CodeArts Build

Best Practices

Issue	01
Date	2023-11-15

HUAWEI TECHNOLOGIES CO., LTD.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Creating a Docker Image with a Maven Artifact and Pushing the Image to SWR (Built-in Executors, GUI)

Scenario

In this practice, you use CodeArts Build to build a Maven project. Afterward, you create a Dockerfile that outlines the process of packaging the Maven build outputs into a container image. Then, you build a Docker image from the Dockerfile and push the image to the image repository in SoftWare Repository for Container (SWR). The build output contained within the image can be used for future compilation or deployment purposes.

Requirements

- You have **created an organization** named **codeci_gray** in SWR.
- You have permissions for CodeArts Repo.

Procedure

Table 1-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a CodeArts Repo Repository	Create the code file used for your build.
Creating a Build Task	Create, configure, and run a build task.
Viewing the Build Results	Check the build logs and result files to verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

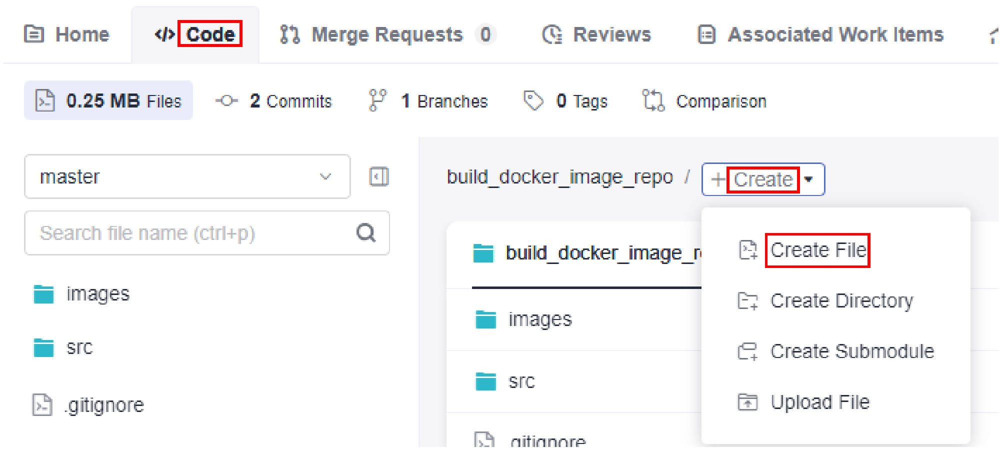
Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the displayed page, set **Repository Name** to **build_docker_image_repo**, and leave the other parameters as default. Click **OK**. The details page of the new code repository is displayed.

Step 5 In the root directory of the code repository, choose **Create** and select **Create File** from the drop-down list.

Figure 1-1 Creating a file



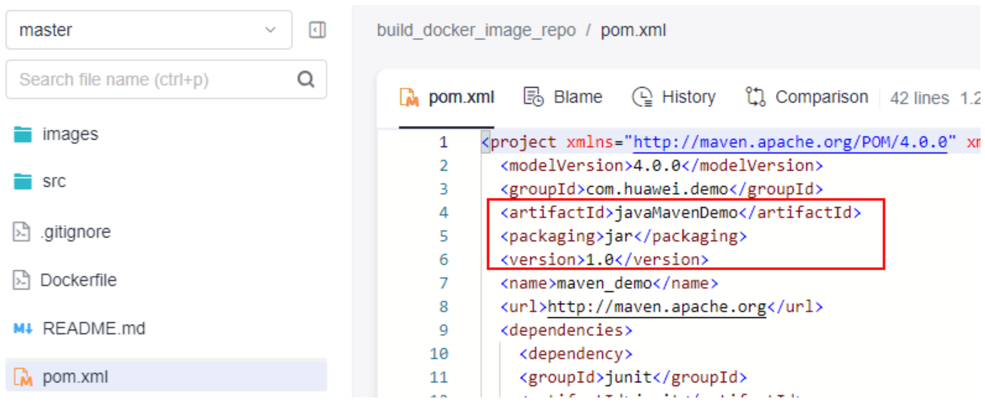
Step 6 Name the file **Dockerfile**, copy the following code to the file (as shown in Figure 1-3), and click **Submit**.

```
FROM ubuntu:latest

# set maintainer
LABEL maintainer=build
```

```
RUN mkdir /release_app
COPY ./target/javaMavenDemo-1.0.jar /release_app/maven_app.jar

USER build
```

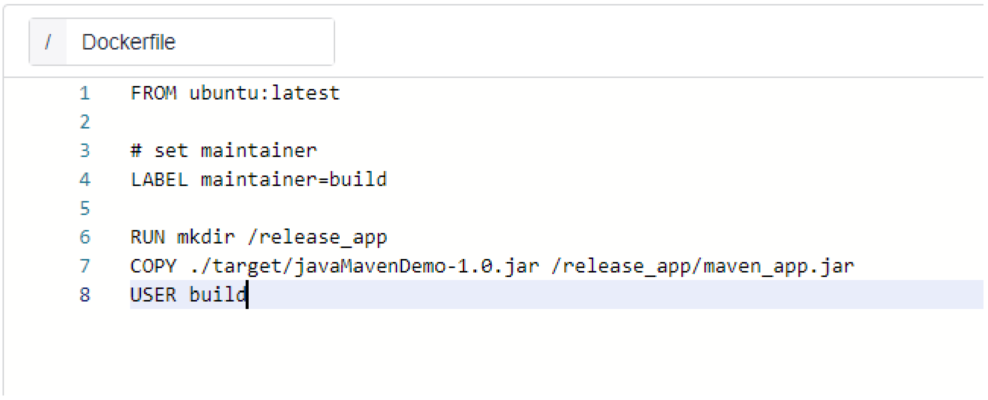
javaMavenDemo-1.0.jar is the file name in the format of **${artifactId}-${version}.${packaging}** defined in the **pom.xml** file, as shown in **Figure 1-2**.

Figure 1-2 pom.xml file



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xr
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>maven_demo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
```

Figure 1-3 Dockerfile content



```
FROM ubuntu:latest

# set maintainer
LABEL maintainer=build

RUN mkdir /release_app
COPY ./target/javaMavenDemo-1.0.jar /release_app/maven_app.jar
USER build
```

----End

Creating a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 1-2**. Then, click **Next**.

Table 1-2 Basic information

Parameter	Description
Name	Assign a custom name to the build task, for example, **build_docker_image_task**.

Parameter	Description
Code Source	Select the code source from which code will be pulled for your build. In this practice, select **Repo**.
Repository	Select **build_docker_image_repo**, the code repository created in **Creating a CodeArts Repo Repository**.
Default Branch	Keep the default value **master**.

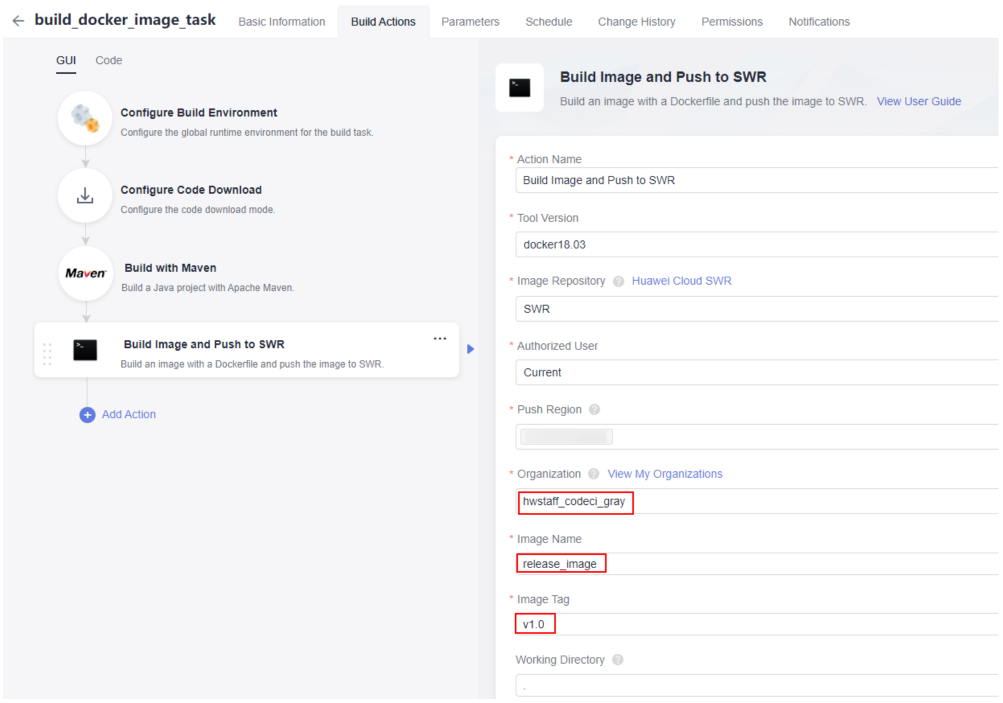
Step 3 Select **Blank Template** and click **OK** to create the build task. The **Build Actions** page is displayed.

Step 4 On the **Build Actions** page, click the **GUI** tab. Then click **Add Build Actions** on the left and add the **Build with Maven** action. Leave the parameters as default.

Step 5 Click **Add Action**. In the right pane, click the **Container related** tab. Hover over action **Build Image and Push to SWR** and click **Add** on the card. Set the parameters according to **Figure 1-4**.

Expand the **Organization** drop-down list and select **hwstaff_codeci_gray** created to meet the **requirements**. Type **release_image** in the **Image Name** text box, and **v1.0** in **Image Tag**. Leave the other parameters as default.

Figure 1-4 Configuring the build action



Configure Build Environment
Configure the global runtime environment for the build task.

Configure Code Download
Configure the code download mode.

Build with Maven
Build a Java project with Apache Maven.

Build Image and Push to SWR
Build an image with a Dockerfile and push the image to SWR.

Build Image and Push to SWR
Build an image with a Dockerfile and push the image to SWR.

* Action Name: Build Image and Push to SWR
* Tool Version: docker18.03
* Image Repository (Huawei Cloud SWR): SWR
* Authorized User: Current
* Push Region:
* Organization: hwstaff_codeci_gray
* Image Name: release_image
* Image Tag: v1.0
Working Directory: .

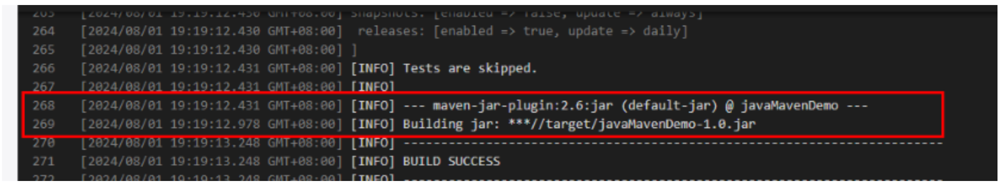
Step 6 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

----End

Viewing the Build Results

Step 1 On the **Actions** tab, check that the console displays the logs of the build task as it runs. The **Build Logs** window automatically scrolls down to show new entries. As shown in Figure 1-5, the build log window displays the information about the product of the Maven build.

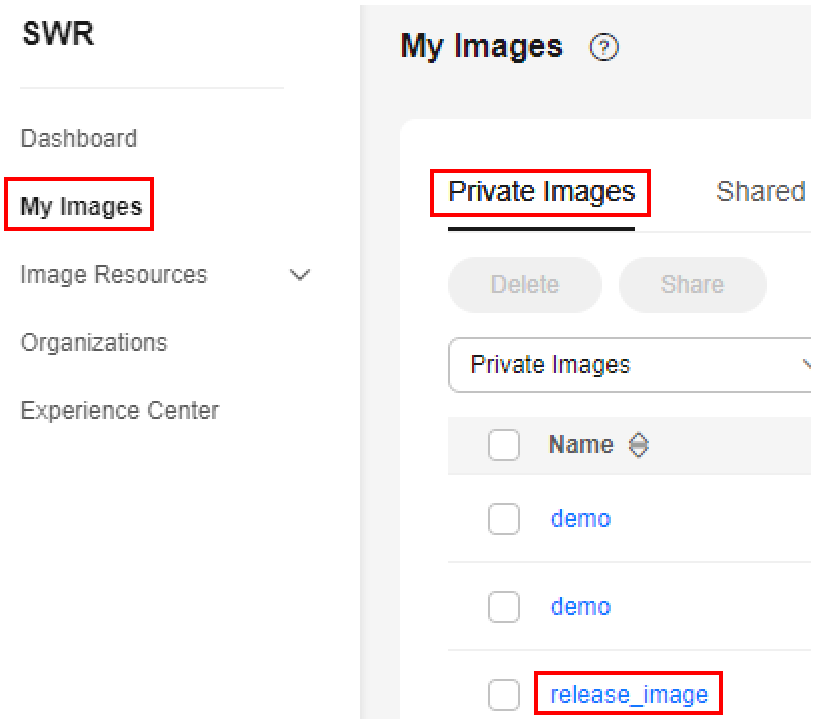
Figure 1-5 Information about the Maven build product



```
[2024/08/01 19:19:12.430 GMT+08:00] snapshots: [enabled => false, update => always]
[2024/08/01 19:19:12.430 GMT+08:00]  releases: [enabled => true, update => daily]
[2024/08/01 19:19:12.430 GMT+08:00] ]
[2024/08/01 19:19:12.431 GMT+08:00] [INFO] Tests are skipped.
[2024/08/01 19:19:12.431 GMT+08:00] [INFO]
[2024/08/01 19:19:12.431 GMT+08:00] [INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ javaMavenDemo ---
[2024/08/01 19:19:12.978 GMT+08:00] [INFO] Building jar: ***//target/javaMavenDemo-1.0.jar
[2024/08/01 19:19:13.248 GMT+08:00] [INFO] ------------------------------------------------------------------------
[2024/08/01 19:19:13.248 GMT+08:00] [INFO] BUILD SUCCESS
[2024/08/01 19:19:13.248 GMT+08:00] [INFO] ------------------------------------------------------------------------
```

Step 2 After the build task is successfully run, go to the SWR console. Choose **My Images** and click the **Private Images** tab (which is displayed by default). From there, click the name of the image (**release_image**) created in Step 5 to access its details page.

Figure 1-6 Image list



----End

2 Building with Maven and Uploading the Software Package to the Self-hosted Repo (Built-in Executors, GUI)

Scenario

CodeArts Build provides default dependency repositories. If they fall short of your service needs, you can create custom repositories for your builds. In this practice, you build with Maven and upload the resulting package to the self-hosted repo for future use. Same procedures apply when you work with other programming languages.

These steps depend on the following services:

- **CodeArts Repo**: stores your project code.
- **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.

Procedure

Table 2-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a Self-hosted Repo	Create a self-hosted repo.
Creating a CodeArts Repo Repository	Create a code repository.

Step	Description
Creating and Running a Build Task	Create, configure, and run a build task.
Viewing the Build Results	Check the build logs and result files to verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a Self-hosted Repo

Step 1 In the navigation pane, choose **Artifact** > **Self-hosted Repos**.

Step 2 Click **Create** and set parameters according to Table 2-2.

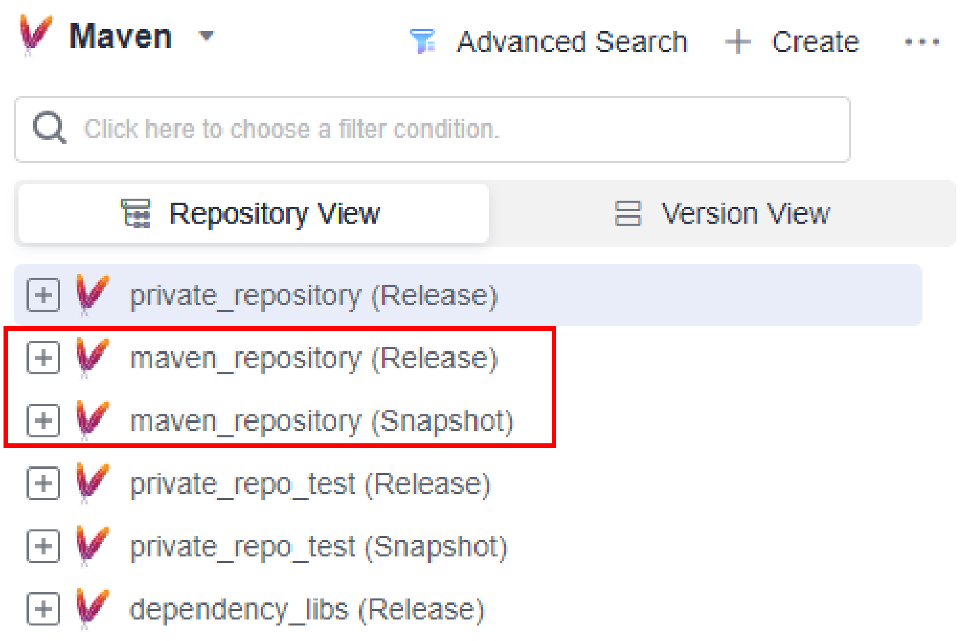
Table 2-2 Parameters for creating a self-hosted repo

Parameter	Description
Repository Type	Select **Local Repository**.
Repository Name	Assign a custom name to the repository, for example, **maven_repository**.
Package Type	Select **Maven**.
Project	Leave the value as default. The field is autofilled with **build-bestpractice** in this practice.
Include Patterns	Leave it blank.

Parameter	Description
Version Policy	Specify the version of artifacts stored in the repository. Select either **Release** (the release version with stable functions) or **Snapshot** (the development version with unstable functions). You can also select both of them if needed. In this practice, select both **Release** and **Snapshot**.
Descriptio n	Enter additional information to describe the repository. Use no more than 200 characters.

Step 3 Click **Submit**. The details page of **maven_repository** is displayed. **Figure 2-1** shows the created self-hosted repo.

Figure 2-1 Self-hosted repo


Maven
Advanced Search
Create
Click here to choose a filter condition.
Repository View
Version View
private_repository (Release)
maven_repository (Release)
maven_repository (Snapshot)
private_repo_test (Release)
private_repo_test (Snapshot)
dependency_libs (Release)

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

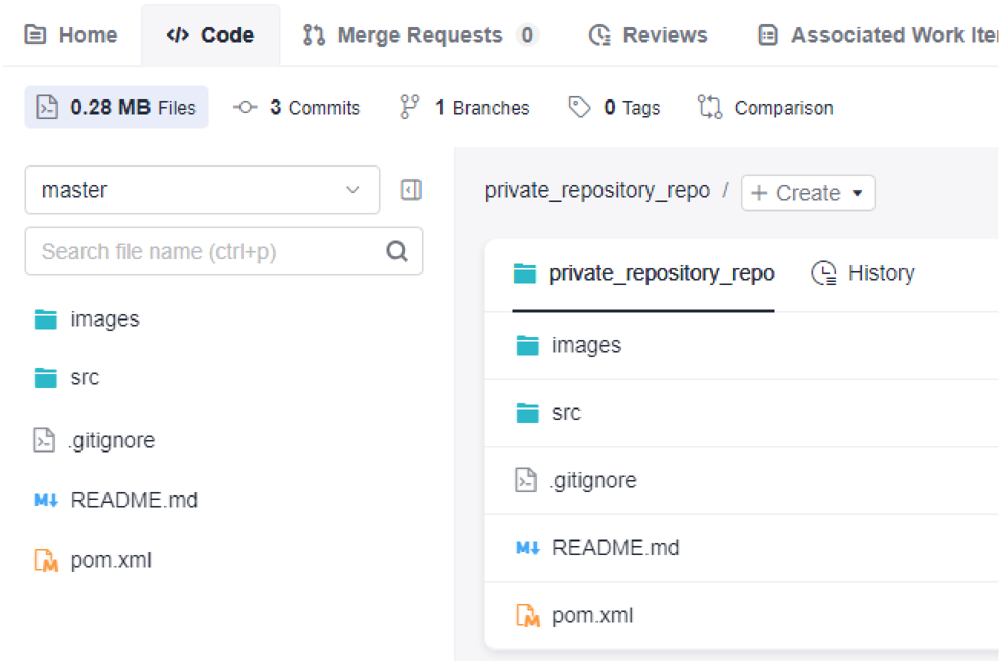
Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the repository creation page, type **maven_private_repository_repo** in the **Repository Name** field, and leave the other parameters as default.

Step 5 Click **OK**. The code repository details page is displayed. **Figure 2-2** shows the directory that stores files of the code repository.

Figure 2-2 Directory



----End

Creating and Running a Build Task

Step 1 Access CodeArts Build. Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.

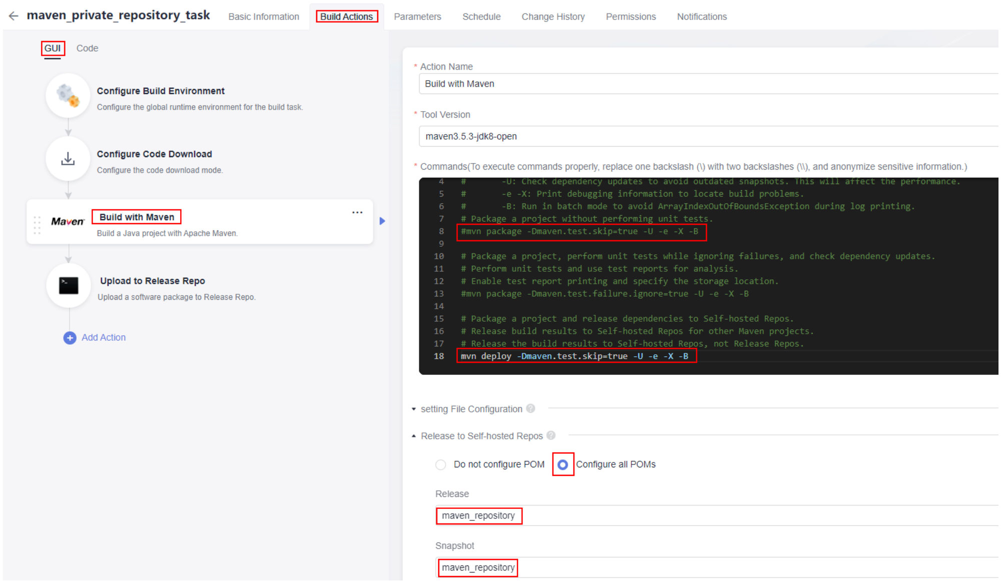
- **Name**: Assign a custom name to the build task, for example, **maven_private_repository_task**.
- **Code Source**: Select the code source from which code will be pulled for this build. In this practice, select **Repo**.
- **Repository**: Select **maven_private_repository_repo**, the code repository created in **Creating a CodeArts Repo Repository**.

Step 2 Click **Next** and select the **Maven** template. Click **OK**. The **Build Actions** page is displayed.

Step 3 On the **Build Actions** page, click the **GUI** tab. Click the **Build with Maven** action on the left, and set the following parameters. Leave the other parameters as default.

- **Commands**: Add a number sign (#) before the **mvn package -Dmaven.test.skip=true -U -e -X -B** command and delete the number sign before the **#mvn deploy -Dmaven.test.skip=true -U -e -X -B** command.
- **Release to Self-hosted Repos**: Select **Configure all POMs**.
- **Release** and **Snapshot**: Select the name (**maven_repository** is used in this practice) of the **created self-hosted repo**.

Figure 2-3 Configuring the action of building with Maven


maven_private_repository_task
Basic Information
Build Actions
Parameters
Schedule
Change History
Permissions
Notifications
GUI
Code
Configure Build Environment
Configure the global runtime environment for the build task.
Configure Code Download
Configure the code download mode.
Build with Maven
Build a Java project with Apache Maven.
Upload to Release Repo
Upload a software package to Release Repo.
Add Action

* Action Name: Build with Maven

* Tool Version: maven3.5.3-jdk8-open

* Commands(To execute commands properly, replace one backslash (\) with two backslashes (\\), and anonymize sensitive information.)

```
#       -U: Check dependency updates to avoid outdated snapshots. This will affect the performance.
#       -e -X: Print debugging information to locate build problems.
#       -B: Run in batch mode to avoid ArrayIndexOutOfBoundsException during log printing.
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B

# Package a project, perform unit tests while ignoring failures, and check dependency updates.
# Perform unit tests and use test reports for analysis.
# Enable test report printing and specify the storage location.
#mvn package -Dmaven.test.failure.ignore=true -U -e -X -B

# Package a project and release dependencies to Self-hosted Repos.
# Release build results to Self-hosted Repos for other Maven projects.
# Release the build results to Self-hosted Repos, not Release Repos.
mvn deploy -Dmaven.test.skip=true -U -e -X -B
```

setting File Configuration

Release to Self-hosted Repos

Do not configure POM [✓ Configure all POMs]

Release: maven_repository

Snapshot: maven_repository

Step 4 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

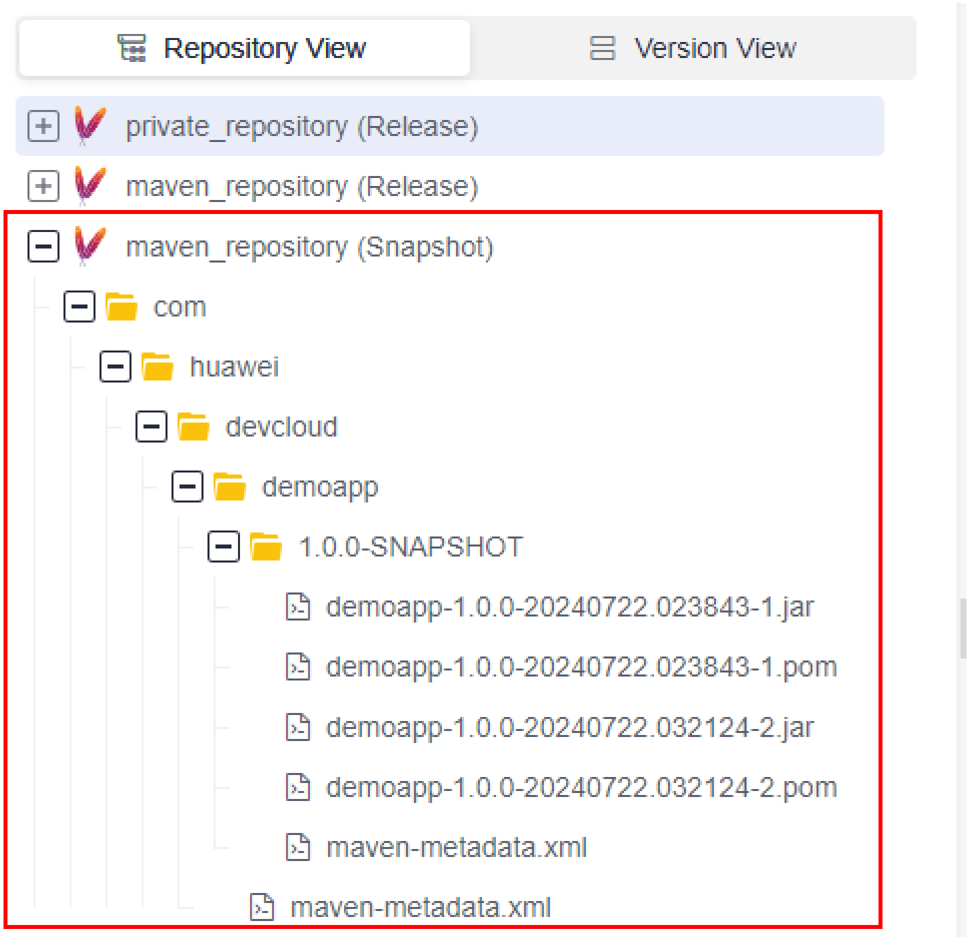
----End

Viewing the Build Results

Step 1 After the build task is successfully run, go to the **Actions** tab page. Click the **Build with Maven** action. The build log window will display the information about the upload of the Maven build product to the self-hosted repo.

Step 2 In the navigation pane, choose **Artifact** > **Self-hosted Repos**. On the displayed page, expand the directory of **maven_repository (Snapshot)** to view the released dependencies, as shown in **Figure 2-4**.

Figure 2-4 Checking the software package



```
private_repository (Release)
maven_repository (Release)
maven_repository (Snapshot)
  com
    huawei
      devcloud
        demoapp
          1.0.0-SNAPSHOT
            demoapp-1.0.0-20240722.023843-1.jar
            demoapp-1.0.0-20240722.023843-1.pom
            demoapp-1.0.0-20240722.032124-2.jar
            demoapp-1.0.0-20240722.032124-2.pom
            maven-metadata.xml
          maven-metadata.xml
```

----End

Related Information

This practice shows how to archive **Snapshot**, the unreleased development version. To archive the unchanged **Release** version, remove the snapshot suffix from the **version** (change **1.0.0-SNAPSHOT** to **1.0.0**) in the **pom.xml** file of **the CodeArts Repo repository you created**, commit the file change, and run the build task again.

During a build process, Maven differentiates between snapshot and release versions by the occurrence of the **-SNAPSHOT** suffix in the module's version number (the value of **version** in the **pom.xml** file).

3 Building with Maven to Generate a Private Dependency for Another Build (Built-in Executors, GUI)

Scenario

In this practice, you upload the artifact of a Maven build project to a self-hosted repo and include it as a second-party or third-party dependency within another Maven build project, which then downloads and uses this artifact to build an application in CodeArts Build. You will need:

- **dependency_task**: This build project releases the resulting **dependencyProject-1.0.jar** package defined in the **pom.xml** file to the self-hosted repo. The package will be used for build project **release_task**.
- **release_task**: This build project looks for the **dependencyProject-1.0.jar** package released by build project **dependency_repo** in the self-hosted repo, includes the package as a dependency in the build process, and releases the final application.
- **dependency_libs**: This self-hosted repo stores the **dependencyProject-1.0.jar** package released by build project **dependency_task**. Build project **release_task** will download the package from this repo.

  These steps depend on the following services:

  - **CodeArts Repo**: stores your project code.
  - **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.

Procedure

Table 3-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a Self-hosted Repo	Create a self-hosted repo.
Creating Code Repository dependency_repo	Create the code repository for releasing the **dependencyProject-1.0.jar** package.
Creating Build Task dependency_task	Create the build task for releasing the **dependencyProject-1.0.jar** package.
Creating Code Repository release_repo	Create a code repository for releasing the application.
Creating Build Task release_task	Create the build task that depends on the **dependencyProject-1.0.jar** package to build and release the application.
Viewing the Build Results	Verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a Self-hosted Repo

Step 1 In the navigation pane, choose **Artifact** > **Self-hosted Repos**.

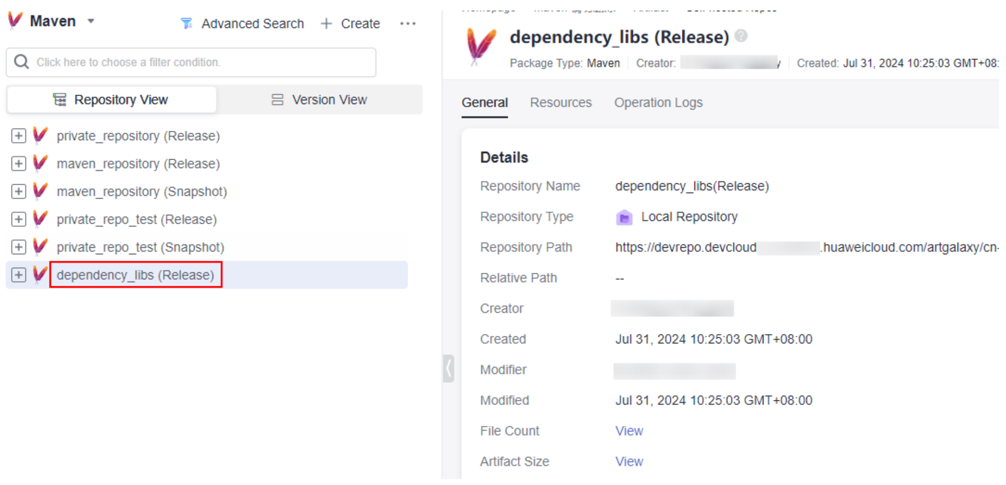
Step 2 Click **Create** and set parameters according to **Table 3-2**.

Table 3-2 Parameters for creating a self-hosted repo

Parameter	Description
Repository Type	Select **Local Repository**.
Repository Name	Assign a custom name to the repository, for example, **dependency_libs**.
Package Type	Select **Maven**.
Project	Leave the value as default. The field is autofilled with **build-bestpractice** in this practice.
Include Patterns	Leave it blank.
Version Policy	Specify the version of artifacts stored in the repository. Select either **Release** (the release version with stable functions) or **Snapshot** (the development version with unstable functions). You can also select both of them if needed. In this practice, select **Release**.
Description	Enter additional information to describe the repository. Use no more than 200 characters.

Step 3 Click **Submit**. The details page of **dependency_libs** is displayed. **Figure 3-1** shows the created self-hosted repo.

Figure 3-1 Self-hosted repo



private_repository (Release)
maven_repository (Release)
maven_repository (Snapshot)
private_repo_test (Release)
private_repo_test (Snapshot)
dependency_libs (Release)

dependency_libs (Release)
Package Type: Maven | Creator: [redacted] | Created: Jul 31, 2024 10:25:03 GMT+08:

Details

Repository Name	dependency_libs(Release)
Repository Type	Local Repository
Repository Path	https://devrepo.devcloud[redacted].huaweicloud.com/artgalaxy/cn-
Relative Path	--
Creator	[redacted]
Created	Jul 31, 2024 10:25:03 GMT+08:00
Modifier	[redacted]
Modified	Jul 31, 2024 10:25:03 GMT+08:00
File Count	View
Artifact Size	View

----End

Creating Code Repository dependency_repo

Step 1 In the navigation pane, choose **Code** > **Repo**.

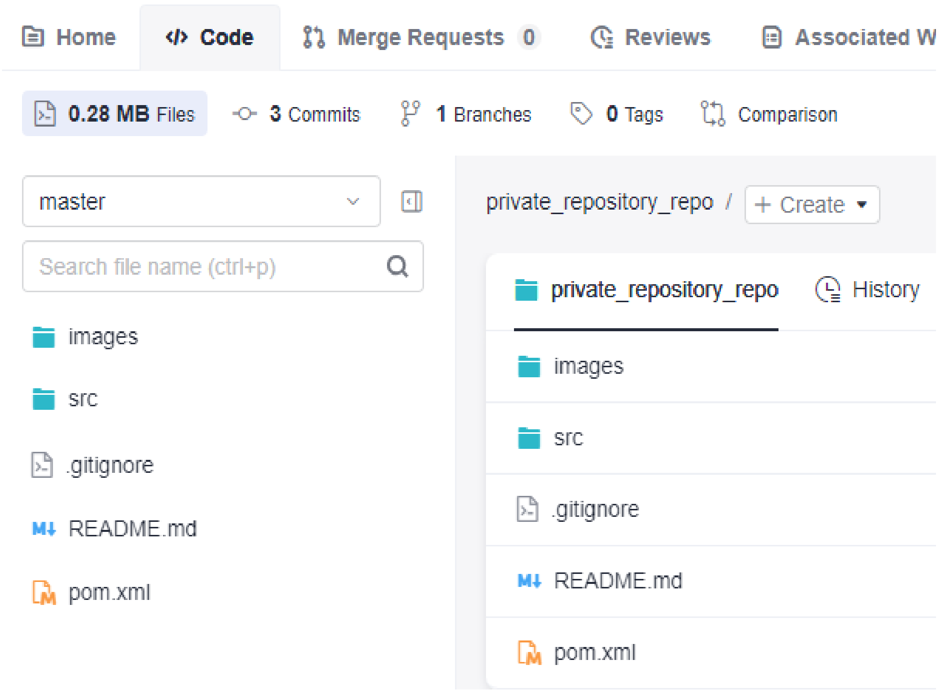
Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the repository creation page, type **dependency_repo** in the **Repository Name** field, and leave the other parameters as default.

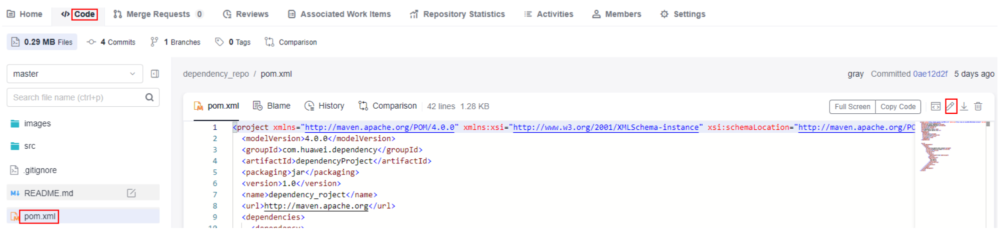
Step 5 Click **OK**. The code repository details page is displayed. Figure 3-2 shows the directory that stores files of the code repository.

Figure 3-2 Directory



Step 6 Click the **pom.xml** file to enter its details page. In the right pane, Click ✎ to access the file editor.

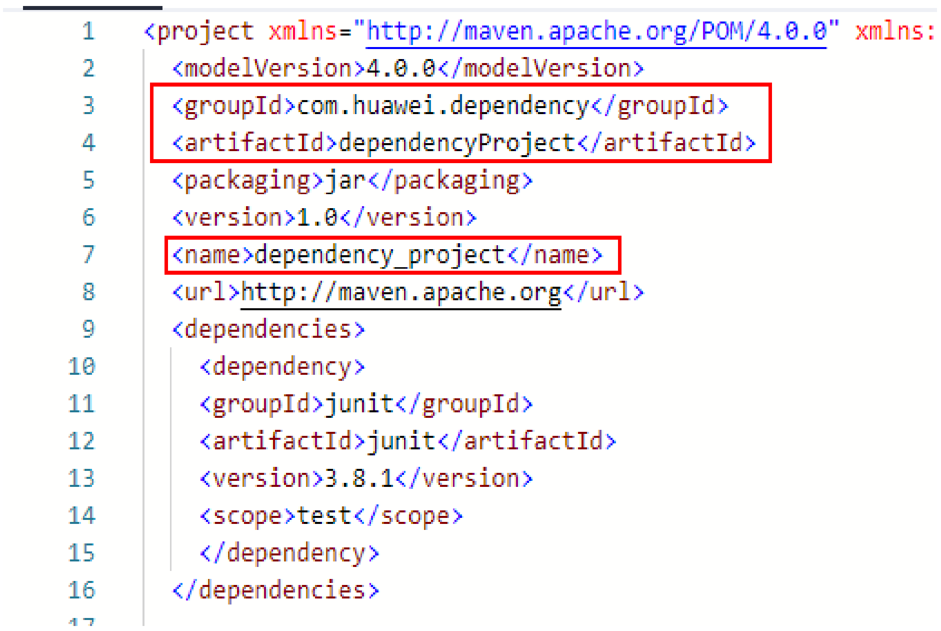
Figure 3-3 Editing the pom.xml file



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/PC
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.dependency</groupId>
  <artifactId>dependencyProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>dependency_roject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
```

Step 7 Change **groupId** to **com.huawei.dependency**, **artifactId** to **dependencyProject**, and **name** to **dependency_project**, as shown in Figure 3-4. Click **OK** to save your changes to the **pom.xml** file.

Figure 3-4 Modifying the pom.xml file



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.dependency</groupId>
  <artifactId>dependencyProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>dependency_project</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
  </dependencies>
```

----End

Creating Build Task dependency_task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.

- **Name**: Assign a custom name to the build task, for example, **private_repository_task**.
- **Code Source**: Select **Repo**.
- **Repository**: Select **private_repository_repo**, the code repository created in **Creating Code Repository dependency_repo**.

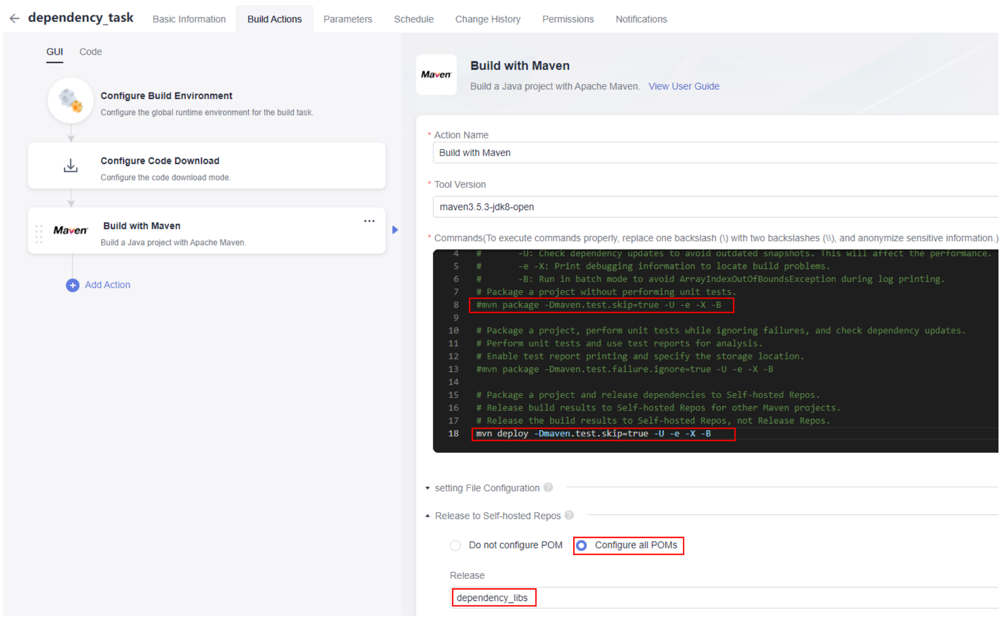
Step 3 Click **Next**. On the displayed page, select **Blank Template**. Click **OK**. The **Build Actions** page is displayed.

Step 4 On the **Build Actions** page, click the **GUI** tab. Then click **Add Build Actions** on the left and add the **Build with Maven** action.

Step 5 Click the **Build with Maven** action, and set the following parameters. Leave the other parameters as default.

- **Commands**: Add a number sign (#) before the **mvn package -Dmaven.test.skip=true -U -e -X -B** command and delete the number sign before the **#mvn deploy -Dmaven.test.skip=true -U -e -X -B** command.
- **Release to Self-hosted Repos**: Select **Configure all POMs**.
- **Release**: Select the name (**dependency_libs** is used in this practice) of the **created self-hosted repo**.

Figure 3-5 Configuring the action of building with Maven



Configure Build Environment
Configure the global runtime environment for the build task.

Configure Code Download
Configure the code download mode.

Build with Maven
Build a Java project with Apache Maven.

Build with Maven
Build a Java project with Apache Maven. View User Guide

* Action Name: Build with Maven

* Tool Version: maven3.5.3-jdk8-open

* Commands(To execute commands properly, replace one backslash (\) with two backslashes (\\), and anonymize sensitive information.)

```
#       -U: Check dependency updates to avoid outdated snapshots. This will affect the performance.
#       -e -X: Print debugging information to locate build problems.
#       -B: Run in batch mode to avoid ArrayIndexOutOfBoundsException during log printing.
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B

# Package a project, perform unit tests while ignoring failures, and check dependency updates.
# Perform unit tests and use test reports for analysis.
# Enable test report printing and specify the storage location.
#mvn package -Dmaven.test.failure.ignore=true -U -e -X -B

# Package a project and release dependencies to Self-hosted Repos.
# Release build results to Self-hosted Repos for other Maven projects.
# Release the build results to Self-hosted Repos, not Release Repos.
mvn deploy -Dmaven.test.skip=true -U -e -X -B
```

setting File Configuration

Release to Self-hosted Repos

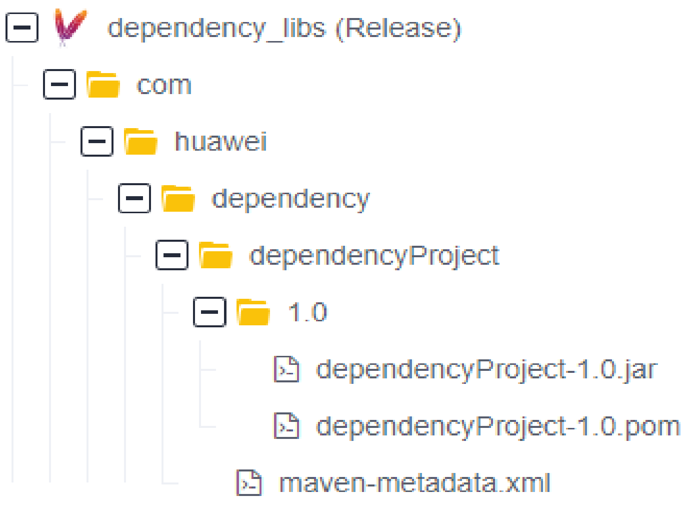
Do not configure POM [✓ Configure all POMs]

Release: dependency_libs

Step 6 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

Step 7 After the build task is complete, check the build products. In the navigation pane, choose **Artifact** > **Self-hosted Repos**. On the displayed page, expand the directory of **dependency_libs (Release)** to check the released dependencies as build outputs, as shown in **Figure 3-6**.

Figure 3-6 Checking dependencies



```
dependency_libs (Release)
  com
    huawei
      dependency
        dependencyProject
          1.0
            dependencyProject-1.0.jar
            dependencyProject-1.0.pom
          maven-metadata.xml
```

----End

Creating Code Repository release_repo

Step 1 In the navigation pane, choose **Code** > **Repo**.

Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the repository creation page, type **release_repo** in the **Repository Name** field, and leave the other parameters as default.

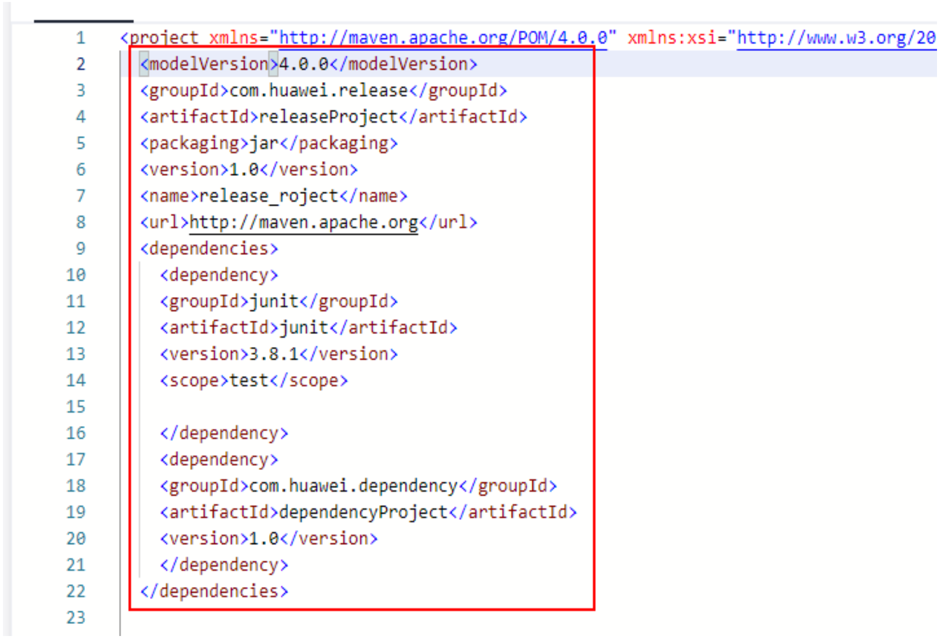
Step 5 Click **OK**. The code repository details page is displayed.

Step 6 Click the **pom.xml** file to enter its details page. In the right pane, Click ✎ to access the file editor. Copy the following code to the red box shown in **Figure 3-7**. Click **OK** to save your changes to the **pom.xml** file.

The following sample code adds the **dependencyProject-1.0.jar** package generated as the output of the **build task dependency_task** in the **dependency** element.

```
<modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.release</groupId>
  <artifactId>releaseProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>release_roject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
    <dependency>
    <groupId>com.huawei.dependency</groupId>
    <artifactId>dependencyProject</artifactId>
    <version>1.0</version>
    </dependency>
  </dependencies>
```

Figure 3-7 Modifying the pom.xml file



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/20
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.release</groupId>
  <artifactId>releaseProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>release_roject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>

    </dependency>
    <dependency>
    <groupId>com.huawei.dependency</groupId>
    <artifactId>dependencyProject</artifactId>
    <version>1.0</version>
    </dependency>
  </dependencies>
```

----End

Creating Build Task release_task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.

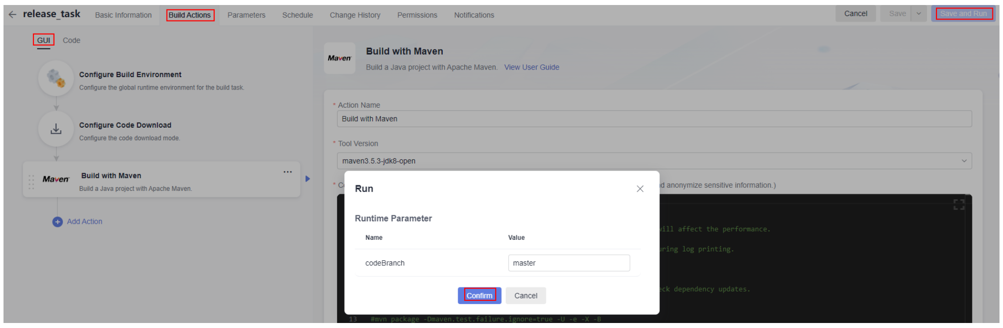
- **Name**: Assign a custom name to the build task, for example, **release_task**.
- **Code Source**: Select **Repo**.
- **Repository**: Select **release_repo**, the code repository created in **Creating Code Repository release_repo**.

Step 3 Click **Next**. On the displayed page, select **Blank Template**. Click **OK**. The **Build Actions** page is displayed.

Step 4 On the **Build Actions** page, click the **GUI** tab. Then click **Add Build Actions** on the left and add the **Build with Maven** action. Leave the parameters as default.

Step 5 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

Figure 3-8 Running a build task


release_task
Basic Information
Build Actions
Parameters
Schedule
Change History
Permissions
Notifications
Cancel
Save
Save and Run
GUI
Code
Configure Build Environment
Configure the global runtime environment for the build task.
Configure Code Download
Configure the code download mode.
Build with Maven
Build a Java project with Apache Maven.
Add Action
Build with Maven
Build a Java project with Apache Maven. View User Guide
Confirm
Cancel

* Action Name: Build with Maven
* Tool Version: maven3.5.3-jdk8-open

Run

Runtime Parameter

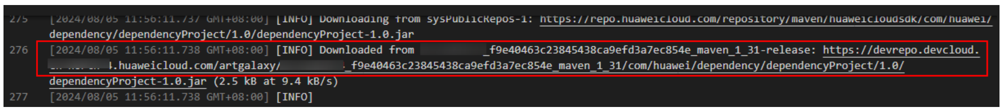
Name	Value
codeBranch	master

----End

Viewing the Build Results

On the **Actions** tab, check that the console displays the logs of the build task as it runs. The **Build Logs** window automatically scrolls down to show new entries. As shown in the following figure, the build log indicates that the **dependencyProject-1.0.jar** package produced as the output of the **build task dependency_task** is pulled as a dependency from the self-hosted repo **dependency_libs**.

Figure 3-9 Viewing the build log



```
[2024/08/05 11:56:11.737 GMT+08:00] [INFO] Downloading from sysPublicRepos-1: https://repo.huaweicloud.com/repository/maven/huaweicloudsdk/com/huawei/
dependency/dependencyProject/1.0/dependencyProject-1.0.jar
[2024/08/05 11:56:11.738 GMT+08:00] [INFO] Downloaded from [redacted]_f9e40463c23845438ca9efd3a7ec854e_maven_1_31-release: https://devrepo.devcloud.
[redacted].huaweicloud.com/artgalaxy/[redacted]_f9e40463c23845438ca9efd3a7ec854e_maven_1_31/com/huawei/dependency/dependencyProject/1.0/
dependencyProject-1.0.jar (2.5 kB at 9.4 kB/s)
[2024/08/05 11:56:11.738 GMT+08:00] [INFO]
```

4 Building with npm and Uploading the Software Package to the Release Repo (Built-in Executors, GUI)

Scenario

In this practice, you use built-in executors provided by CodeArts Build to compile a Node.js project and upload the resulting software package to the release repo. These steps will be carried out through the graphical user interface (GUI).

These steps depend on the following services:

- **CodeArts Repo**: stores your project code.
- **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.

Procedure

Table 4-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a CodeArts Repo Repository	Create a CodeArts Repo repository.
Creating a Build Task	Create a build task.

Step	Description
Configuring Build Actions and Running the Build Task	Configure build actions and run the build task.
Viewing and Verifying the Build Results	View and verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

Step 2 Access CodeArts Repo, click **New Repository**. On the displayed page, select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Nodejs Webpack Demo** template and click **Next**.

Step 4 On the repository creation page, type **nodesource** in the **Repository Name** field, leave the other parameters as default, and click **OK**.

----End

Creating a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 4-2**. Then, click **Next**.

Table 4-2 Basic information

Parameter	Description
Name	Assign a custom name to the build task, for example, **npm_yml_build**.
Code Source	Select **Repo**.
Repository	Select **nodesource**, the code repository created in **Creating a CodeArts Repo Repository**.
Default Branch	Keep the default value **master**.
Description	Enter additional information to describe the build task.

Step 3 Select the **npm** template and click **OK**. The **Build Actions** page is displayed.

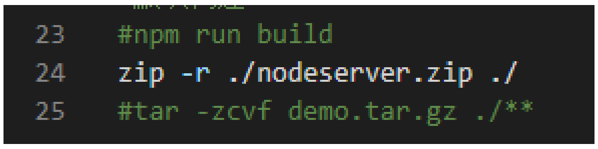
----End

Configuring Build Actions and Running the Build Task

Step 1 Configure action **Build with npm**.

In the command editor, add **#** before the **npm run build** command. Then add the **zip -r ./nodeserver.zip ./** command to pack the code into **nodeserver.zip**, as shown in **Figure 4-1**. Leave the other parameters as default.

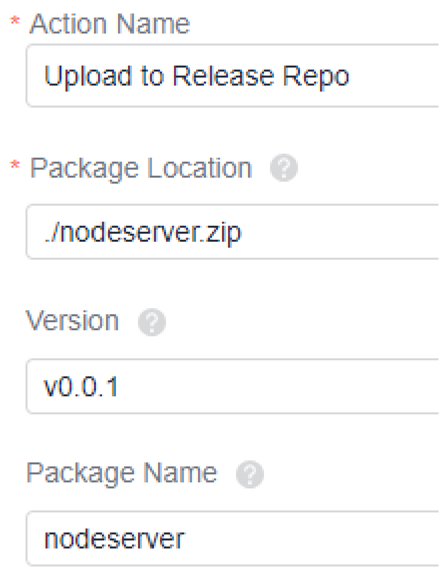
Figure 4-1 Command example



```
#npm run build
zip -r ./nodeserver.zip ./
#tar -zcvf demo.tar.gz ./**
```

Step 2 Configure action **Upload to Release Repo** according to **Figure 4-2**.

Figure 4-2 Configuring the action of uploading a software package to the release repo



* Action Name

Upload to Release Repo

* Package Location

./nodeserver.zip

Version

v0.0.1

Package Name

nodeserver

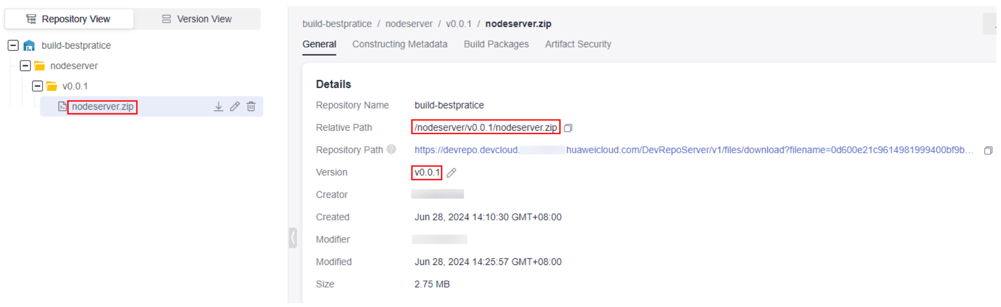
Step 3 Once you have finished configuring all build actions, click **Save and Run** to run the build task.

----End

Viewing and Verifying the Build Results

In the navigation pane, choose **Artifact** > **Release Repos**. On the displayed page, find the uploaded software package, as shown in **Figure 4-3**.

Figure 4-3 Checking the uploaded the software package



Details

Repository Name	build-bestpratice
Relative Path	/nodeserver/v0.0.1/nodeserver.zip
Repository Path	https://devrepo.devcloud.[redacted]huaweicloud.com/DevRepoServer/v1/files/download?filename=0d600e21c9614981999400bf9b...
Version	v0.0.1
Creator	[redacted]
Created	Jun 28, 2024 14:10:30 GMT+08:00
Modifier	[redacted]
Modified	Jun 28, 2024 14:25:57 GMT+08:00
Size	2.75 MB

The package name and release version are the same as those configured in **Step 2**.

5 Building with Maven (Custom Executors, GUI)

Scenario

If the built-in executors provided by CodeArts Build fall short of your service needs, you can use your own computing resources. Once registered, these computing resources can be hosted in CodeArts Build. You can then configure CodeArts Build to schedule them for running build tasks. In this practice, you configure actions **Build with Maven** and **Upload to Release Repo** and run your build on custom executors.

These steps depend on the following services:

- **CodeArts Repo**: stores your project code.
- **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.

Prerequisites

You have purchased an Elastic Cloud Server (ECS) for your custom executors by referring to **Purchasing an ECS**.

Procedure

Table 5-1 Steps

Step	Description
Creating a Project	Create a project.
Creating an Agent Pool	Create a pool of custom executors (agent pool).

Step	Description
Creating a CodeArts Repo Repository	Create a repository to store code.
Creating and Running a Build Task	Create a build task that includes actions **Build with Maven** and **Upload to Release Repo**.
Viewing the Build Task and Its Result	Check the build logs to verify the executors used for the build task and view the uploaded software package in CodeArts Artifact.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating an Agent Pool

Step 1 On the navigation bar, click the username and choose **All Account Settings**.

Step 2 Choose **Agent Management** > **Agent Pool**.

Step 3 Click **Create Pool**. In the displayed dialog box, set parameters according to Table 5-2 and click **Save**.

Table 5-2 Agent pool configuration

Parameter	Description
Pool Name	Assign a custom name to the pool, for example, **custom_pool**.
Pool Type	Select **LINUX_DOCKER**. When a task is initiated, a Linux Docker container will be started to run the task.

Parameter	Description
Description	(Optional) Enter additional information to describe the pool.
This pool can be used by all users of the current account.	(Optional) Selecting this option allows all users within the current account to use the pool.

Step 4 Click the name of the new pool (**custom_pool** is used in this practice). The pool configuration page is displayed.

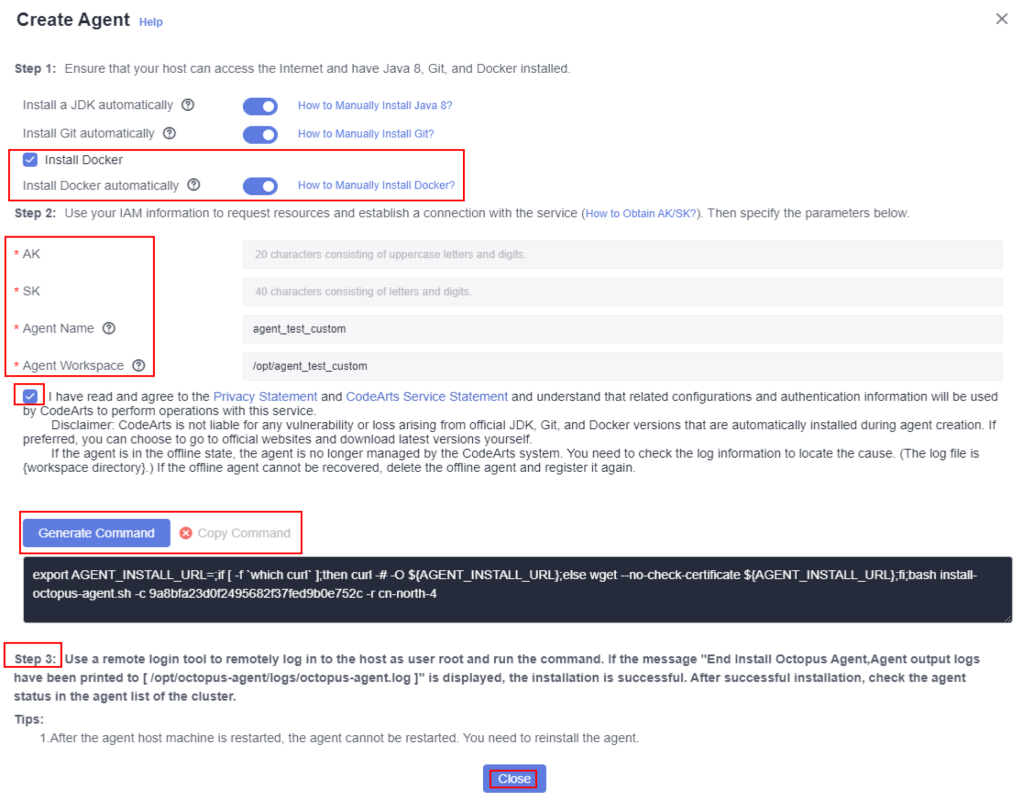
Step 5 Click **Create Agent**. In the displayed dialog box, configure the agent according to **Table 5-3** and leave the other parameters as default.

Table 5-3 Parameters for creating an agent

Parameter	Description
Install Docker	Selecting this option mandates Docker installation.
Install Docker automatically	Toggling on the switch will automatically install Docker.
AK	Obtain the AK by referring to **Obtaining an AK/SK**.
SK	Obtain the SK by referring to **Obtaining an AK/SK**.
Agent Name	Assign a custom name to the agent, for example, **agent_test_custom**.
Agent Workspace	Enter an agent workspace that follows the standard Linux directory structure. For example, **/opt/agent_test_custom**.

Step 6 Select the check box to confirm that you have read and accept the agreements. Then click **Generate Command** and **Copy Command**. Click **Close**.

Figure 5-1 Creating an agent



Step 1: Ensure that your host can access the Internet and have Java 8, Git, and Docker installed.

Install a JDK automatically [✓ on] How to Manually Install Java 8?

Install Git automatically [✓ on] How to Manually Install Git?

[✓] Install Docker

Install Docker automatically [✓ on] How to Manually Install Docker?

Step 2: Use your IAM information to request resources and establish a connection with the service (How to Obtain AK/SK?). Then specify the parameters below.

* AK	20 characters consisting of uppercase letters and digits.
* SK	40 characters consisting of letters and digits.
* Agent Name	agent_test_custom
* Agent Workspace	/opt/agent_test_custom

[✓] I have read and agree to the Privacy Statement and CodeArts Service Statement and understand that related configurations and authentication information will be used by CodeArts to perform operations with this service.

Disclaimer: CodeArts is not liable for any vulnerability or loss arising from official JDK, Git, and Docker versions that are automatically installed during agent creation. If preferred, you can choose to go to official websites and download latest versions yourself.

If the agent is in the offline state, the agent is no longer managed by the CodeArts system. You need to check the log information to locate the cause. (The log file is {workspace directory}.) If the offline agent cannot be recovered, delete the offline agent and register it again.

```
export AGENT_INSTALL_URL=;if [ -f `which curl` ];then curl -# -O ${AGENT_INSTALL_URL};else wget --no-check-certificate ${AGENT_INSTALL_URL};fi;bash install-octopus-agent.sh -c 9a8bfa23d0f2495682f37fed9b0e752c -r cn-north-4
```

Step 3: Use a remote login tool to remotely log in to the host as user root and run the command. If the message "End Install Octopus Agent,Agent output logs have been printed to [/opt/octopus-agent/logs/octopus-agent.log]" is displayed, the installation is successful. After successful installation, check the agent status in the agent list of the cluster.

Tips:

1.After the agent host machine is restarted, the agent cannot be restarted. You need to reinstall the agent.

Step 7 Go to the ECS list page, find the row of the ECS purchased to meet **prerequisites**, click **Remote Login**, and run the command copied in **Step 6**, as instructed by **Step 3**.

Step 8 On the agent list page, click **Refresh List**. After the information is automatically synchronized in the background, a new item will be added to the list. The agent alias is **agent_test_custom-mwlye1NlLG**.

----End

Creating a CodeArts Repo Repository

Step 1 On the navigation bar, choose **Services** > **Repo**. The CodeArts Repo homepage is displayed.

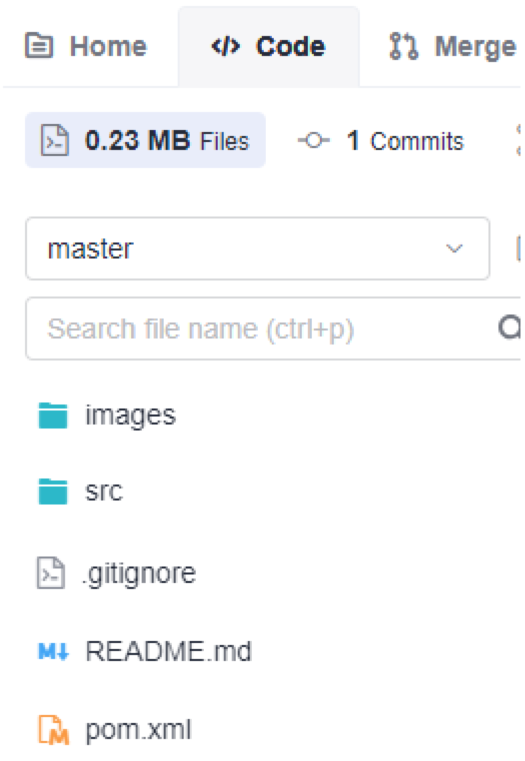
Step 2 Click **New Repository**. On the displayed page, expand the **Project** drop-down list and select the project name you specify when **creating a project**. Then, select **Template** as the repository type. Click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the displayed page, set **Repository Name** to **custom_repo**, and leave the other parameters as default. Click **OK**.

Figure 5-2 shows the directory that stores files of the code repository.

Figure 5-2 Directory that stores files of the code repository



----End

Creating and Running a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 5-4**. Then, click **Next**.

Table 5-4 Basic information

Parameter	Description
Name	Assign a custom name to the build task, for example, **custom_task**.
Code Source	Select **Repo** to pull the code hosted in a CodeArts Repo repository for your build.
Repository	Select **custom_repo**, the code repository created in **Creating a CodeArts Repo Repository**.
Default Branch	Keep the default value **master**.
Description	Enter additional information to describe the build task.

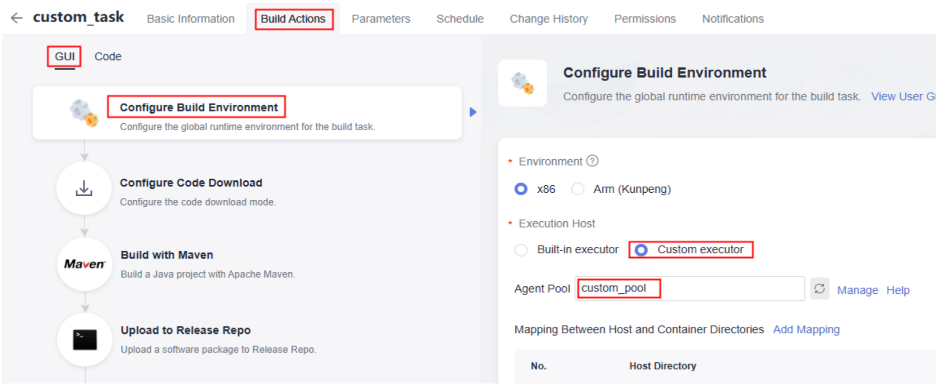
Step 3 Select the **Maven** template and click **OK**. The **Build Actions** page is displayed.

Step 4 Configure the build environment according to the following table. Leave the other parameters as default, and click **Save and Run**.

Table 5-5 Environment parameters

Parameter	Description
Execution Host	Select **Custom executor**.
Agent Pool	Expand the drop-down list and select **custom_pool** you created in section **"Creating an Agent Pool"**.

Figure 5-3 Configuring the build environment


custom_task
Basic Information
Build Actions
Parameters
Schedule
Change History
Permissions
Notifications
GUI
Code
Configure Build Environment
Configure the global runtime environment for the build task.
Configure Code Download
Configure the code download mode.
Build with Maven
Build a Java project with Apache Maven.
Upload to Release Repo
Upload a software package to Release Repo.
Configure Build Environment
Configure the global runtime environment for the build task. View User G
Manage
Help
Mapping Between Host and Container Directories
Add Mapping
No.
Host Directory

Environment: [✓ x86] Arm (Kunpeng)
Execution Host: Built-in executor [✓ Custom executor]
Agent Pool: custom_pool

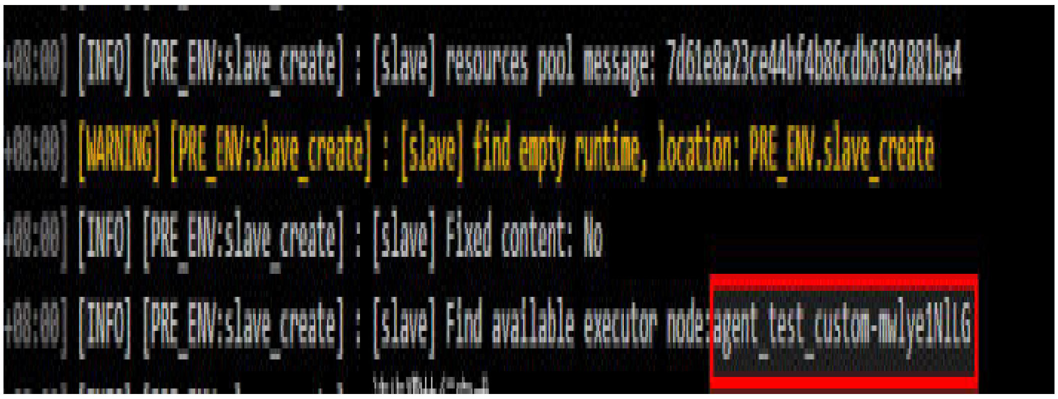
Step 5 In the displayed dialog box, click **Confirm**. The build task run page is displayed.

----End

Viewing the Build Task and Its Result

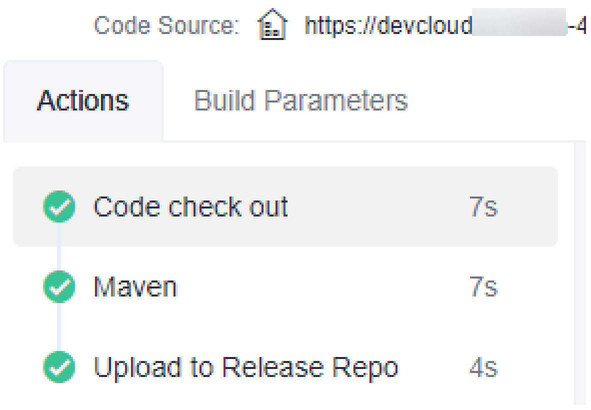
Step 1 The console displays the logs of the build task as it runs. The log window automatically scrolls down to show new entries. As shown in **Figure 5-4**, the log indicates that the current build task is running on executor **agent_test_custom-mwlye1NlLG** in pool **custom_pool** created in section **Creating an Agent Pool**.

Figure 5-4 Build logs



```
+08:00] [INFO] [PRE_ENV:slave_create] : [slave] resources pool message: 7d61e8a23ce44bf4b86cdb6191881ba4
+08:00] [WARNING] [PRE_ENV:slave_create] : [slave] find empty runtime, location: PRE_ENV.slave_create
+08:00] [INFO] [PRE_ENV:slave_create] : [slave] Fixed content: No
+08:00] [INFO] [PRE_ENV:slave_create] : [slave] Find available executor node:agent_test_custom-mwlye1NlLG
```

Step 2 The build task is considered successful if each build action is marked with ✓ after running.



Code Source: https://devcloud[redacted]-4

Actions | Build Parameters

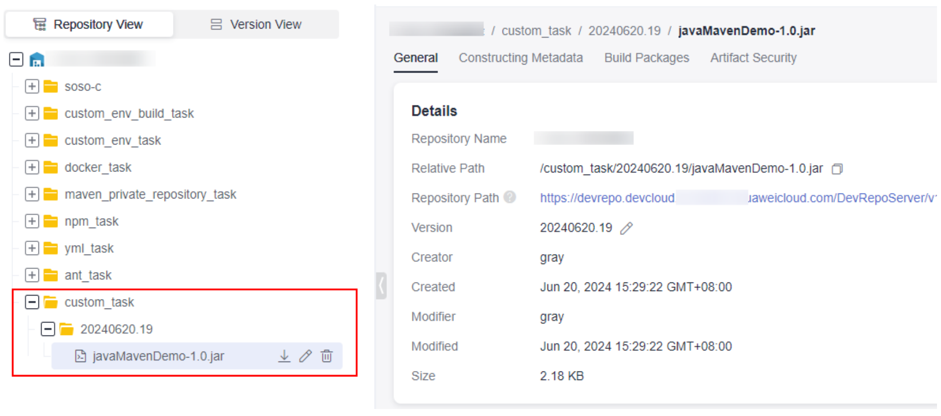
Code check out	7s
Maven	7s
Upload to Release Repo	4s

Figure 5-5 Build success

Step 3 Click action **Upload to Release Repo**. In the logs, you can find the path that stores the software package uploaded to the release repo: **/custom_task/20240620.19/**.

Step 4 In the navigation pane, choose **Artifact** > **Release Repos**. You can find the software package at **/custom_task/20240620.19/javaMavenDemo-1.0.jar**.

Figure 5-6 Software package information



```
[redacted]
  soso-c
  custom_env_build_task
  custom_env_task
  docker_task
  maven_private_repository_task
  npm_task
  yml_task
  ant_task
  custom_task
    20240620.19
      javaMavenDemo-1.0.jar
```

[redacted] / custom_task / 20240620.19 / **javaMavenDemo-1.0.jar**

Details

Repository Name	[redacted]
Relative Path	/custom_task/20240620.19/javaMavenDemo-1.0.jar
Repository Path	https://devrepo.devcloud[redacted]uaweicloud.com/DevRepoServer/v
Version	20240620.19
Creator	gray
Created	Jun 20, 2024 15:29:22 GMT+08:00
Modifier	gray
Modified	Jun 20, 2024 15:29:22 GMT+08:00
Size	2.18 KB

----End

6 Building with Maven, Uploading the Software Package, and Pushing the Image to SWR (Built-in Executors, Code)

Scenario

CodeArts Build allows you to define your build as code using YAML. Your configurations, such as build environments, parameters, commands, and actions, reside in a YAML file (named **build.yml** in this practice). After creating this file, add it along with the source code to a code repository. The file will be used as a script by the system to run a build, making the process traceable, recoverable, secure, and reliable. In this practice, you build with Maven, upload the resulting software package to the release repo, and push the image to SWR.

These steps depend on the following services:

- **SoftWare Repository for Container (SWR)**: provides repositories to store Docker images uploaded by users. These images can be used in build, deployment, and other scenarios.
- **CodeArts Repo**: stores your project code.
- **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have **created an organization** named **codeci_gray** in SWR.
- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.
- You can only use the code hosted in CodeArts Repo for YAML builds.

Procedure

Table 6-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a CodeArts Repo Repository	Create a CodeArts Repo repository to store code files.
Creating a build.yml File	Define the entire build process in **build.yml**.
Creating a Dockerfile	Customize an image by modifying the Dockerfile.
Creating a Build Task	Create a build task.
Viewing and Verifying the Build Results	View and verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the repository creation page, type **maven_yml_build** in the **Repository Name** field, and leave the other parameters as default.

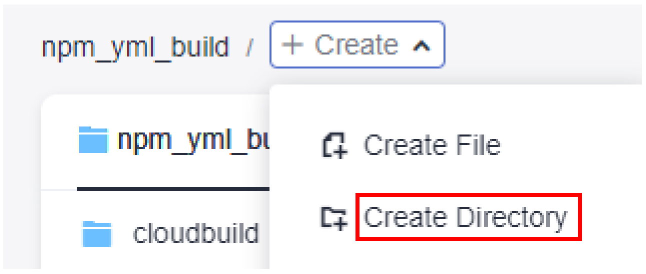
Step 5 Click **OK**. The repository details page is displayed.

----End

Creating a build.yml File

Step 1 On the code repository details page, click **Create** and select **Create Directory** from the drop-down list, as shown in **Figure 6-1**.

Figure 6-1 Creating a directory



Step 2 On the **Create Directory** page, set parameters based on **Table 6-2** and click **OK**.

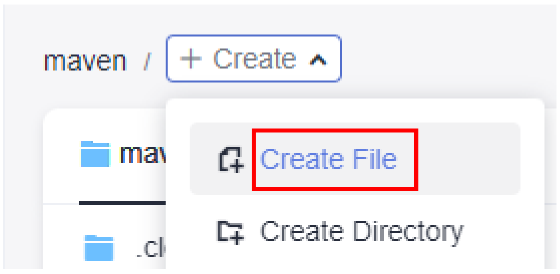
Table 6-2 Creating a directory

Parameter	Description
Directory Name	Enter a directory name, for example, **.cloudbuild**. Use 1 to 100 characters, including letters, digits, slashes (/), underscores (_), hyphens (-), and periods (.).
Commit Message	Describe the files within the directory. Use 1 to 2,000 characters.

Step 3 Click the name of the directory created in **Step 2**.

Step 4 Click **Create** and select **Create File** from the drop-down list, as shown in **Figure 6-2**.

Figure 6-2 Creating a file



Step 5 Name the file **build.yml** and copy the following code to the file:

```
# The YAML provided is a default template that can be edited as needed.
---
version: 2.0
steps:
```

```
BUILD:
- maven:
    image: cloudbuild@maven3.5.3-jdk8-open # The image path can be customized.
    inputs:
      settings:
        public_repos:
          - https://mirrors.huawei.com/maven
      cache: true # Determine whether to enable caching.
      command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B
- upload_artifact:
    inputs:
      path: "**/target/*.?ar"
- build_image:
    inputs:
      organization: codeci_gray # Organization name
      image_name: maven_demo # Image name
      image_tag: 1.0 # Image tag
      dockerfile_path: ./Dockerfile
```

Step 6 Click **Submit**.

----End

Creating a Dockerfile

Step 1 In the root directory, create a file named **Dockerfile** by following **Step 4**. The code in the file is as follows:

```
FROM swr.regionID.myhuaweicloud.com/codeci/special_base_image:centos7-base-1.0.2
MAINTAINER <devcloud@demo.com>
USER root
RUN mkdir /demo
COPY ./target/server-1.0.jar /demo/app.jar
```

server-1.0.jar combines the values of **artifactId**, **packaging**, and **version** in the **pom.xml** file.

Step 2 Click **Submit**.

----End

Creating a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 6-3**.

Table 6-3 Basic information

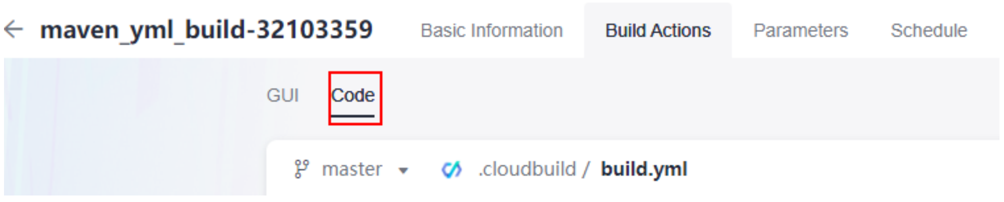
Parameter	Description
Name	Assign a custom name to the build task, for example, **maven_yml_build**.
Code Source	Select **Repo**.
Repository	Select **Repo01**, the code repository created in **Creating a CodeArts Repo Repository**.

Parameter	Description
Default Branch	Keep the default value **master**.
Description	Enter additional information to describe the build task.

Step 3 Click **Next**. On the displayed page, select **Blank Template**. Click **OK**. The **Build Actions** page is displayed.

Step 4 Click the **Code** tab. Then you can view the imported build script, as shown in **Figure 6-3**.

Figure 6-3 Code tab



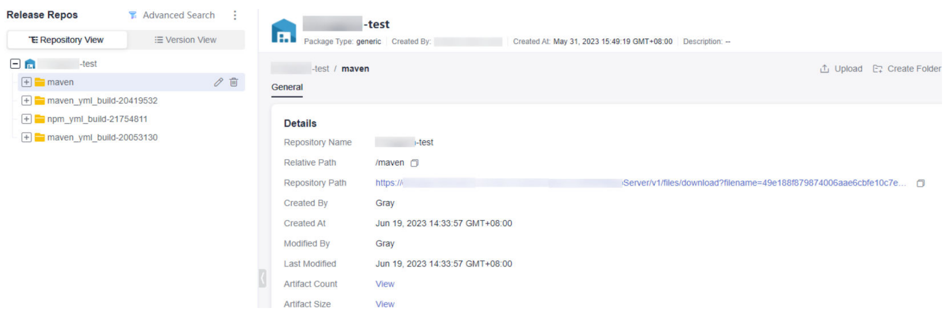
Step 5 Click **Save and Run** in the upper right corner.

----End

Viewing and Verifying the Build Results

- Checking the uploaded the software package
  a. In the navigation pane, choose **Artifact** > **Release Repos**.
  b. On the displayed page, find the folder that shares the same name as the build task (the name you specify when **creating the build task**), as shown in **Figure 6-4**. The software package can be found within this folder.

Figure 6-4 Checking the software package



[redacted]-test
Package Type: generic | Created By: [redacted] | Created At: May 31, 2023 15:49:19 GMT+08:00 | Description: --

[redacted]-test / maven

Details

Repository Name	[redacted]-test
Relative Path	/maven
Repository Path	https://[redacted]Server/v1/files/download?filename=49e188f879874006aae6cbfe10c7e...
Created By	Gray
Created At	Jun 19, 2023 14:33:57 GMT+08:00
Modified By	Gray
Last Modified	Jun 19, 2023 14:33:57 GMT+08:00
Artifact Count	View
Artifact Size	View

- Checking the pushed image
  a. Go to the **SWR** console.
  b. In the navigation pane, choose **My Images**. In the search box, select **Organization** as the filter field, and type the organization name (**codeci_gray** is used in this example) you configure when **creating a build.yml file**.

c. In the filtered results, click the image name (**maven_demo** is used in this example) you configure when **creating a build.yml file**.

7 Running a Multi-Task Maven Build Project (Built-in Executors, Code)

Scenario

A build task is the smallest unit that a build project can be broken down into for simple service scenarios. However, for more complex requirements, you may need to set up multiple build tasks in a modular and fine-grained way, and run them in a specific order. This means that each task depends on the successful completion of its dependency task.

To handle such complex builds, CodeArts Build offers a task model called BuildFlow, which organizes multiple build tasks in a directed acyclic graph (DAG) and runs them in parallel based on their dependencies.

In this practice, your build project involves three build tasks (**Job1**, **Job2**, and **Job3**). **Job3** depends on **Job1** and **Job2**. Your code is stored in **CodeArts Repo** repositories.

Requirements

- You can only use the code hosted in CodeArts Repo for a BuildFlow task.
- You have permissions for CodeArts Repo.

Procedure

Table 7-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a CodeArts Repo Repository	Create a CodeArts Repo repository to store code files.
Creating a build.yml File	Define the entire build process in **build.yml**.

Step	Description
Creating Task Scripts Used in build.yml	Create a script for the whole build.
Creating and Running a Build Task	Create and run a build flow task.
Viewing the Build Results	View the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the displayed page, set **Repository Name** to **Repo01** and keep the default values for other parameters.

Step 5 Click **OK**.

----End

Creating a build.yml File

Step 1 On the code repository details page, click **Create** and select **Create Directory** from the drop-down list.

Step 2 Name the directory as **.cloudbuild** and describe the files it contains. Click **OK**.

Step 3 In the **.cloudbuild** directory, click **Create** and select **Create File** from the drop-down list. Name the file as **build.yml** and add the following code to the file:

```
version: 2.0 # The version number is a mandatory and unique parameter that must be set to 2.0.
params: # Parameters that can be referenced by builds.
  - name: condition_param
    value: 1
# The envs configuration is optional.
envs:
  - condition: condition_param == 0 # The following host specification and type are not used if this
condition is not met.
    resource:
      type: docker
      arch: ARM
  - condition: condition_param == 1 # The following host specification and type are used if this condition is
met.
    resource:
      type: docker
      arch: X86

buildflow:
  jobs: # Build tasks
    - job: Job3 # Define the task name.
      depends_on: # Define the task dependency. In this practice, the configuration indicates that Job3
depends on Job1 and Job2.
        - Job1
        - Job2
      build_ref: .cloudbuild/build_job3.yml # Define the YAML build script to run for Job3.
    - job: Job1
      build_ref: .cloudbuild/build_job1.yml # Define the YAML build script to run for Job1.
    - job: Job2
      build_ref: .cloudbuild/build_job2.yml # Define the YAML build script to run for Job2.
```

build.yml defines the entire build process, with three build tasks currently specified. **Job3** depends on **Job1** and **Job2** and will be run only after they are completed. **Job1** and **Job2** have equal priority and will be triggered at the same time. **build_ref** defines the build script to be run for each build task.

----End

Creating Task Scripts Used in build.yml

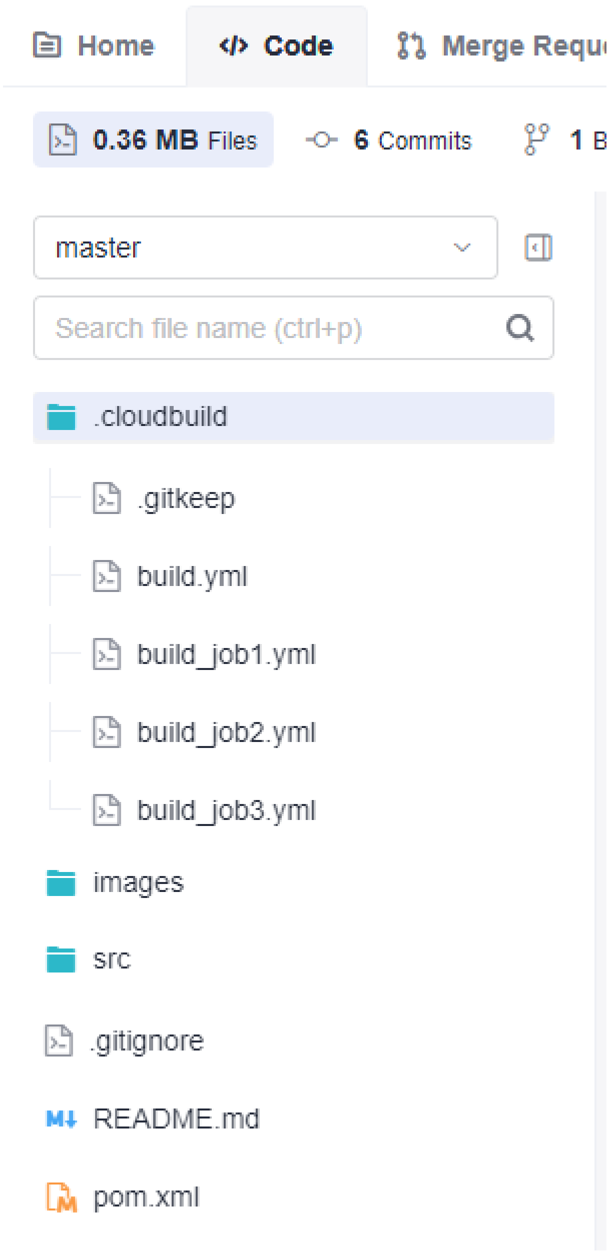
Step 1 In the **.cloudbuild** directory, click **Create** and select **Create File** from the drop-down list. Name the file as **build_job1.yml** and add the following code to the file:

```
version: 2.0
steps:
  BUILD:
  - maven:
      image: cloudbuild@maven3.5.3-jdk8-open # The image is used for your build and can be customized.
      inputs:
        settings:
          public_repos:
            - https://mirrors.huawei.com/maven # Configure the dependency repository.
        cache: true # Determine whether to enable caching.
        command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B # The command to be run.
```

Step 2 Create **build_job2.yml** and **build_job3.yml** with the same code samples by following the instructions in **Step 1**.

Step 3 The following figure shows the directory that stores files of the code repository.

Figure 7-1 Directory



.cloudbuild
.gitkeep
build.yml
build_job1.yml
build_job2.yml
build_job3.yml
images
src
.gitignore
README.md
pom.xml

----End

Creating and Running a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 7-2**.

Table 7-2 Basic information

Parameter	Description
Name	Assign a custom name to the build task, for example, **BuildFlow**.

Parameter	Description
Code Source	Select **Repo**.
Repository	Select **Repo01**, the code repository created in **Creating a CodeArts Repo Repository**.
Default Branch	Keep the default value **master**.
Description	Enter additional information to describe the build task.

Step 3 Click **Next** and select the **Maven** template. Click **OK**. The **Build Actions** page is displayed.

Step 4 Click the **Code** tab. The build scripts in the **Repo01** code repository will be loaded automatically.

Step 5 Click **Save and Run**. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

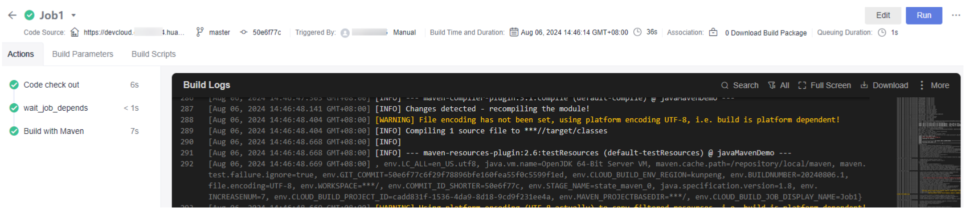
----End

Viewing the Build Results

The **Build Processes** tab page shows the complete flowchart of the current build tasks. While the build tasks are still running, you can see that Job1 and Job2 are running in parallel, and Job3 only runs after Job1 and Job2 are finished.

Step 1 On the **Build Processes** tab page, click either the **Job1** node in the left-hand menu or the green **Job1** rectangle in the right pane. The run details page of **Job1** is displayed, where you can view the build logs of **Job1**, as shown in **Figure 7-2**.

Figure 7-2 Viewing the build results



Code Source: https://devcloud.[redacted]4.hua... master 50e6f77c Triggered By: [redacted] Manual Build Time and Duration: Aug 06, 2024 14:46:14 GMT+08:00 36s Association: 0 Download Build Package Queuing Duration: 1s

Code check out 6s
wait_job_depends < 1s
Build with Maven 7s

```
[Aug 06, 2024 14:46:48.141 GMT+08:00] [INFO] Changes detected - recompiling the module!
[Aug 06, 2024 14:46:48.404 GMT+08:00] [WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[Aug 06, 2024 14:46:48.404 GMT+08:00] [INFO] Compiling 1 source file to ***//target/classes
[Aug 06, 2024 14:46:48.668 GMT+08:00] [INFO]
[Aug 06, 2024 14:46:48.668 GMT+08:00] [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ javaMavenDemo ---
[Aug 06, 2024 14:46:48.669 GMT+08:00] , env.LC_ALL=en_US.utf8, java.vm.name=OpenJDK 64-Bit Server VM, maven.cache.path=/repository/local/maven, maven.
test.failure.ignore=true, env.GIT_COMMIT=50e6f77c6f29f78896bfe160fea55f0c5599f1ed, env.CLOUD_BUILD_ENV_REGION=kunpeng, env.BUILDNUMBER=20240806.1,
file.encoding=UTF-8, env.WORKSPACE=***/, env.COMMIT_ID_SHORTER=50e6f77c, env.STAGE_NAME=state_maven_0, java.specification.version=1.8, env.
INCREASENUM=7, env.CLOUD_BUILD_PROJECT_ID=cadd831f-1536-4da9-8d18-9cd9f231ee4a, env.MAVEN_PROJECTBASEDIR=***/, env.CLOUD_BUILD_JOB_DISPLAY_NAME=Job1}
```

The details page includes the following tabs:

- The **Actions** tab shows the resource scheduling information and sequential position of this task within the whole build process.
- The **Build Parameters** tab shows the global parameters of the current build task.
- The **Build Scripts** tab shows the script run for the current build task.

Step 2 You can view the run details of **Job2** and **Job3** by following instructions in **Step 1**.

----End

8 Using the File from the Self-hosted Repo to Build with Maven and Uploading the Resulting Software Package (Built-in Executors, GUI)

Scenario

CodeArts Build provides default dependency repositories. If they fall short of your service needs, you can create custom repositories for Maven builds.

These steps depend on the following services:

- **CodeArts Repo**: stores your project code.
- **CodeArts Artifact**: stores your private dependencies.

Requirements

- You have permissions for CodeArts Artifact.
- You have permissions for CodeArts Repo.

Procedure

Table 8-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a Self-hosted Repo	Create a self-hosted repo.
Checking Information About the Self-hosted Repo	Check the ID and URL of the self-hosted repo, which will be used for configuring the **pom.xml** file of the code repository.

Step	Description
Uploading the settings.xml File to CodeArts Build	Upload the **settings.xml** file to **Files** for management in CodeArts Build.
Creating a CodeArts Repo Repository	Create a code repository.
Configuring the Self-hosted Repo Address for Storing Maven Build Products	Configure the address of the self-hosted repo where the build product will be uploaded.
Creating a Build Task	Create a build task.
Configuring Build Actions and Running the Build Task	Configure actions **Download File from File Manager** and **Build with Maven**, and run the build task.
Viewing the Build Results	View the build results stored in the self-hosted repo.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a Self-hosted Repo

Step 1 In the navigation pane, choose **Artifact** > **Self-hosted Repos**.

Step 2 Click **Create** and set parameters according to the following table.

Table 8-2 Parameters for creating a self-hosted repo

Parameter	Description
Repository Type	Select **Local Repository**.
Repository Name	Assign a custom name to the repository, for example, **private_repository**.
Package Type	Select **Maven**.
Project	Leave the value as default. The field is autofilled with **build-bestpractice** in this practice.
Include Patterns	Leave it blank.
Version Policy	Specify the version of artifacts stored in the repository. Select either **Release** (the release version with stable functions) or **Snapshot** (the development version with unstable functions). You can also select both of them if needed. In this practice, select **Release**.
Description	Enter additional information to describe the repository. Use no more than 200 characters.

Step 3 Click **Submit**. The details page of **private_repository** is displayed.

----End

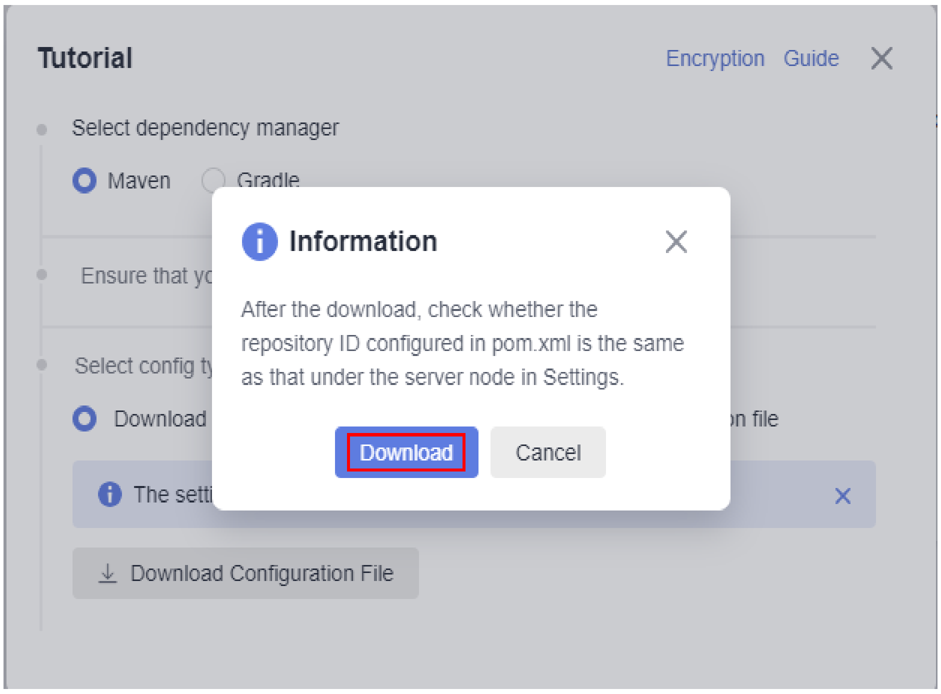
Checking Information About the Self-hosted Repo

Step 1 Click **Tutorial** in the upper right corner.

Step 2 In the displayed dialog box, click **Download Configuration File**, leaving the settings as default.

Step 3 In the dialog box that appears next, click **Download**.

Figure 8-1 Downloading the configuration file

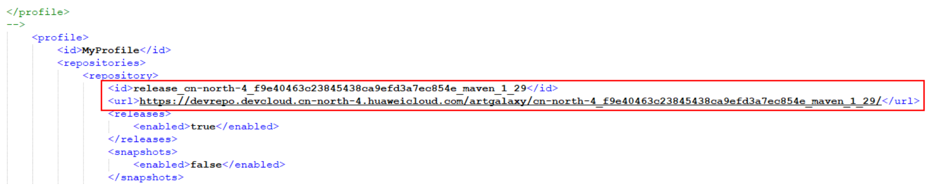


Information

After the download, check whether the repository ID configured in pom.xml is the same as that under the server node in Settings.

Step 4 Open the downloaded **settings.xml** file and navigate to the **<profile>** section that includes the **<repositories>** element. Under this element, find the **<repository>** block that contains information about the repository. Record the values of **id** and **url**.

Figure 8-2 Checking the repository ID and URL



```
</profile>
-->
    <profile>
        <id>MyProfile</id>
        <repositories>
            <repository>
                <id>release_cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29</id>
                <url>https://devrepo.devcloud.cn-north-4.huaweicloud.com/artgalaxy/cn-north-4_f9e40463c23845438ca9efd3a7ec854e_maven_1_29/</url>
                <releases>
                    <enabled>true</enabled>
                </releases>
                <snapshots>
                    <enabled>false</enabled>
                </snapshots>
```

----End

Uploading the settings.xml File to CodeArts Build

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 On the build task list page, choose **More** > **Files**.

Step 3 On the displayed page, click **Upload File**.

Step 4 In the displayed dialog box, upload the **settings.xml** file you download when **checking information about the self-hosted repo**. Select the check box to confirm that you have read and accept the agreements. Then click **Save**.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

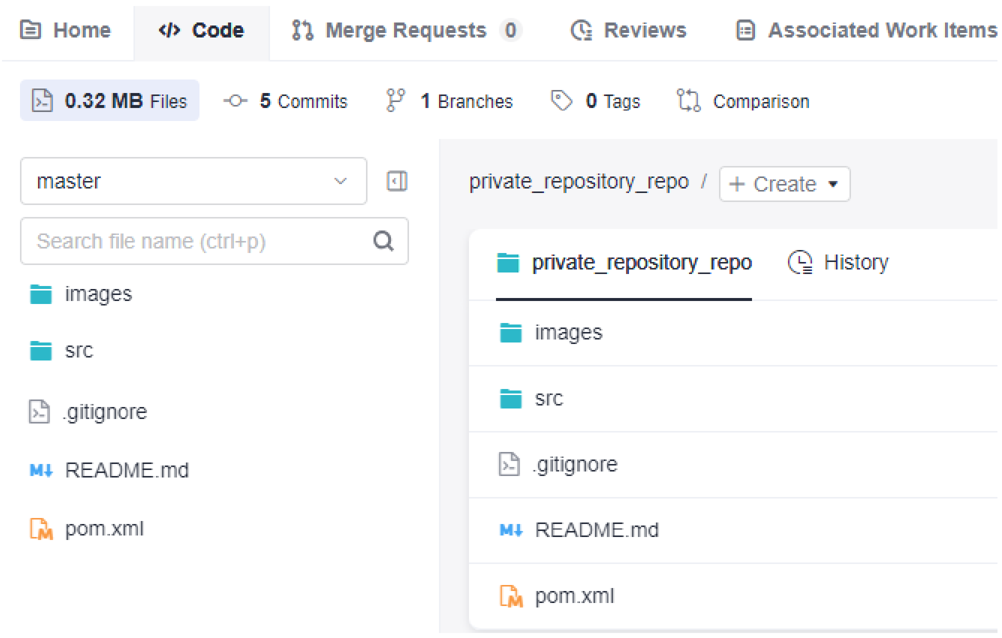
Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the repository creation page, type **private_repository_repo** in the **Repository Name** field, and leave the other parameters as default.

Step 5 Click **OK**. Figure 8-3 shows the directory that stores files of the code repository.

Figure 8-3 Directory



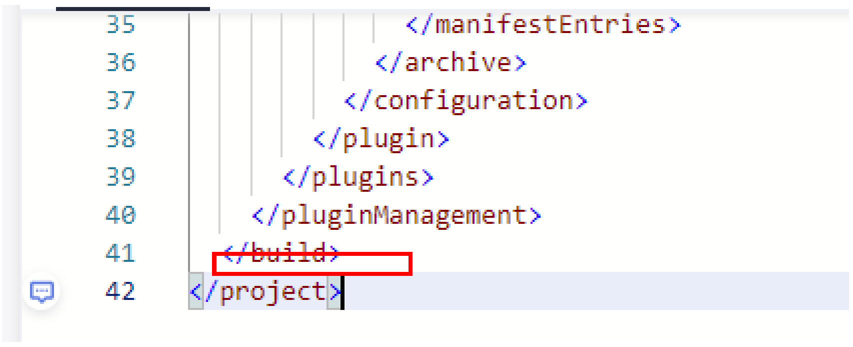
----End

Configuring the Self-hosted Repo Address for Storing Maven Build Products

Step 1 Click the **pom.xml** file to enter its details page. In the right pane, Click ✎ to access the file editor.

Step 2 Copy the following sample code and add it under the **build** tag.

Figure 8-4 Sample code of the pom.xml file



```
            </manifestEntries>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
</project>
```

```
<distributionManagement>
    <repository>
    <id>ID</id>
    <url>https://example/</url>
  </repository>
</distributionManagement>
```

Replace the placeholders of **id** and **url** (**ID** and **https://example/**) with the values of **id** and **url** recorded in **Step 4**.

Step 3 Click **OK**.

----End

Creating a Build Task

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.

- **Name**: Assign a custom name to the build task, for example, **private_repository_task**.
- **Code Source**: Select **Repo**.
- **Repository**: Select **private_repository_repo**, the code repository created in **Creating a CodeArts Repo Repository**.

Step 3 Click **Next**. On the displayed page, select **Blank Template**.

Step 4 Click **OK**. The **Build Actions** page is displayed.

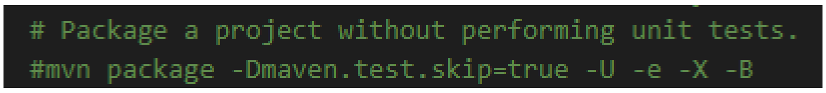
----End

Configuring Build Actions and Running the Build Task

Step 1 Click **Add Action**. Add action **Download File from File Manager**. Leave **Action Name** and **Tool Version** as default. Expand the **File Name** drop-down list and select the **settings.xml file uploaded to CodeArts Build**.

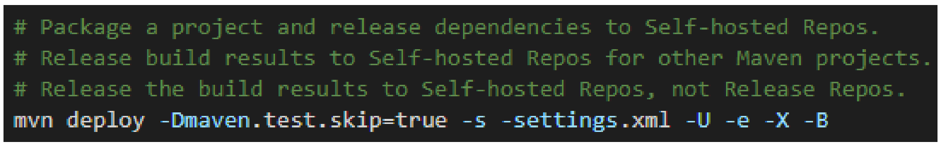
Step 2 Click **Add Action**. Add action **Build with Maven**. In the **Commands** window, add a number sign (#) before the **mvn package -Dmaven.test.skip=true -U -e -X -B** command, delete the number sign before **mvn deploy -Dmaven.test.skip=true -U -e -X -B**, and change **mvn deploy -Dmaven.test.skip=true -U -e -X -B** to **mvn deploy -Dmaven.test.skip=true -s settings.xml -U -e -X -B**. Leave the other parameters as default.

Figure 8-5 Command for packaging



```
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B
```

Figure 8-6 Command for releasing the dependencies



```
# Package a project and release dependencies to Self-hosted Repos.
# Release build results to Self-hosted Repos for other Maven projects.
# Release the build results to Self-hosted Repos, not Release Repos.
mvn deploy -Dmaven.test.skip=true -s -settings.xml -U -e -X -B
```

Step 3 Click **Save and Run**. In the displayed dialog box, click **Confirm**. Wait for the build task to complete.

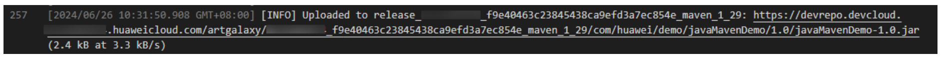
----End

Viewing the Build Results

Step 1 Click the build task name (**private_repository_task** is used in this practice).

Step 2 On the **Build History** tab page, click the build ID. On the displayed page, check the action logs. **com/huawei/demo/javaMavenDemo/1.0** is the path of the uploaded build product in the self-hosted repo **private_repository**.

Figure 8-7 Path for uploading dependencies

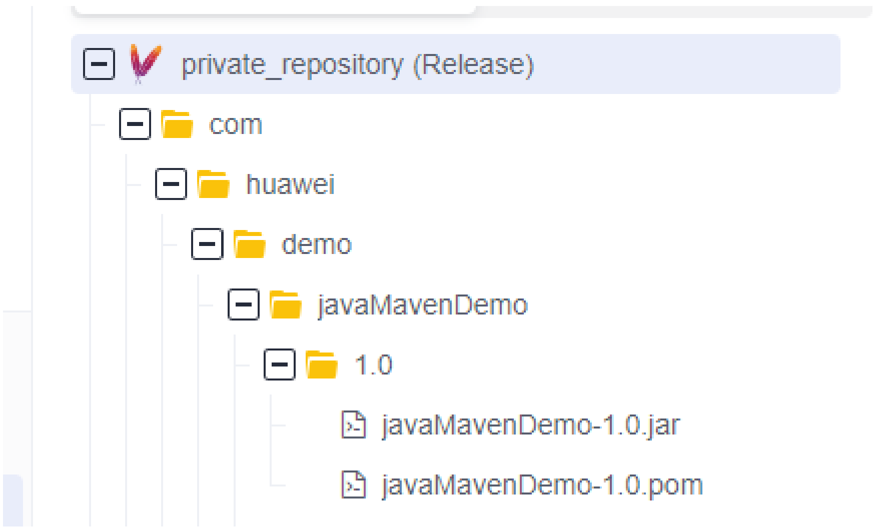


```
[2024/06/26 10:31:50.908 GMT+08:00] [INFO] Uploaded to release_[redacted]_f9e40463c23845438ca9efd3a7ec854e_maven_1_29: https://devrepo.devcloud.
[redacted].huaweicloud.com/artgalaxy/[redacted]_f9e40463c23845438ca9efd3a7ec854e_maven_1_29/com/huawei/demo/javaMavenDemo/1.0/javaMavenDemo-1.0.jar
(2.4 kB at 3.3 kB/s)
```

Step 3 In the navigation pane, choose **Artifact** > **Self-hosted Repos**.

Step 4 Expand **private_repository** and its subfolders. Find the uploaded software package at **com/huawei/demo/javaMavenDemo/1.0**.

Figure 8-8 Checking the uploaded the software package



----End

9 Running a Build Task on a Custom Environment (Built-in Executors, GUI)

Scenario

You may need to extend CodeArts Build with custom environments in the following scenarios:

- CodeArts Build uses Java 1.8 by default, but you may require Java 21.
- Your build may require your enterprise's dedicated tools that are not supported by CodeArts Build.

In this practice, you customize a build environment to run a build task.

Requirements

- You have **created an organization** named **hwstaff_codeci_gray** in SWR.
  - You have permissions for CodeArts Artifact.
  - You have permissions for CodeArts Repo.

Procedure

Table 9-1 Steps

Step	Description
Creating a Project	Create a project.
Creating a CodeArts Repo Repository	Create the code file used for your build.
Creating an Image as the Custom Build Environment	Create an image as the custom environment.
Creating and Running a Build Task	Create, configure, and run a build task.

Step	Description
Viewing the Build Results	Check the build logs and result files to verify the build results.

Creating a Project

Step 1 Log in to the Huawei Cloud console with your Huawei Cloud account.

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

Step 3 Click **Access Service**. The homepage of CodeArts is displayed.

Step 4 Click **Create Project**, and select the **Scrum** template.

Step 5 Set the project name to **build-bestpractice**, and leave the other parameters as default.

Step 6 Click **OK** to access the project.

----End

Creating a CodeArts Repo Repository

Step 1 In the navigation pane, choose **Code** > **Repo**.

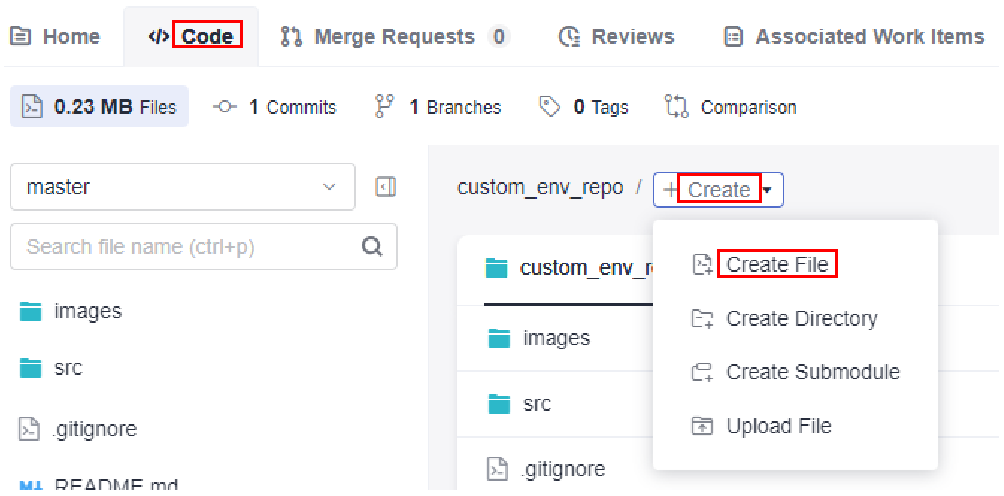
Step 2 On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

Step 3 On the template selection page, select the **Java Maven Demo** template and click **Next**.

Step 4 On the displayed page, set **Repository Name** to **custom_env_repo**, and leave the other parameters as default. Click **OK**. The details page of the new code repository is displayed.

Step 5 In the root directory of the code repository, choose **Create** and select **Create File** from the drop-down list.

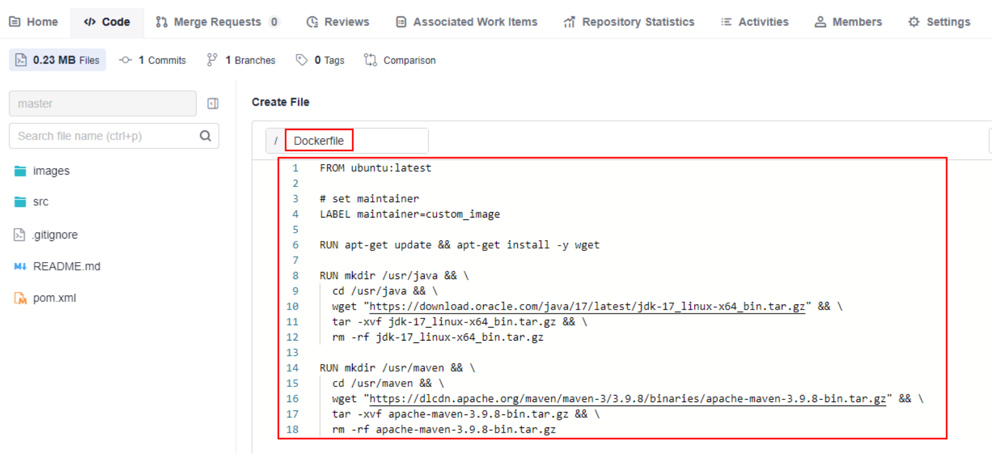
Figure 9-1 Creating a file



Step 6 Name the file **Dockerfile**, copy the following code to the file, and click **Submit**.

```
FROM ubuntu:latest

# set maintainer
LABEL maintainer=custom_image

RUN apt-get update && apt-get install -y wget

RUN mkdir /usr/java && \
  cd /usr/java && \
  wget "https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz" && \
  tar -xvf jdk-17_linux-x64_bin.tar.gz && \
  rm -rf jdk-17_linux-x64_bin.tar.gz

RUN mkdir /usr/maven && \
  cd /usr/maven && \
  wget "https://dlcdn.apache.org/maven/maven-3/3.9.8/binaries/apache-maven-3.9.8-bin.tar.gz" && \
  tar -xvf apache-maven-3.9.8-bin.tar.gz && \
  rm -rf apache-maven-3.9.8-bin.tar.gz

ENV JAVA_HOME /usr/java/jdk-17.0.12
ENV MAVEN_HOME /usr/maven/apache-maven-3.9.8
ENV PATH $PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin

RUN java -version && mvn -v

USER build
```

Figure 9-2 Dockerfile content



```
FROM ubuntu:latest

# set maintainer
LABEL maintainer=custom_image

RUN apt-get update && apt-get install -y wget

RUN mkdir /usr/java && \
  cd /usr/java && \
  wget "https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz" && \
  tar -xvf jdk-17_linux-x64_bin.tar.gz && \
  rm -rf jdk-17_linux-x64_bin.tar.gz

RUN mkdir /usr/maven && \
  cd /usr/maven && \
  wget "https://dlcdn.apache.org/maven/maven-3/3.9.8/binaries/apache-maven-3.9.8-bin.tar.gz" && \
  tar -xvf apache-maven-3.9.8-bin.tar.gz && \
  rm -rf apache-maven-3.9.8-bin.tar.gz
```

Dockerfile contains the following instructions for building a container image as your custom environment.

Table 9-2 Dockerfile instructions

Instruction	Description
FROM	It specifies the base image (the latest official Ubuntu image is used in this practice) and must be the first instruction.
LABEL	It adds metadata to an image.

Instruction	Description
RUN	They are run when **docker build** creates an image. In this practice, **RUN** instructions install three tools: Wget, Java Development Kit (JDK) 17, and Maven 3.9.8. After environment variables are set, these instructions run commands to verify the installation of proper JDK and Maven versions.
ENV	They set environment variables. In this practice, **ENV** instructions set environment variables for JDK and Maven and add their directory to the **PATH** environment variable so that users can conveniently run JDK and Maven by invoking shortcut commands.
USER	It sets the user when the container is run. In this practice, the **build** user is set for the container runtime.

----End

Creating an Image as the Custom Build Environment

Step 1 In the navigation pane, choose **CICD** > **Build**.

Step 2 Click **Create Task**. On the displayed **Basic Information** page, set parameters according to **Table 9-3**. Then, click **Next**.

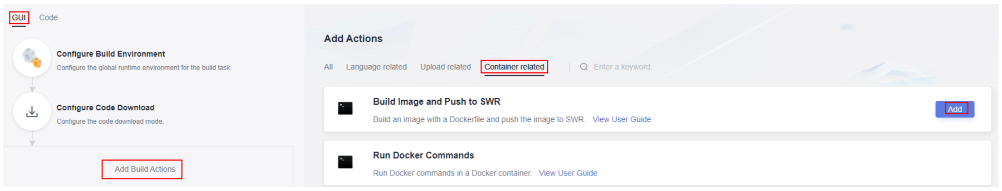
Table 9-3 Basic information

Parameter	Description
Name	Assign a custom name to the build task, for example, **custom_env_task**.
Code Source	Select the code source from which code will be pulled for your build. In this practice, select **Repo**.
Repository	Select **custom_env_repo**, the code repository created in **Creating a CodeArts Repo Repository**.
Default Branch	Keep the default value **master**.

Step 3 Select **Blank Template** and click **OK** to create the build task. The **Build Actions** page is displayed.

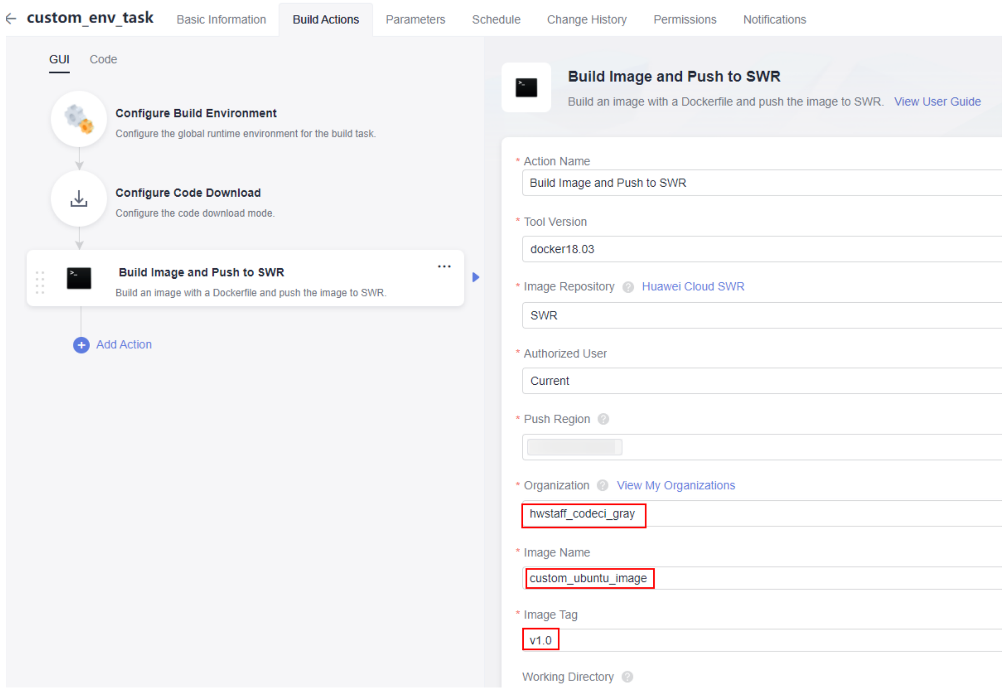
Step 4 On the **Build Actions** page, click the **GUI** tab and click **Add Build Actions**.

Figure 9-3 Adding a build action


GUI
Code
Configure Build Environment
Configure the global runtime environment for the build task.
Configure Code Download
Configure the code download mode.
Add Build Actions
Add Actions
All
Language related
Upload related
Container related
Enter a keyword.
Build Image and Push to SWR
Build an image with a Dockerfile and push the image to SWR.
View User Guide
Add
Run Docker Commands
Run Docker commands in a Docker container.
View User Guide

Step 5 In the right pane, click the **Container related** tab. Hover over action **Build Image and Push to SWR** and click **Add** on the card. Set the parameters according to **Figure 9-4**. Expand the **Organization** drop-down list and select **hwstaff_codeci_gray** created to meet the **requirements**. Type **custom_ubuntu_image** in the **Image Name** text box, and **v1.0** in **Image Tag**. Leave the other parameters as default.

Figure 9-4 Configuring the build action


custom_env_task
Basic Information
Build Actions
Parameters
Schedule
Change History
Permissions
Notifications
GUI
Code
Configure Build Environment
Configure the global runtime environment for the build task.
Configure Code Download
Configure the code download mode.
Build Image and Push to SWR
Build an image with a Dockerfile and push the image to SWR.
Add Action
Build Image and Push to SWR
Build an image with a Dockerfile and push the image to SWR. View User Guide

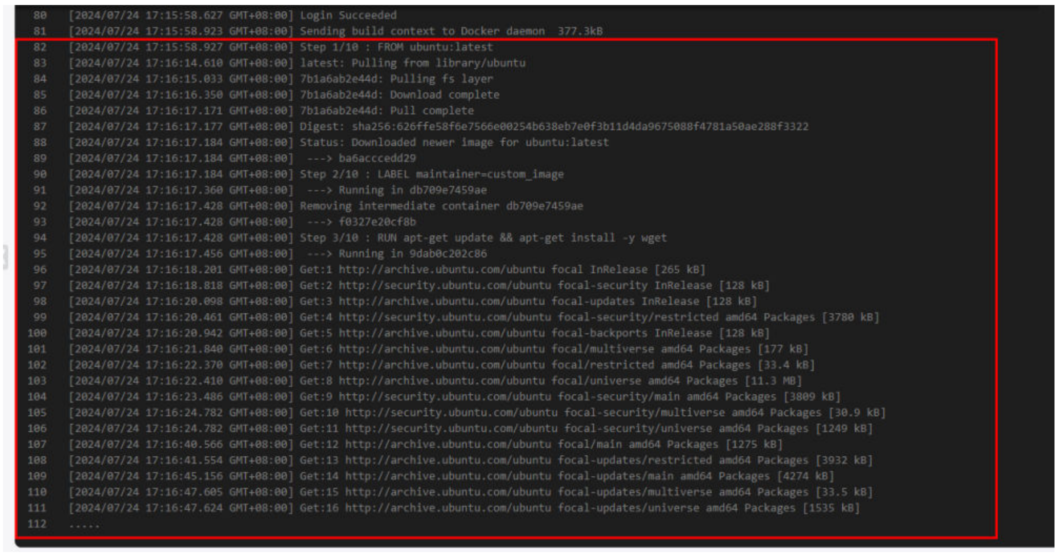
* Action Name: Build Image and Push to SWR
* Tool Version: docker18.03
* Image Repository (Huawei Cloud SWR): SWR
* Authorized User: Current
* Push Region:
* Organization (View My Organizations): hwstaff_codeci_gray
* Image Name: custom_ubuntu_image
* Image Tag: v1.0

Working Directory

Step 6 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

Step 7 On the **Actions** tab, check that the console displays the logs of the build task as it runs. The **Build Logs** window automatically scrolls down to show new entries. As shown in **Figure 9-5**, the console prints logs of creating an image from the **Dockerfile** stored in the code repository.

Figure 9-5 Build task run logs

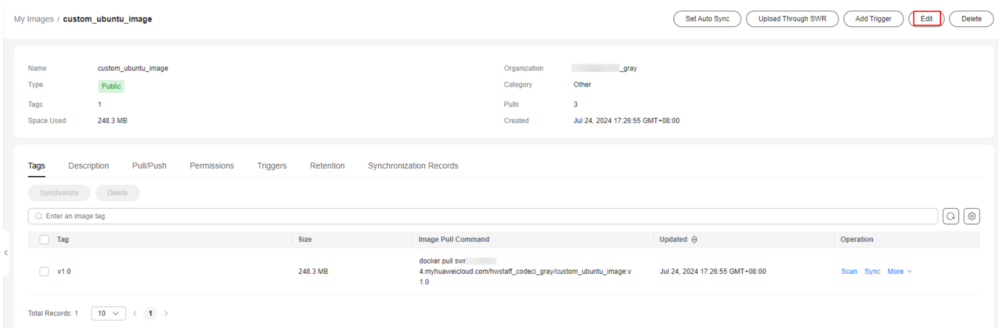


```
[2024/07/24 17:15:58.627 GMT+08:00] Login Succeeded
[2024/07/24 17:15:58.923 GMT+08:00] Sending build context to Docker daemon  377.3kB
[2024/07/24 17:15:58.927 GMT+08:00] Step 1/10 : FROM ubuntu:latest
[2024/07/24 17:16:14.610 GMT+08:00] latest: Pulling from library/ubuntu
[2024/07/24 17:16:15.033 GMT+08:00] 7b1a6ab2e44d: Pulling fs layer
[2024/07/24 17:16:16.350 GMT+08:00] 7b1a6ab2e44d: Download complete
[2024/07/24 17:16:17.171 GMT+08:00] 7b1a6ab2e44d: Pull complete
[2024/07/24 17:16:17.177 GMT+08:00] Digest: sha256:626ffe58f6e7566e00254b638eb7e0f3b11d4da9675088f4781a50ae288f3322
[2024/07/24 17:16:17.184 GMT+08:00] Status: Downloaded newer image for ubuntu:latest
[2024/07/24 17:16:17.184 GMT+08:00]  ---> ba6acccedd29
[2024/07/24 17:16:17.184 GMT+08:00] Step 2/10 : LABEL maintainer=custom_image
[2024/07/24 17:16:17.360 GMT+08:00]  ---> Running in db709e7459ae
[2024/07/24 17:16:17.428 GMT+08:00] Removing intermediate container db709e7459ae
[2024/07/24 17:16:17.428 GMT+08:00]  ---> f0327e20cf8b
[2024/07/24 17:16:17.428 GMT+08:00] Step 3/10 : RUN apt-get update && apt-get install -y wget
[2024/07/24 17:16:17.456 GMT+08:00]  ---> Running in 9dab0c202c86
[2024/07/24 17:16:18.201 GMT+08:00] Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
[2024/07/24 17:16:18.818 GMT+08:00] Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
[2024/07/24 17:16:20.098 GMT+08:00] Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
[2024/07/24 17:16:20.461 GMT+08:00] Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3780 kB]
[2024/07/24 17:16:20.942 GMT+08:00] Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [128 kB]
[2024/07/24 17:16:21.840 GMT+08:00] Get:6 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
[2024/07/24 17:16:22.370 GMT+08:00] Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
[2024/07/24 17:16:22.410 GMT+08:00] Get:8 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
[2024/07/24 17:16:23.486 GMT+08:00] Get:9 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3809 kB]
[2024/07/24 17:16:24.782 GMT+08:00] Get:10 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [30.9 kB]
[2024/07/24 17:16:24.782 GMT+08:00] Get:11 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1249 kB]
[2024/07/24 17:16:40.566 GMT+08:00] Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
[2024/07/24 17:16:41.554 GMT+08:00] Get:13 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3932 kB]
[2024/07/24 17:16:45.156 GMT+08:00] Get:14 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [4274 kB]
[2024/07/24 17:16:47.605 GMT+08:00] Get:15 http://archive.ubuntu.com/ubuntu focal-updates/multiverse amd64 Packages [33.5 kB]
[2024/07/24 17:16:47.624 GMT+08:00] Get:16 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1535 kB]
.....
```

Step 8 After the build task is successfully run, go to the SWR console. Choose **My Images** and click the **Private Images** tab (which is displayed by default). From there, click the name of the image (**custom_ubuntu_image**) created in **Step 5** to access its details page.

Step 9 On the image details page, click **Edit**. In the displayed dialog box, set **Type** to **Public** and click **OK**.

Figure 9-6 Image details



Name	custom_ubuntu_image	Organization	[redacted]_gray
Type	Public	Category	Other
Tags	1	Pulls	3
Space Used	248.3 MB	Created	Jul 24, 2024 17:26:55 GMT+08:00

Tag	Size	Image Pull Command	Updated	Operation
v1.0	248.3 MB	docker pull swr[redacted]4.myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v1.0	Jul 24, 2024 17:26:55 GMT+08:00	Scan Sync More

Total Records: 1 10 < 1 >

Step 10 In the **Image Pull Command** column, copy and save the complete image name following **docker pull** (**swr.{regionID}.myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v1.0** is copied in this practice) for future use. **{regionID}** indicates the ID of the current region.

Figure 9-7 Complete image name

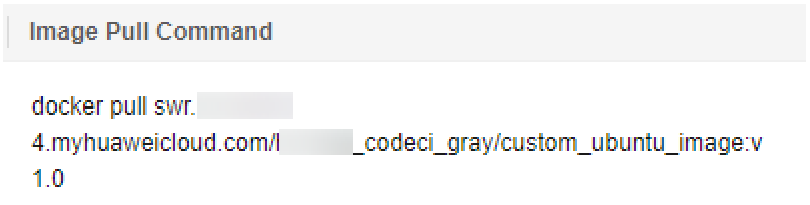


Image Pull Command

docker pull swr.[redacted]
4.myhuaweicloud.com/l[redacted]_codeci_gray/custom_ubuntu_image:v
1.0

----End

Creating and Running a Build Task

Step 1 Access CodeArts Build. Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.

- **Name**: Assign a custom name to the build task, for example, **custom_env_build_task**.
- **Code Source**: Select the code source from which code will be pulled for this build. In this practice, select **Repo**.
- **Repository**: Select **custom_env_repo**, the code repository created in **Creating a CodeArts Repo Repository**.

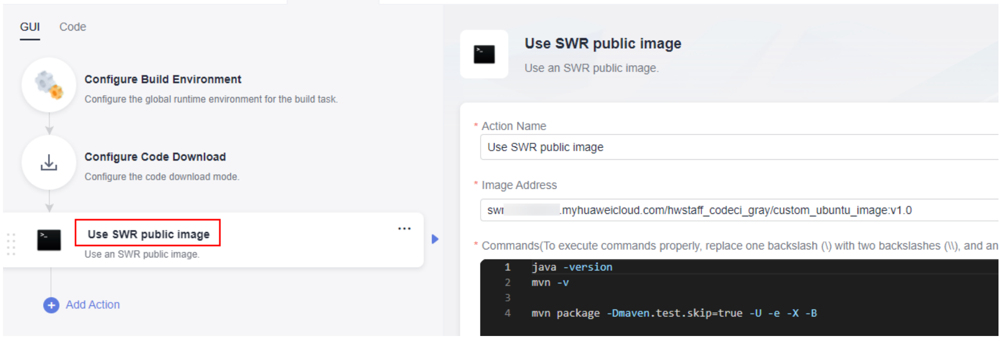
Step 2 Click **Next**. On the displayed page, select **Blank Template**. Click **OK**. The **Build Actions** page is displayed.

Step 3 On the **Build Actions** page, click the **GUI** tab and click **Add Build Actions**.

Step 4 In the right pane, click the **Container related** tab. Hover over action **Use SWR Public Image** and click **Add** on the card. Set the parameters according to **Figure 9-4**. In the **Image Address** text box, enter the complete image name **swr.{regionID}.myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v1.0** saved in **Step 10**. Copy the following sample code to the **Commands** window. Leave the other parameters as default.

```
java -version # Show the JDK version contained in the current image.
mvn -v    # Show the Maven version contained in the current image.
mvn package -Dmaven.test.skip=true -U -e -X -B  # Run the maven build command.
```

Figure 9-8 Configuring the action of using the SWR public image



Configure Build Environment
Configure the global runtime environment for the build task.

Configure Code Download
Configure the code download mode.

Use SWR public image
Use an SWR public image.

Use SWR public image
Use an SWR public image.

* Action Name
Use SWR public image

* Image Address
swr[redacted].myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v1.0

* Commands(To execute commands properly, replace one backslash (\) with two backslashes (\\), and an

```
java -version
mvn -v

mvn package -Dmaven.test.skip=true -U -e -X -B
```

Step 5 Click **Add Action** and add the **Upload to Release Repo** action. In the **Package Location** text box, enter ****/target/*.?ar**. Leave the other parameters as default.

Step 6 Click **Save and Run** in the upper right corner. In the displayed dialog box, click **Confirm**. The build task run page is displayed.

----End

Viewing the Build Results

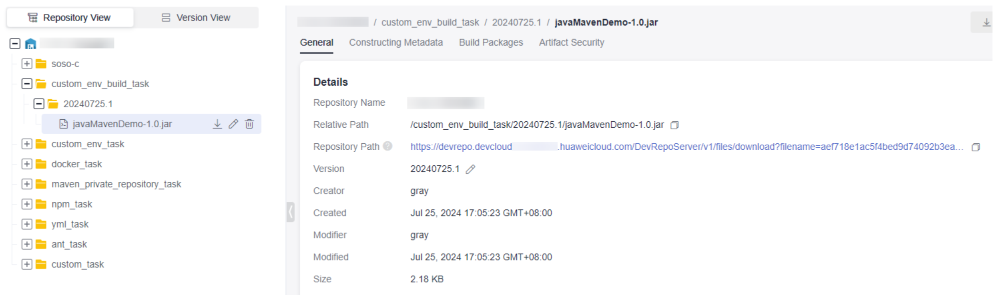
Step 1 After the build task is successfully run, go to the **Actions** tab page. On the top of the **Build Logs** window, click **View More** to scroll the window up and reveal the preceding logs. If you see **Status: Downloaded newer image for swr.{regionID}.myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v1.0** in the logs, the current build environment is created using the custom image.

Step 2 Click the **Upload to Release Repo** action on the left. The right-hand log window will display information about the upload of the build product to the release repo.

Step 3 In the navigation pane, choose **Artifact** > **Release Repos**.

On the displayed page, find the folder that shares the same name as the **build task**, as shown in **Figure 9-9**. The software package can be found within this folder.

Figure 9-9 Checking the software package



Details

Repository Name	
Relative Path	/custom_env_build_task/20240725.1/javaMavenDemo-1.0.jar
Repository Path	https://devrepo.devcloud .huaweicloud.com/DevRepoServer/v1/files/download?filename=aef718e1ac5f4bed9d74092b3ea...
Version	20240725.1
Creator	gray
Created	Jul 25, 2024 17:05:23 GMT+08:00
Modifier	gray
Modified	Jul 25, 2024 17:05:23 GMT+08:00
Size	2.18 KB

----End