3 HE2E DevOps Practice: Configuring a Pipeline.................... 11

1 On-demand Feature Release in Microservices

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

Preparations

- **You have created a project**. The following uses a Scrum project named **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** in the project. The following uses a repository named **test01** (created using the Maven template) as an example.
- **You have created a CodeArts Repo HTTPS service endpoint**. The following uses an endpoint named **https_endpoint** as an example.

Step 1: Create a Microservice

Step 1 **Log in to CodeArts**.

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 Click the **Microservices** tab.

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

Table 1-1 Microservice parameters

Parameter	Description
Project	Keep the default value, which is the project of the microservice.
Microservice Name	Enter **microservice01**.
Pipeline Source	Select the pipeline source. Only **Repo** is supported.
Repository	Select the repository **test01** created in **Preparations**.
Default Branch	Select **master**.
Language	Select **Java**.
Description	(Optional) Enter a microservice description.

Step 6 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 1-2 Pipeline parameters

Parameter	Description
Project	Keep the default value, which is the project of the pipeline.
Name	Use the default name.
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.

Parameter	Description
Repo HTTPS Authorization	This is mandatory if you enabled **Change-based Trigger**. Select the authorization endpoint **https_endpoint** 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) Describe the pipeline.

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 **Get-Started** template. The code check, build, and deployment stages are generated. You can orchestrate the pipeline as needed.

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 1-3 Change parameters

Parameter	Description
Change Subject	Enter **bugfix**.
Repository	Keep the default value, which is the same as that of the microservice.
Branch	Select **Pull new from default** and enter a branch name, or select **Associate with existing** to select an existing branch. Here we pull a new branch from the default branch, and enter the 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 the newly pulled branch.

----End

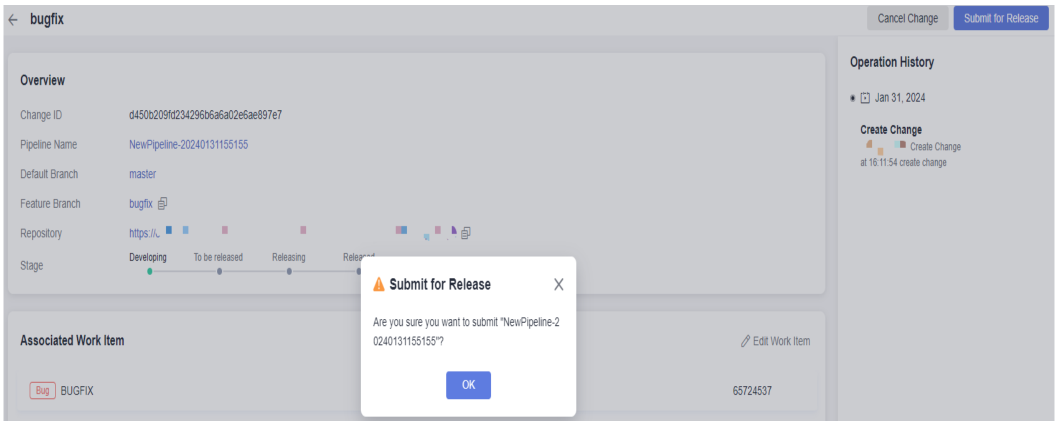
Step 4: Run a Change-triggered Pipeline

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



bugfix

Overview

Change ID	d450b209fd234296b6a6a02e6ae897e7
Pipeline Name	NewPipeline-20240131155155
Default Branch	master
Feature Branch	bugfix
Repository	https://...
Stage	Developing · To be released · Releasing · Released

Associated Work Item

Bug BUGFIX 65724537

Operation History

Jan 31, 2024

Create Change

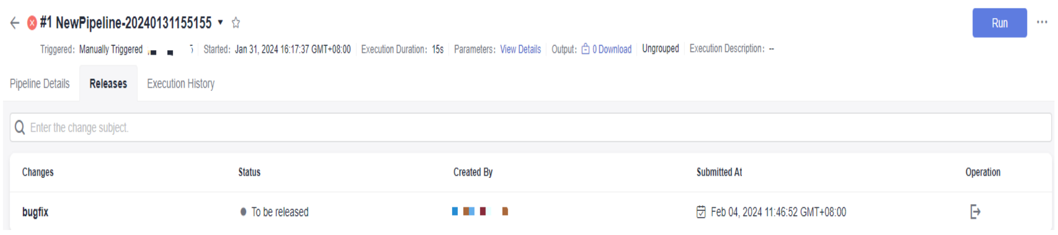
Create Change at 16:11:54 create change

Submit for Release

Are you sure you want to submit "NewPipeline-20240131155155"?

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

Figure 1-2 Releases



#1 NewPipeline-20240131155155

Triggered: Manually Triggered | Started: Jan 31, 2024 16:17:37 GMT+08:00 | Execution Duration: 15s | Parameters: View Details | Output: 0 Download | Ungrouped | Execution Description: --

Changes	Status	Created By	Submitted At	Operation
bugfix	● To be released		Feb 04, 2024 11:46:52 GMT+08:00	

Step 4 Click **Run** in the upper right corner. In the displayed dialog box, select the submitted change, configure runtime parameters, execution stages, and jobs as needed.

Step 5 Click **Run**.

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, build, and deployment 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 Pipelines

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.

Traditional automotive enterprises mainly rely on manual tests, leading to low efficiency.

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

Procedure

You can add pass conditions for a stage where there is a code check task. If the code check result meets the pass conditions, the pipeline continues to run. Perform the following procedure.

- **Step1: 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**

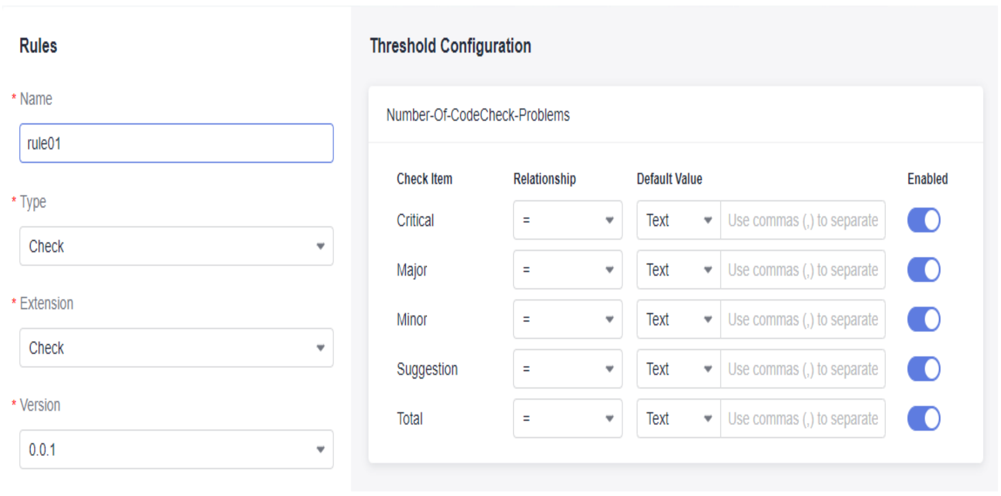
Preparations

- **You have created a project**. The following uses a Scrum project named **Project01** as an example.
- **You have created a code repository** in the project. The following uses a repository named **repo01** (created using the Maven template) as an example. (You can choose to automatically create a code check task.)
- **You have created a code check task** with the **repo01** repository. The following uses a code check task named **check01** as an example.
- **You have created a build task** with the **repo01** repository. The following uses a build task named **build01** (created using the Maven template) 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.

Step1: Create a Rule and Configure Thresholds

1. **Log in to CodeArts**.
2. Click the avatar icon in the upper right corner and choose **All Account Settings** from the drop-down list.
3. In the navigation pane on the left, choose **Policy Management** > **Rules**.
4. Click **Create Rule**. On the displayed page, configure parameters.

Figure 2-1 Creating a rule



Rules

* Name: rule01

* Type: Check

* Extension: Check

* Version: 0.0.1

Threshold Configuration

Number-Of-CodeCheck-Problems

Check Item	Relationship	Default Value		Enabled
Critical	=	Text	Use commas (,) to separate	✓
Major	=	Text	Use commas (,) to separate	✓
Minor	=	Text	Use commas (,) to separate	✓
Suggestion	=	Text	Use commas (,) to separate	✓
Total	=	Text	Use commas (,) to separate	✓

Table 2-1 Rule parameters

Parameter	Description
Name	Enter a rule name, for example **rule01**.
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.

5. Click **Confirm**.

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.

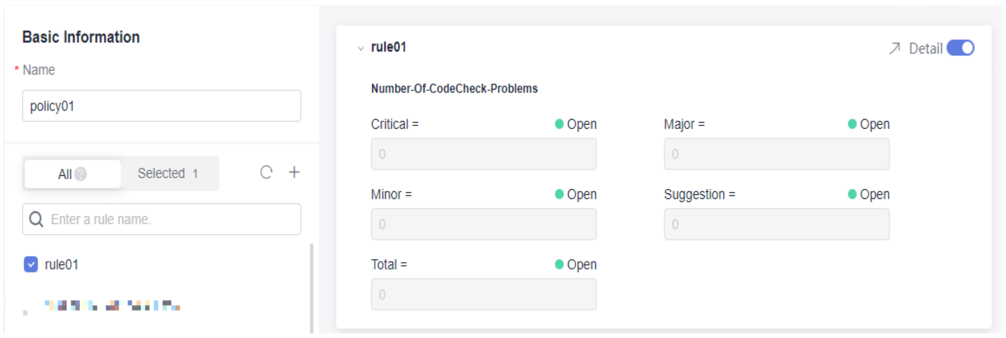
1. In the navigation pane on the left, choose **Policies**.

NOTE

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

2. Click **Create Policy**. On the displayed page, enter a policy name **policy01** and select the rule **rule01** created in **step 1**.

Figure 2-2 Creating a policy



Basic Information

* Name

policy01

All | Selected 1

Enter a rule name.

☑ rule01

rule01 | Detail

Number-Of-CodeCheck-Problems

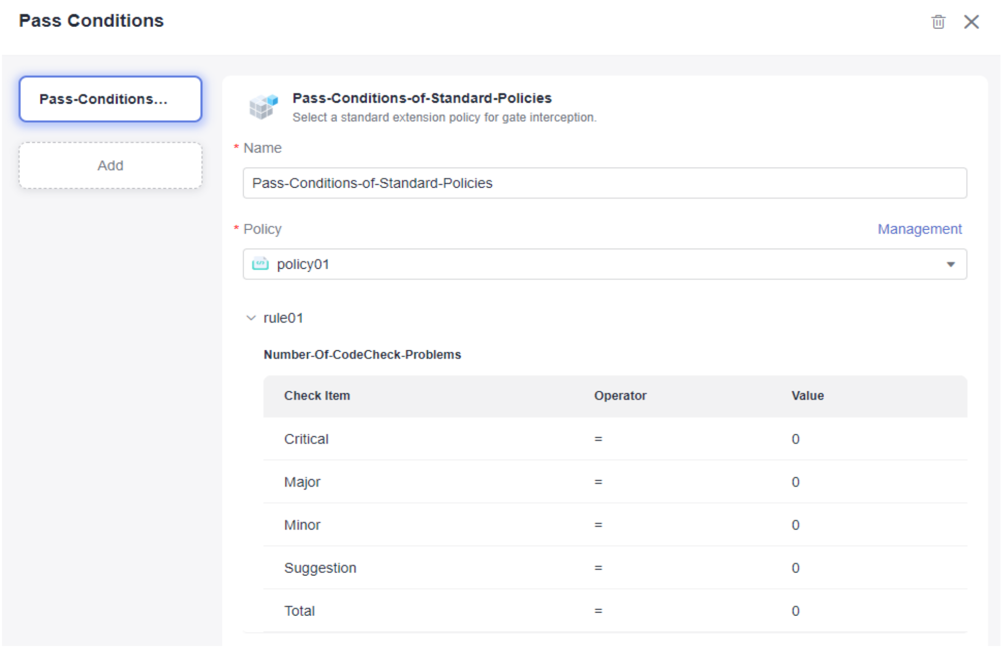
Critical =	● Open	Major =	● Open
0		0	
Minor =	● Open	Suggestion =	● Open
0		0	
Total =	● Open		
0			

3. Click **Confirm**.

Step 3: Configure a Pipeline

1. On the top navigation bar, click **Homepage**.
2. Search for the project created in **Preparations** and access the project.
3. In the navigation pane on the left, choose **CICD** > **Pipeline**.
4. Search for the pipeline created in **Preparations**, click ··· in the **Operation** column, and select **Edit**. The **Task Orchestration** page is displayed.
5. In stage 1, add the code check task created in **Preparations**, click Pass Conditions, add **Pass-Conditions-of-Standard-Policies**, and select the policy **policy 01** created in **step 2**.

Figure 2-3 Adding pass conditions


Pass Conditions

Pass-Conditions...

Add

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

* Name
Pass-Conditions-of-Standard-Policies

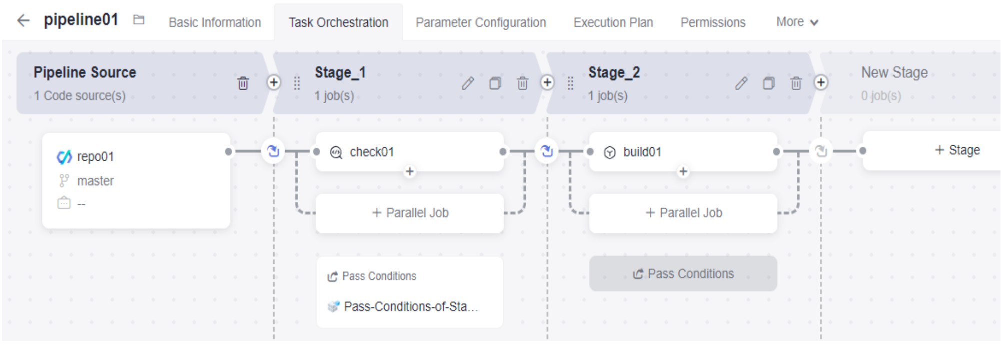
* Policy
Management
policy01

rule01
Number-Of-CodeCheck-Problems

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

6. Click ⊕ or + Stage to add a new stage for the pipeline and add the build task created in **Preparations** to the new stage.

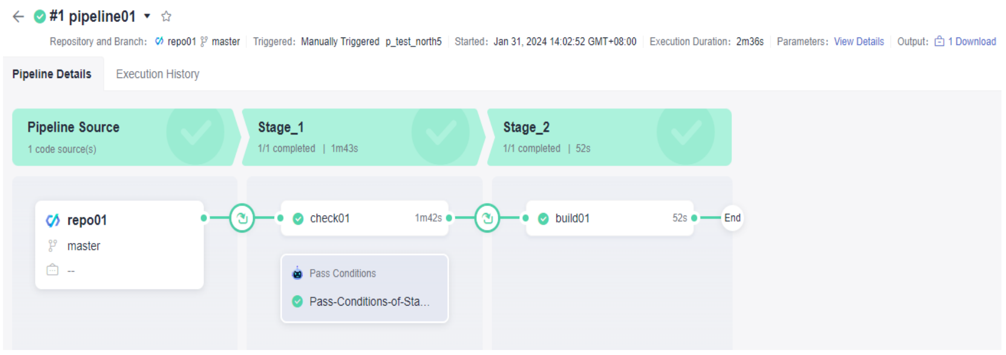
Figure 2-4 Configuring a pipeline



Step 4: Execute the Pipeline

1. After configuring the pipeline, click **Save and Run**.
2. After execution, the code check task passes the verification and the pipeline proceeds to the next stage.

Figure 2-5 Executing a pipeline



#1 pipeline01

Repository and Branch: repo01 master | Triggered: Manually Triggered p_test_north5 | Started: Jan 31, 2024 14:02:52 GMT+08:00 | Execution Duration: 2m36s | Parameters: View Details | Output: 1 Download

3 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**.

Preset Pipelines

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

Table 3-1 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

Step 1 Configure a pipeline.

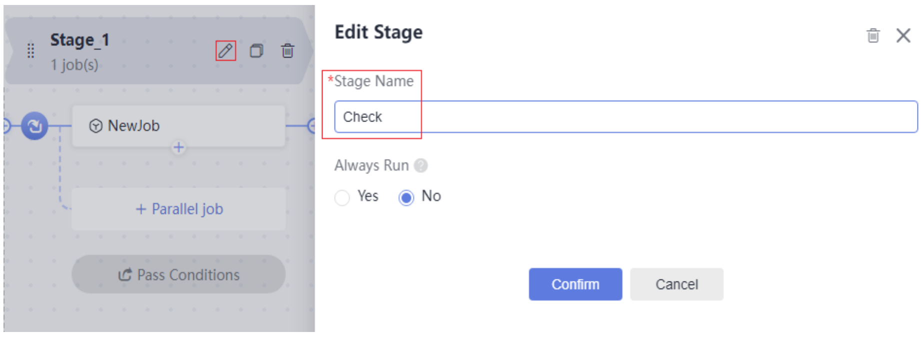
1. Go to the **Phoenix Mall** project and choose **CICD** > **Pipeline**.
2. Find pipeline **phoenix-workflow**. Click ··· and click **Edit**.

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 3-1 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.

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 , 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 **Run** 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

Step 1 On the **phoenix-workflow** details page, click ··· in the upper right corner and choose **Edit** from the drop-down list.

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 **phoenix-workflow** details page, click **Edit** in the upper right corner.

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