

**CodeArts Build**

# **Best Practices**

**Issue** 01  
**Date** 2024-11-30



**Copyright © Huawei Technologies Co., Ltd. 2024. All rights reserved.**

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

## **Trademarks and Permissions**



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Notice**

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

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

# 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)..... 1

2 Building with Maven and Uploading the Software Package to the Self-hosted Repo (Built-in Executors, GUI)..... 6

3 Building with Maven to Generate a Private Dependency for Another Build (Built-in Executors, GUI)..... 12

4 Building with npm and Uploading the Software Package to the Release Repo (Built-in Executors, GUI)..... 21

5 Building with Maven (Custom Executors, GUI)..... 25

6 Building with Maven, Uploading the Software Package, and Pushing the Image to SWR (Built-in Executors, Code)..... 32

7 Running a Multi-Task Maven Build Project (Built-in Executors, Code)..... 38

8 Using the File from the Self-hosted Repo to Build with Maven and Uploading the Resulting Software Package (Built-in Executors, GUI)..... 44

9 Running a Build Task on a Custom Environment (Built-in Executors, GUI)..... 51

# 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 SWR image repository. 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
<a href="#">Creating a Project</a>	Create a project.
<a href="#">Creating a CodeArts Repo Repository</a>	Create the code file used for your build.
<a href="#">Creating a Build Task</a>	Create, configure, and run a build task.
<a href="#">Viewing the Build Results</a>	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 , 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**.

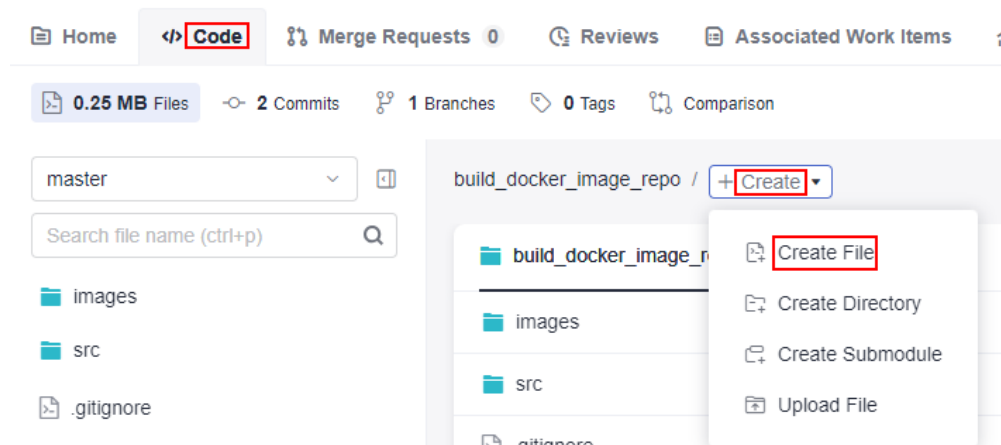
**Step 2** On the displayed page, click **New Repository**. Select **Template**, and click **Next**.

**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

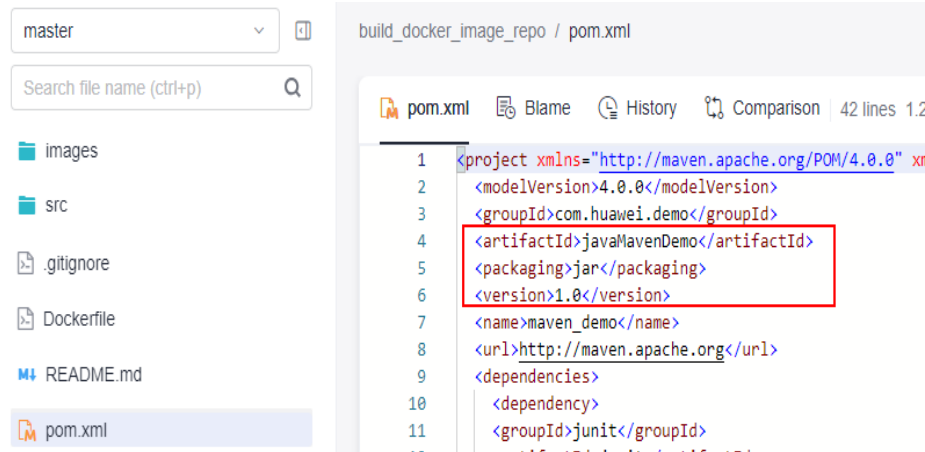
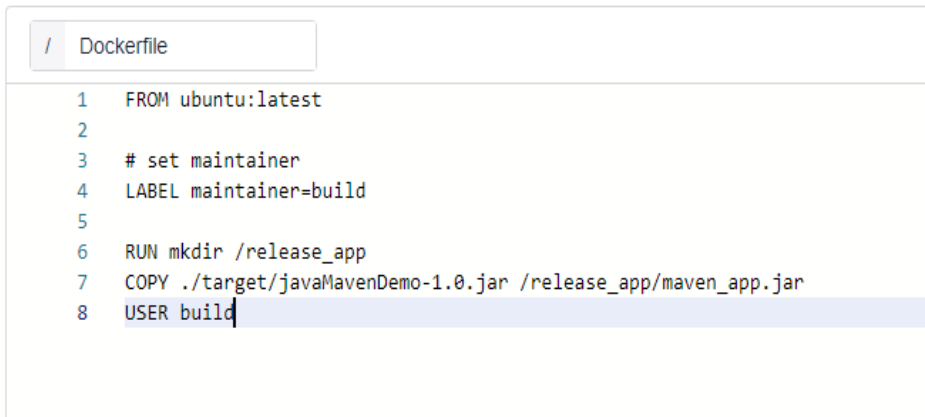


Figure 1-3 Dockerfile content



----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, <b>build_docker_image_task</b> .

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

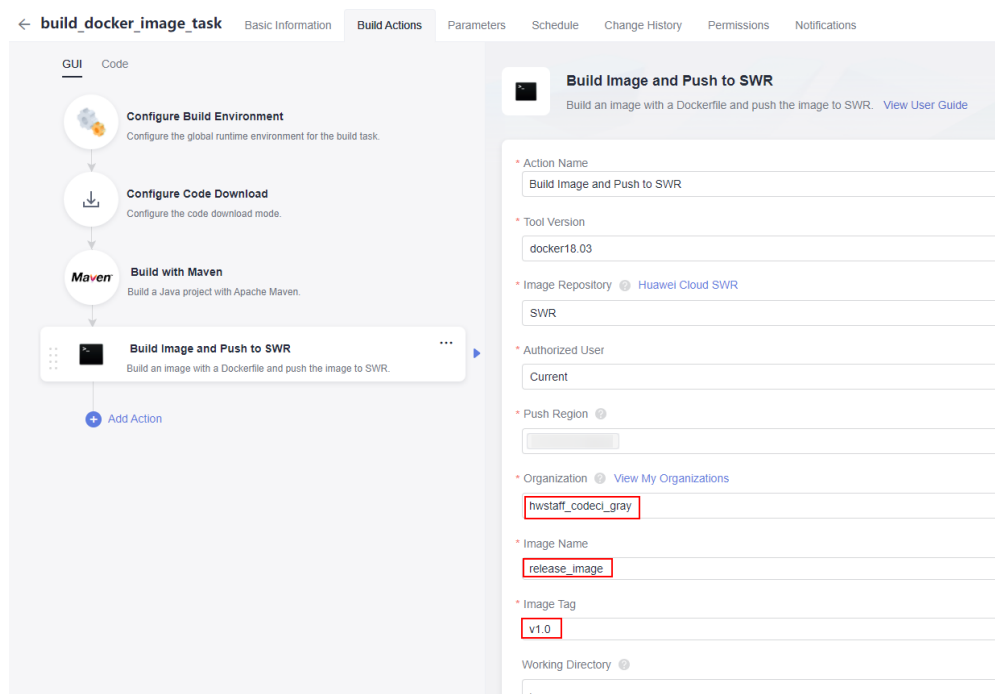
**Step 3** Select **Blank Template** and click **OK** to create the build task. The **Build Actions** page is displayed.

**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



**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

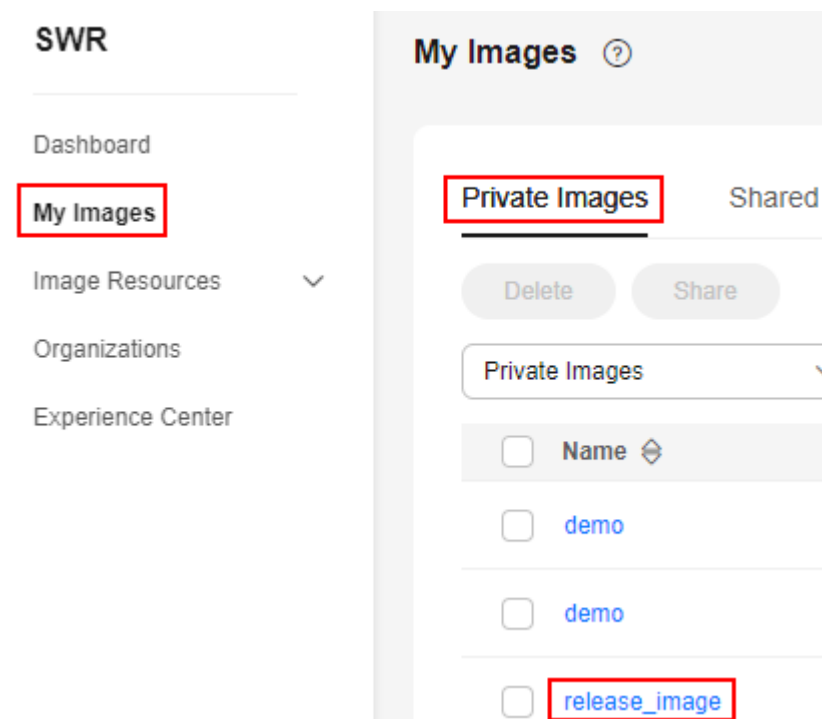
- Step 1** On the **Actions** tab, check that the console displays the logs of the build task as it runs. The **Build Logs** window automatically scrolls down to show new entries. As shown in [Figure 1-5](#), the build log window displays the information about the product of the Maven build.

**Figure 1-5** Information about the Maven build product

```
264 [2024/08/01 19:19:12.430 GMT+08:00] releases: [enabled => true, update => daily]
265 [2024/08/01 19:19:12.430 GMT+08:00] ]
266 [2024/08/01 19:19:12.431 GMT+08:00] [INFO] Tests are skipped.
267 [2024/08/01 19:19:12.431 GMT+08:00] [INFO] --- maven-jar-plugin:2.6:jar (default-jar) @ javaMavenDemo ---
268 [2024/08/01 19:19:12.431 GMT+08:00] [INFO] Building jar: ***/target/javaMavenDemo-1.0.jar
269 [2024/08/01 19:19:12.978 GMT+08:00] [INFO] -----
270 [2024/08/01 19:19:13.248 GMT+08:00] [INFO] BUILD SUCCESS
271 [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 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
<a href="#">Creating a Project</a>	Create a project.
<a href="#">Creating a Self-hosted Repo</a>	Create a self-hosted repo.
<a href="#">Creating a CodeArts Repo Repository</a>	Create a code repository.

Step	Description
<a href="#">Creating and Running a Build Task</a>	Create, configure, and run a build task.
<a href="#">Viewing the Build Results</a>	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 , 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](#).

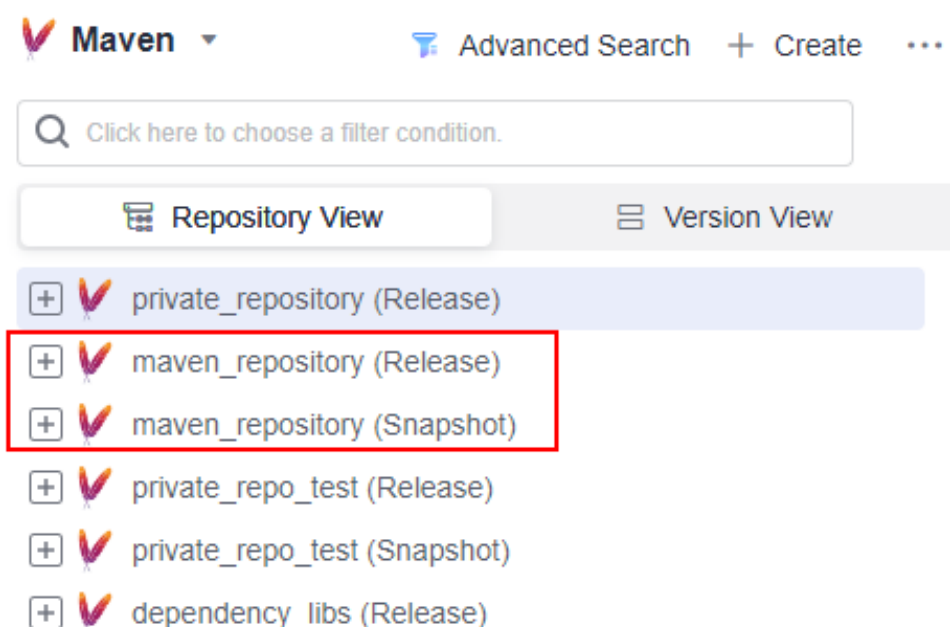
**Table 2-2** Parameters for creating a self-hosted repo

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

Parameter	Description
Version Policy	Specify the version of artifacts stored in the repository. Select either <b>Release</b> (the release version with stable functions) or <b>Snapshot</b> (the development version with unstable functions). You can also select both of them if needed. In this practice, select both <b>Release</b> and <b>Snapshot</b> .
Description	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

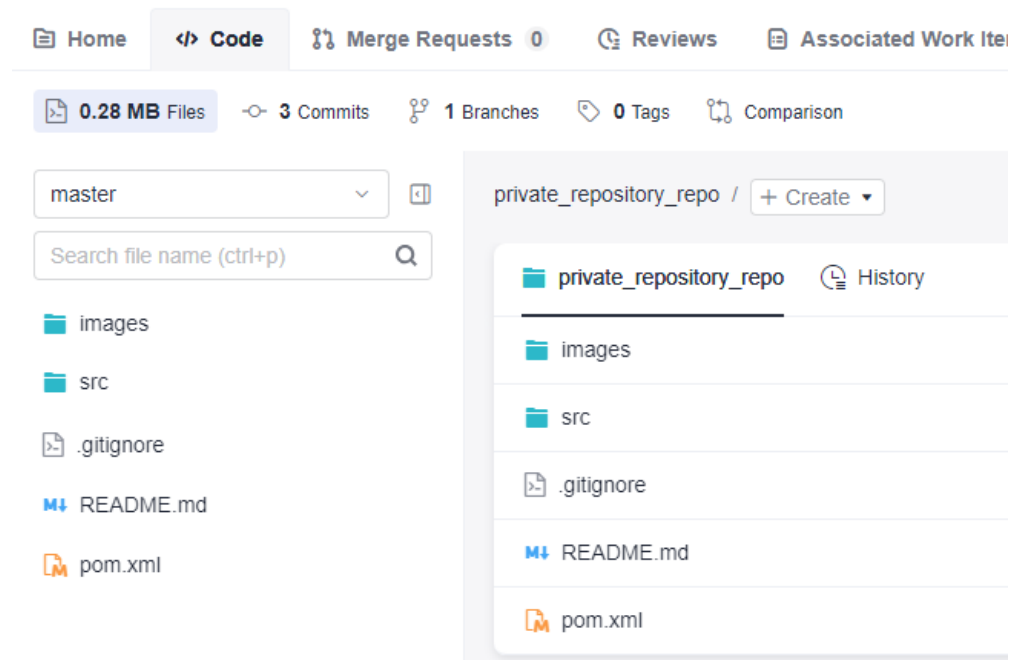


----End

## 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.

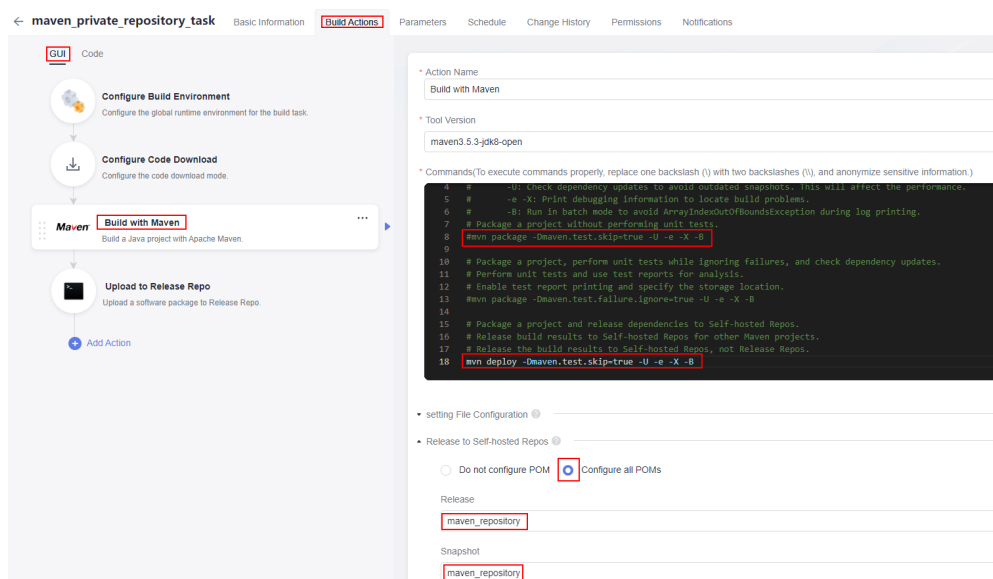
Figure 2-2 Directory



----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](#).

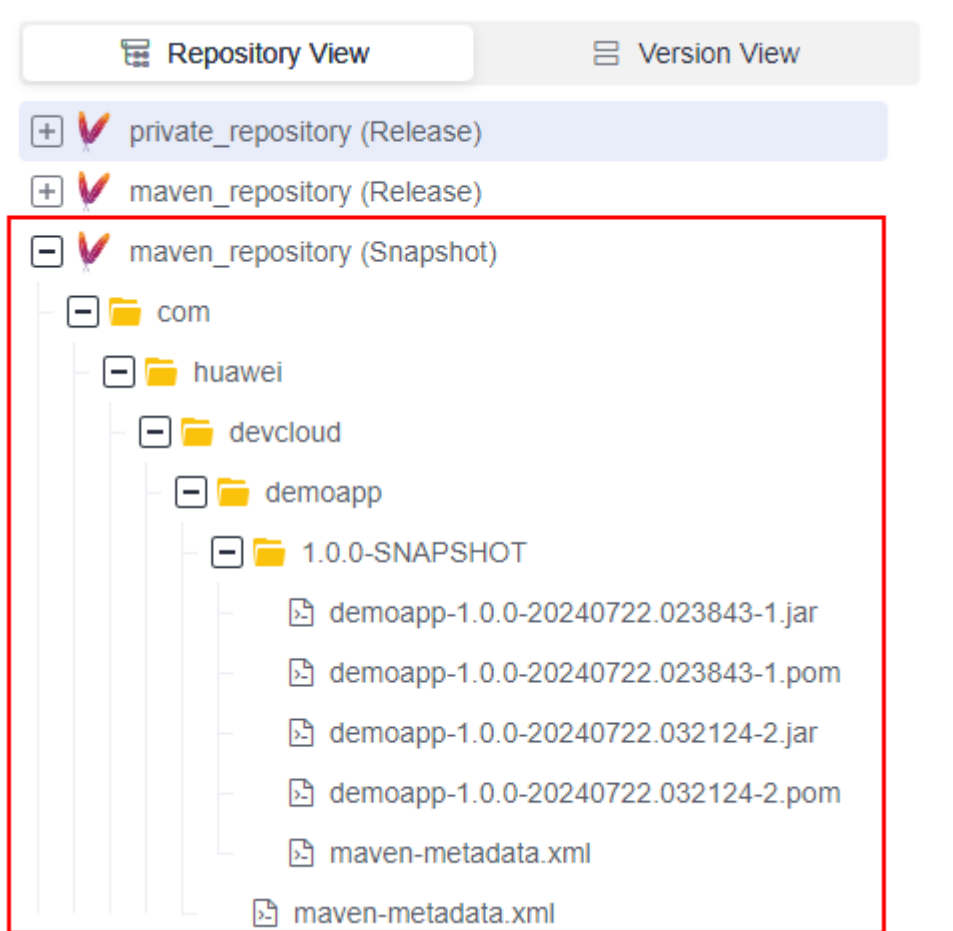
**Figure 2-3** Configuring the action of building with Maven

**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

----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
#codeci_practice_1037/en-us_topic_0000001975247246_section034264112171	Create a project.
<a href="#">Creating a Self-hosted Repo</a>	Create a self-hosted repo.
<a href="#">Creating Code Repository dependency_repo</a>	Create the code repository for releasing the <b>dependencyProject-1.0.jar</b> package.
<a href="#">Creating Build Task dependency_task</a>	Create the build task for releasing the <b>dependencyProject-1.0.jar</b> package.
<a href="#">Creating Code Repository release_repo</a>	Create a code repository for releasing the application.
<a href="#">Creating Build Task release_task</a>	Create the build task that depends on the <b>dependencyProject-1.0.jar</b> package to build and release the application.
<a href="#">Viewing the Build Results</a>	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 , 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.**

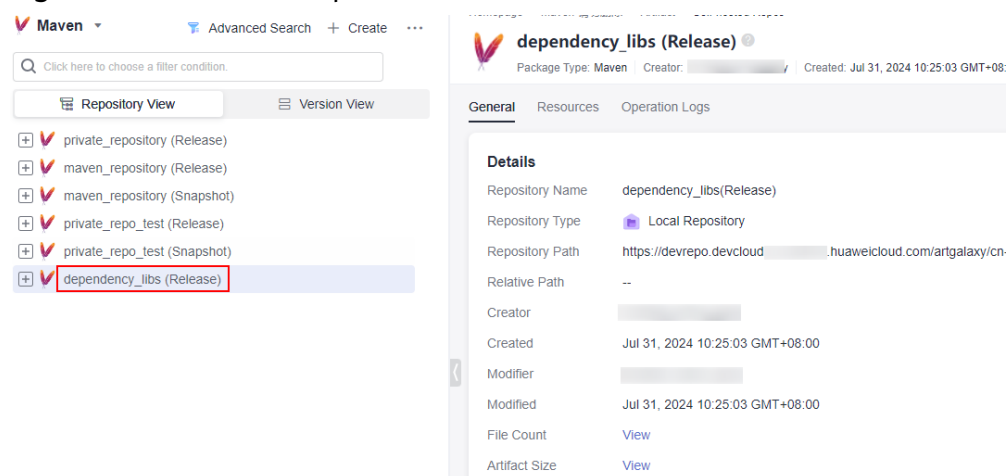
**Step 2** Click **Create** and set parameters according to [Table 3-2](#).

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

Parameter	Description
Repository Type	Select <b>Local Repository</b> .
Repository Name	Assign a custom name to the repository, for example, <b>dependency_libs</b> .
Package Type	Select <b>Maven</b> .
Project	Leave the value as default. The field is autofilled with <b>build-bestpractice</b> in this practice.
Include Patterns	Leave it blank.
Version Policy	Specify the version of artifacts stored in the repository. Select either <b>Release</b> (the release version with stable functions) or <b>Snapshot</b> (the development version with unstable functions). You can also select both of them if needed. In this practice, select <b>Release</b> .
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

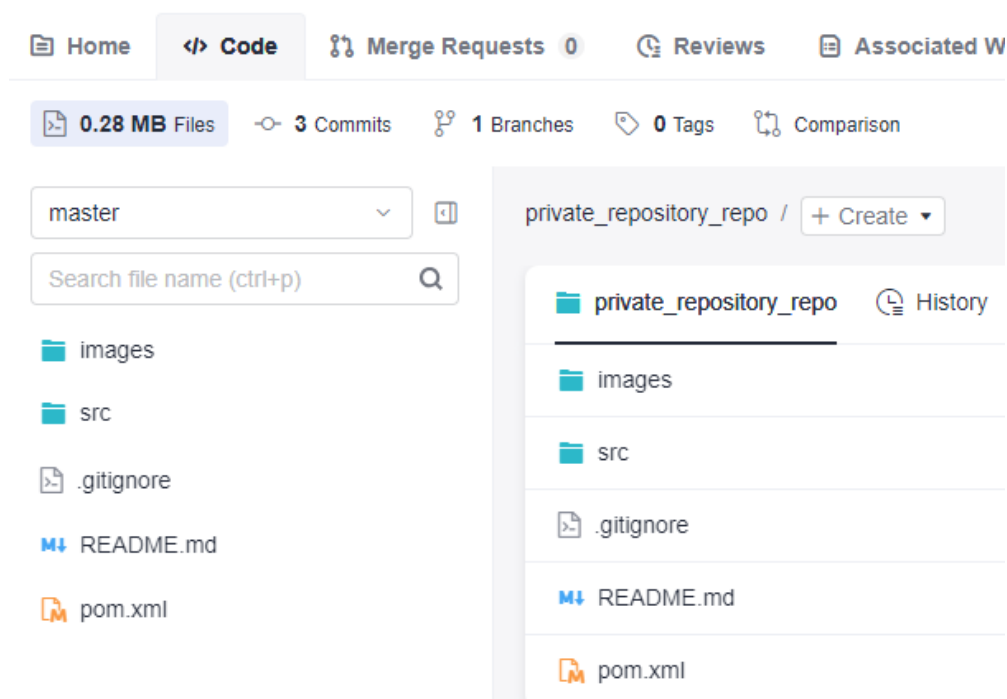



----End

## Creating Code Repository `dependency_repo`

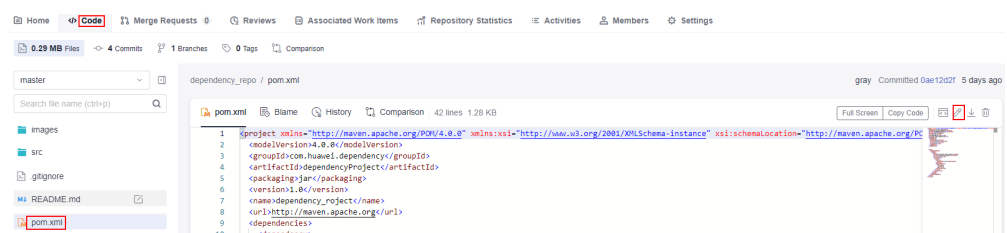
- Step 1** In the navigation pane, choose **Code > Repo**.
- Step 2** On the displayed page, click **New Repository**. Select **Template**, and click **Next**.
- Step 3** On the template selection page, select the **Java Maven Demo** template and click **Next**.
- Step 4** On the repository creation 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



- 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

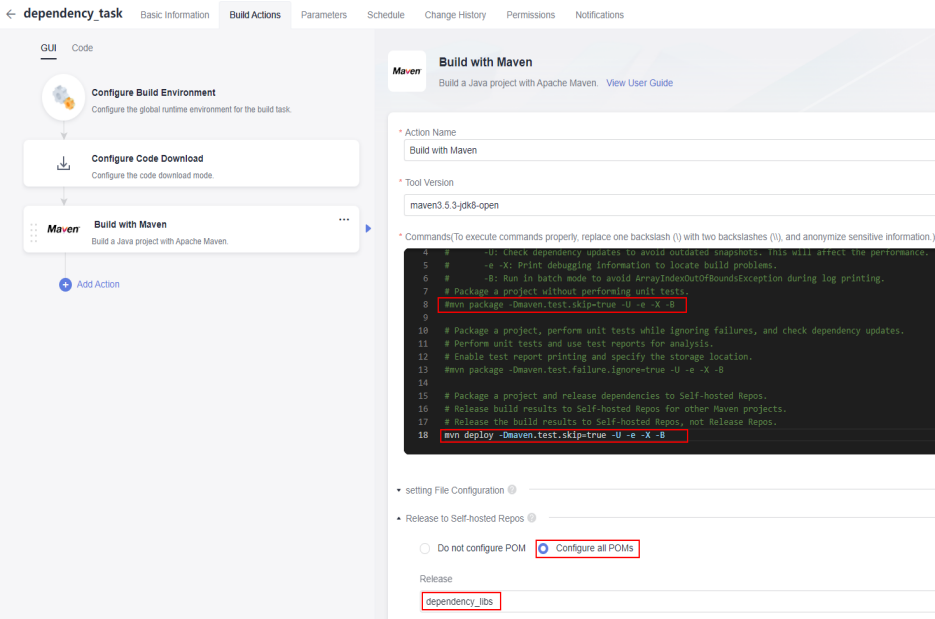
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>com.huawei.dependency</groupId>
4 <artifactId>dependencyProject</artifactId>
5 <packaging>jar</packaging>
6 <version>1.0</version>
7 <name>dependency_project</name>
8 <url>http://maven.apache.org</url>
9 <dependencies>
10 <dependency>
11 <groupId>junit</groupId>
12 <artifactId>junit</artifactId>
13 <version>3.8.1</version>
14 <scope>test</scope>
15 </dependency>
16 </dependencies>
```

----End

## Creating Build Task `dependency_task`

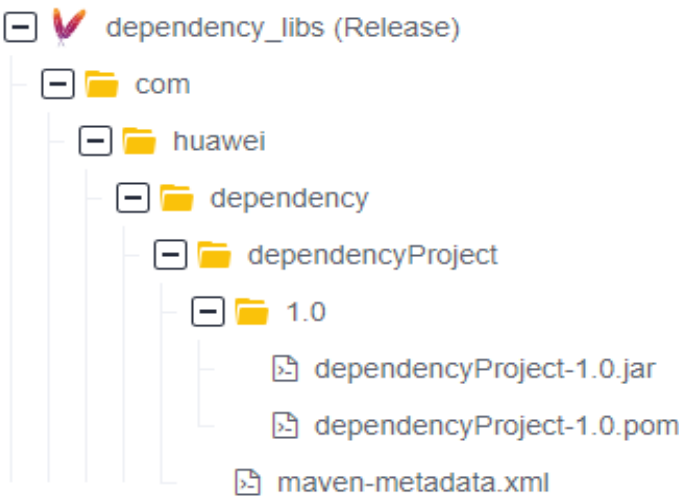
- Step 1** In the navigation pane, choose **CICD > Build**.
- Step 2** Click **Create Task**. On the displayed **Basic Information** page, set the following parameters. Leave the other parameters as default.
- **Name:** Assign a custom name to the build task, for example, **private\_repository\_task**.
  - **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



- 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



----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

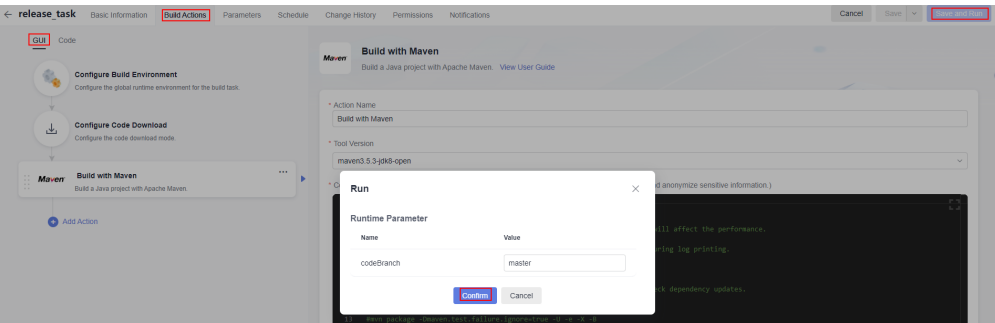
```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/20
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>com.huawei.release</groupId>
4 <artifactId>releaseProject</artifactId>
5 <packaging>jar</packaging>
6 <version>1.0</version>
7 <name>release_roject</name>
8 <url>http://maven.apache.org</url>
9 <dependencies>
10 <dependency>
11 <groupId>junit</groupId>
12 <artifactId>junit</artifactId>
13 <version>3.8.1</version>
14 <scope>test</scope>
15
16 </dependency>
17 <dependency>
18 <groupId>com.huawei.dependency</groupId>
19 <artifactId>dependencyProject</artifactId>
20 <version>1.0</version>
21 </dependency>
22 </dependencies>
23
```

----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

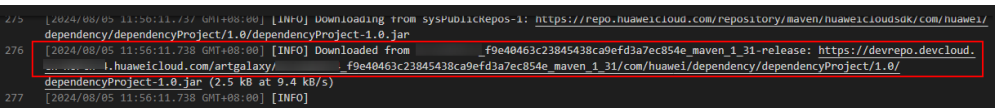


----End

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





# 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
#codeci_practice_1003/en-us_topic_0000001947776221_section1885415962013	Create a project.
<a href="#">Creating a CodeArts Repo Repository</a>	Create a CodeArts Repo repository.

Step	Description
<a href="#">Creating a Build Task</a>	Create a build task.
<a href="#">Configuring Build Actions and Running the Build Task</a>	Configure build actions and run the build task.
<a href="#">Viewing and Verifying the Build Results</a>	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 , 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, <b>npm_yaml_build</b> .
Code Source	Select <b>Repo</b> .
Repository	Select <b>nodesource</b> , the code repository created in <a href="#">Creating a CodeArts Repo Repository</a> .
Default Branch	Keep the default value <b>master</b> .
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 ./nodelserver.zip ./** command to pack the code into **nodelserver.zip**, as shown in [Figure 4-1](#). Leave the other parameters as default.

**Figure 4-1** Command example

```
23 #npm run build
24 zip -r ./nodelserver.zip ./
25 #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 ?

./nodereserver.zip

Version ?

v0.0.1

Package Name ?

nodereserver

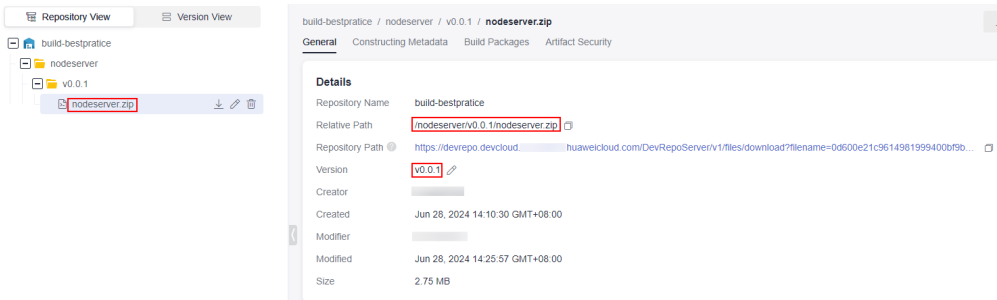
**Step 3** Once you have finished configuring all build actions, click **Save and Run** to run the build task.

-----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



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
#codeci_practice_1001/en-us_topic_0000001917657144_section034264112171	Create a project.

Step	Description
Creating an Agent Pool	Create a pool of custom executors (agent pool).
Creating a CodeArts Repo Repository	Create a repository to store code.
Creating and Running a Build Task	Create a build task that includes actions <b>Build with Maven</b> and <b>Upload to Release Repo</b> .
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 , 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, <b>custom_pool</b> .

Parameter	Description
Pool Type	Select <b>LINUX_DOCKER</b> . When a task is initiated, a Linux Docker container will be started to run the task.
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 .
SK	Obtain the SK by referring to .
Agent Name	Assign a custom name to the agent, for example, <b>agent_test_custom</b> .
Agent Workspace	Enter an agent workspace that follows the standard Linux directory structure. For example, <b>/opt/agent_test_custom</b> .

**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

**Create Agent** [Help](#) ×

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

Install a JDK automatically ☐ [How to Manually Install Java 8?](#)

Install Git automatically ☐ [How to Manually Install Git?](#)

☒ **Install Docker** [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

\* SK

\* Agent Name

\* Agent Workspace

☒ 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.

[Generate Command](#) [Copy Command](#)

```
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 9a8bf23d0f2495682f37ed9b0e752c -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.

[Close](#)

**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-mwlye1N1LG**.

----End

## Creating a CodeArts Repo Repository

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

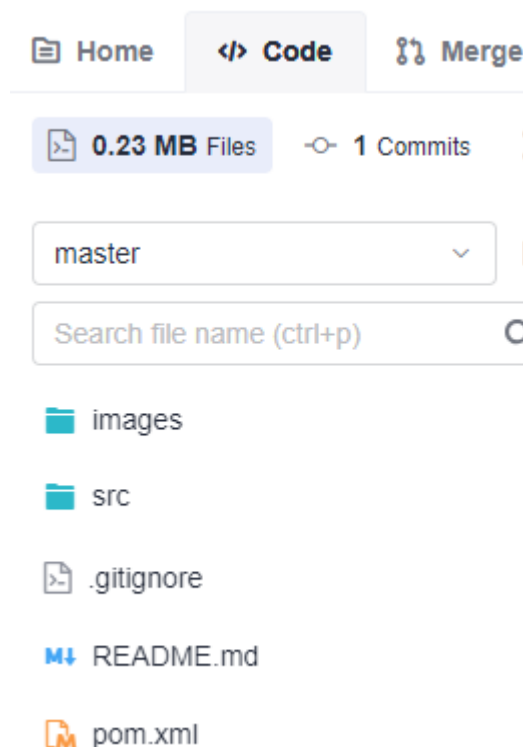
**Step 2** Click **New Repository**. On the displayed page, expand the **Project** drop-down list and select the project name you specify when creating a project. Then, select **Template** as the repository type. Click **Next**.

**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, <b>custom_task</b> .
Code Source	Select <b>Repo</b> to pull the code hosted in a CodeArts Repo repository for your build.
Repository	Select <b>custom_repo</b> , the code repository created in <a href="#">Creating a CodeArts Repo Repository</a> .
Default Branch	Keep the default value <b>master</b> .
Description	Enter additional information to describe the build task.

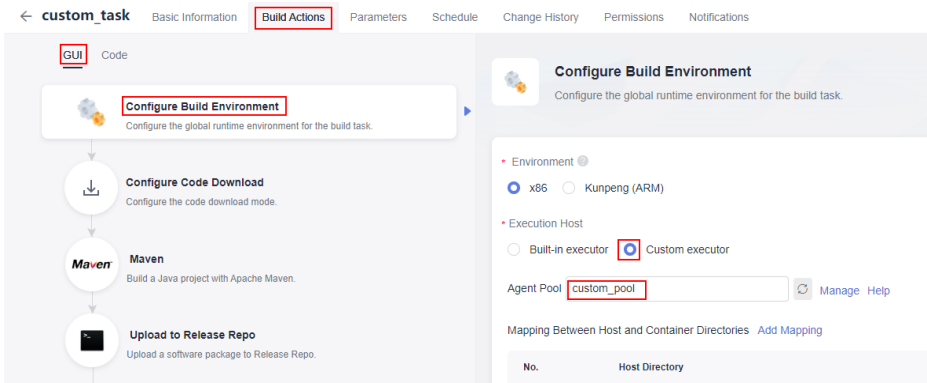
- Step 3** Select the **Maven** template and click **OK**. The **Build Actions** page is displayed.

**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 <b>Custom executor</b> .
Agent Pool	Expand the drop-down list and select <b>custom_pool</b> you created in section <b>"Creating an Agent Pool"</b> .

**Figure 5-3** Configuring the build environment

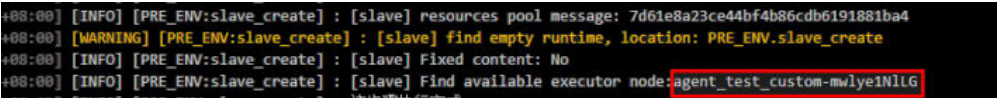


**Step 5** In the displayed dialog box, click **Confirm**. The build task run page is displayed.  
-----End

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




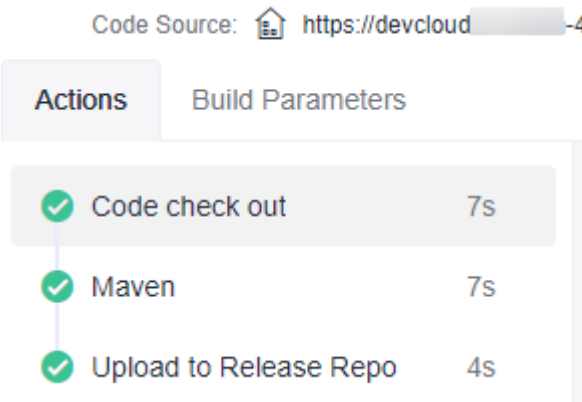
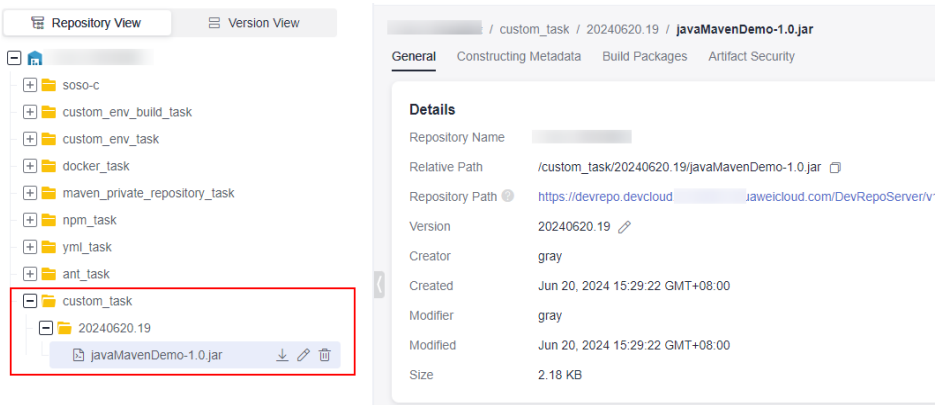
**Step 2** The build task is considered successful if each build action is marked with  after running.

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



----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
<a href="#">Creating a Project</a>	Create a project.
<a href="#">Creating a CodeArts Repo Repository</a>	Create a CodeArts Repo repository to store code files.
<a href="#">Creating a build.yml File</a>	Define the entire build process in <b>build.yml</b> .
<a href="#">Creating a Dockerfile</a>	Customize an image by modifying the Dockerfile.
<a href="#">Creating a Build Task</a>	Create a build task.
<a href="#">Viewing and Verifying the Build Results</a>	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 , 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\_yaml\_build** in the **Repository Name** field, and leave the other parameters as default.

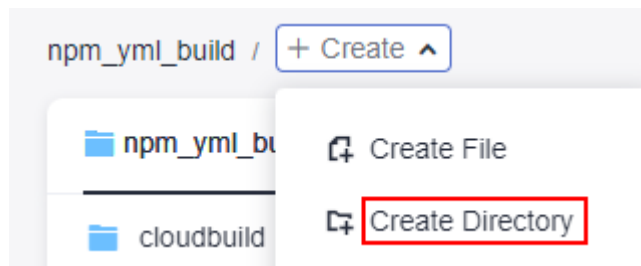
**Step 5** Click **OK**. The repository details page is displayed.

----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**.

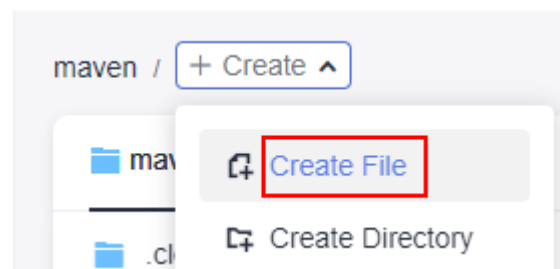
**Table 6-2** Creating a directory

Parameter	Description
Directory Name	Enter a directory name, for example, <b>.cloudbuild</b> . 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 [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 [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

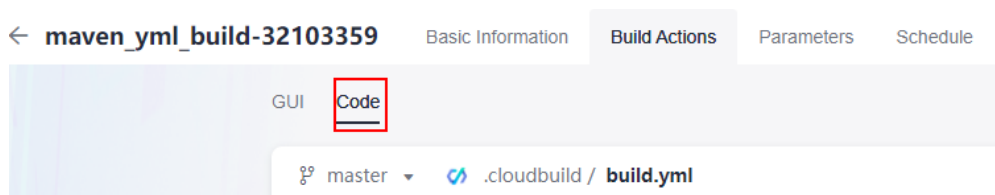
Parameter	Description
Name	Assign a custom name to the build task, for example, <b>maven_yml_build</b> .
Code Source	Select <b>Repo</b> .
Repository	Select <b>Repo01</b> , the code repository created in <a href="#">Creating a CodeArts Repo Repository</a> .

Parameter	Description
Default Branch	Keep the default value <b>master</b> .
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



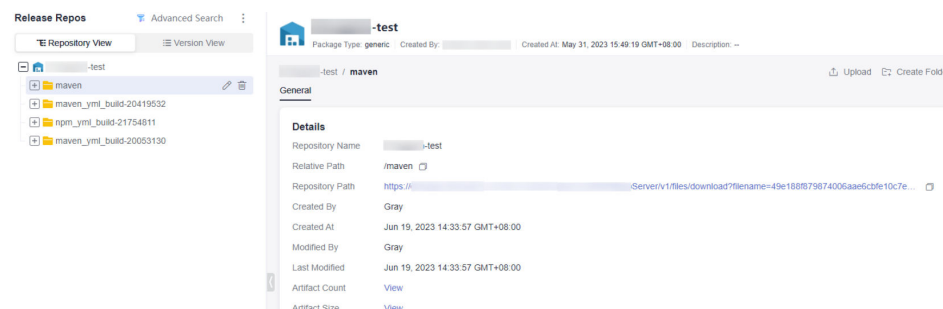
**Step 5** Click **Save and Run** in the upper right corner.

----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



- 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 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 build flow.
- You have permissions for CodeArts Repo.

Procedure


Table 7-1 Steps

Step	Description
#codeci_practice_1020/en-us_topic_0000001961117537_section034264112171	Create a project.
<a href="#">Creating a CodeArts Repo Repository</a>	Create a CodeArts Repo repository to store code files.

Step	Description
<a href="#">Creating a build.yml File</a>	Define the entire build process in <b>build.yml</b> .
<a href="#">Creating Task Scripts Used in build.yml</a>	Create a script for the whole build.
<a href="#">Creating and Running a Build Task</a>	Create and run a build flow task.
<a href="#">Viewing the Build Results</a>	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 , 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 # Assign a custom name to the task.
      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

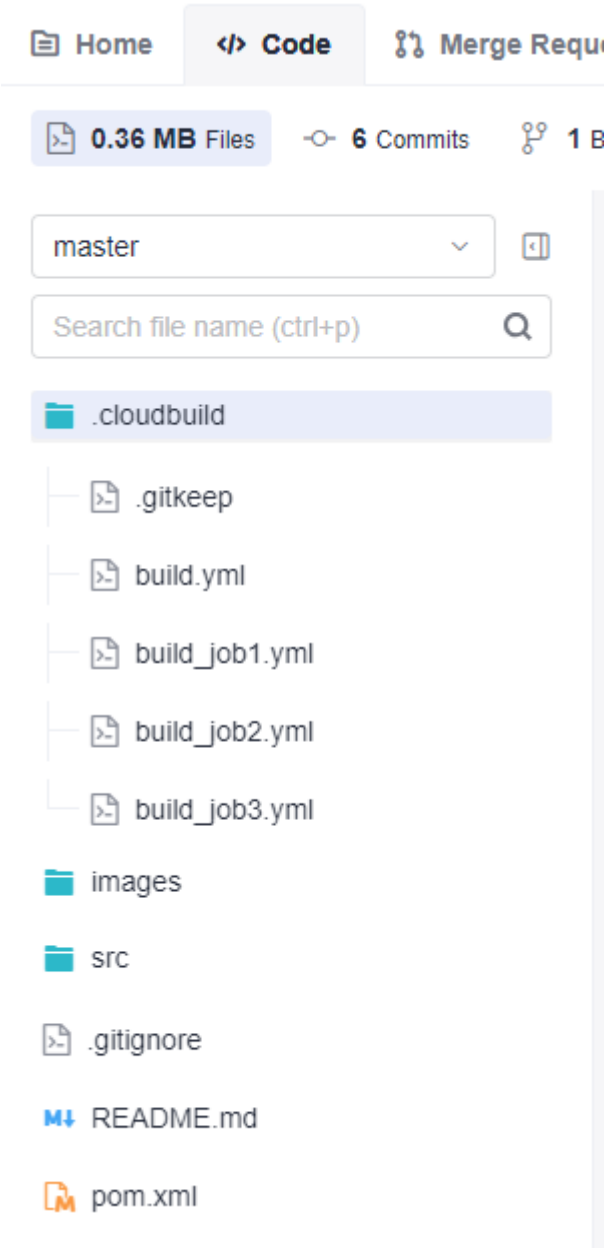
**Step 1** In the **.cloudbuild** directory, click **Create** and select **Create File** from the drop-down list. Name the file as **build\_job1.yml** and add the following code to the file:

```
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



----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, <b>BuildFlow</b> .

Parameter	Description
Code Source	Select <b>Repo</b> .
Repository	Select <b>Repo01</b> , the code repository created in <a href="#">Creating a CodeArts Repo Repository</a> .
Default Branch	Keep the default value <b>master</b> .
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.

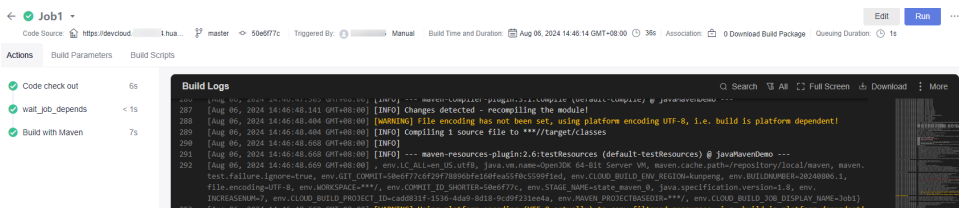
----End

## 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



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 [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
#codeci_practice_1002/en-us_topic_0000001962216721_section034264112171	Create a project.
<a href="#">Creating a Self-hosted Repo</a>	Create a self-hosted repo.



Step	Description
<a href="#">Checking Information About the Self-hosted Repo</a>	Check the ID and URL of the self-hosted repo, which will be used for configuring the <b>pom.xml</b> file of the code repository.
<a href="#">Uploading the settings.xml File to CodeArts Build</a>	Upload the <b>settings.xml</b> file to <b>Files</b> for management in CodeArts Build.
<a href="#">Creating a CodeArts Repo Repository</a>	Create a code repository.
<a href="#">Configuring the Self-hosted Repo Address for Storing Maven Build Products</a>	Configure the address of the self-hosted repo where the build product will be uploaded.
<a href="#">Creating a Build Task</a>	Create a build task.
<a href="#">Configuring Build Actions and Running the Build Task</a>	Configure actions <b>Download File from File Manager</b> and <b>Build with Maven</b> , and run the build task.
<a href="#">Viewing the Build Results</a>	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 , 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 <b>Local Repository</b> .
Repository Name	Assign a custom name to the repository, for example, <b>private_repository</b> .
Package Type	Select <b>Maven</b> .
Project	Leave the value as default. The field is autofilled with <b>build-bestpractice</b> in this practice.
Include Patterns	Leave it blank.
Version Policy	Specify the version of artifacts stored in the repository. Select either <b>Release</b> (the release version with stable functions) or <b>Snapshot</b> (the development version with unstable functions). You can also select both of them if needed. In this practice, select <b>Release</b> .
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.

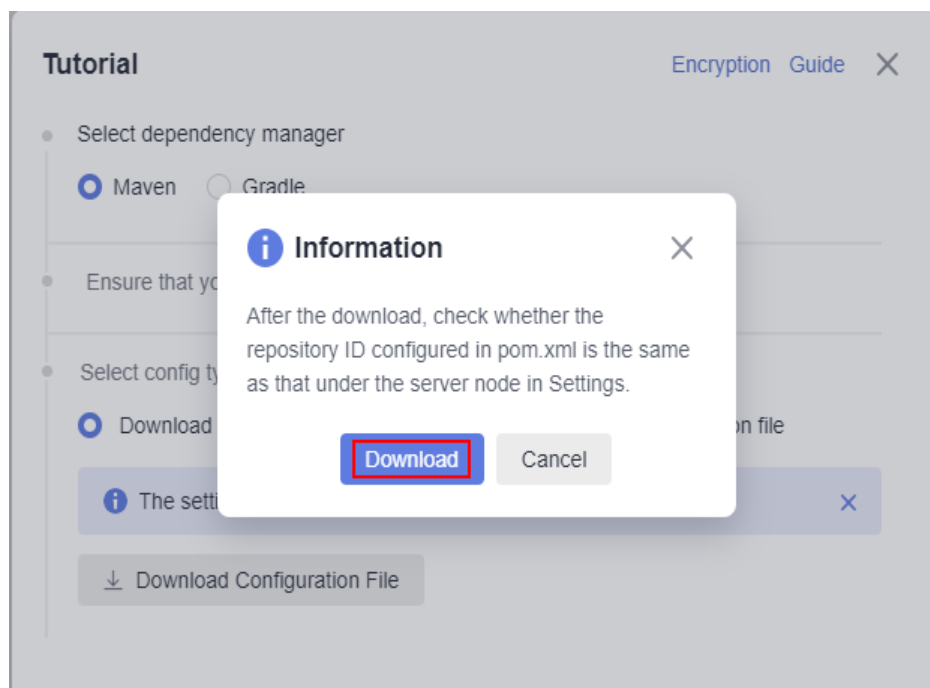
----End

## 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

- 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>
    </repository>
  </repositories>
  <enabled>true</enabled>
</profile>
<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>
  </repository>
  <enabled>true</enabled>
</repositories>
<enabled>false</enabled>
</snapshots>
```

----End

## Uploading the settings.xml File to CodeArts Build

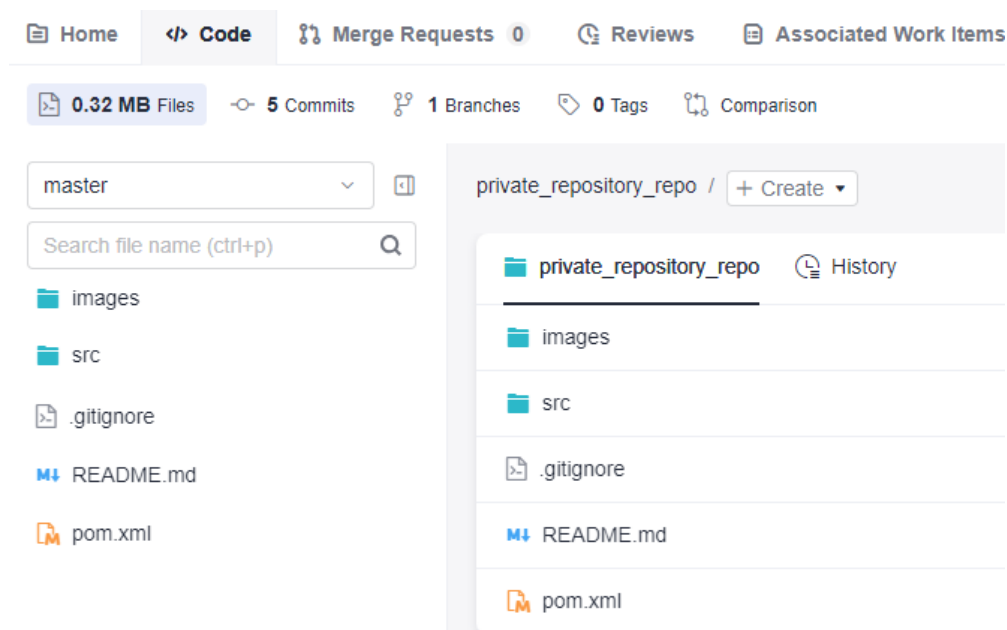
- Step 1** In the navigation pane, choose **CICD > Build**.
- Step 2** On the build task list page, choose **More > Files**.
- Step 3** On the displayed page, click **Upload File**.
- Step 4** In the displayed dialog box, upload the **settings.xml** file you download when [checking information about 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

```
<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**.

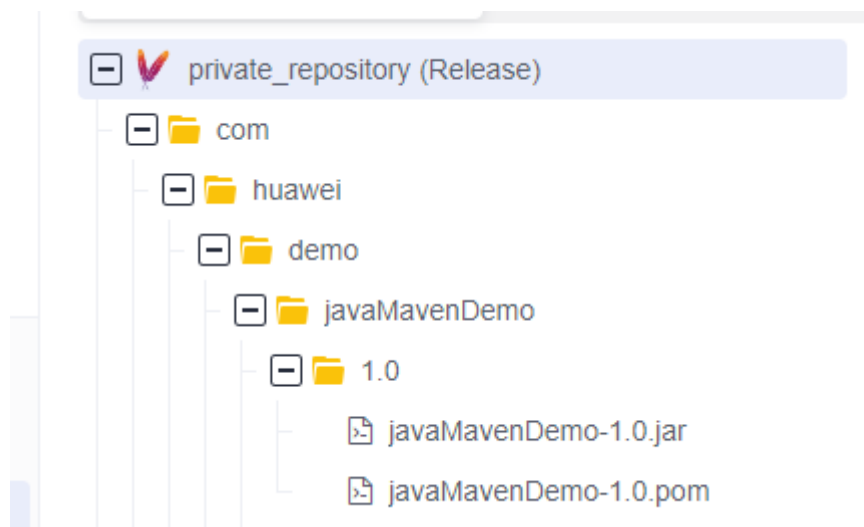
**Figure 8-7** Path for uploading dependencies

```
257 [2024/06/26 10:31:58.988 GMT+08:00] [INFO] Uploaded to release_..._f9e48463c23845438ca9efd3a7ec854e_maven_1_29: https://devrepo.devcloud.  
..._huaweicloud.com/artgalaxy/..._f9e48463c23845438ca9efd3a7ec854e_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
#codeci_practice_1035/en-us_topic_0000001970572204_section034264112171	Create a project.
<a href="#">Creating a CodeArts Repo Repository</a>	Create the code file used for your build.

Step	Description
<a href="#">Creating an Image as the Custom Build Environment</a>	Create an image as the custom environment.
<a href="#">Creating and Running a Build Task</a>	Create, configure, and run a build task.
<a href="#">Viewing the Build Results</a>	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 , 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**.

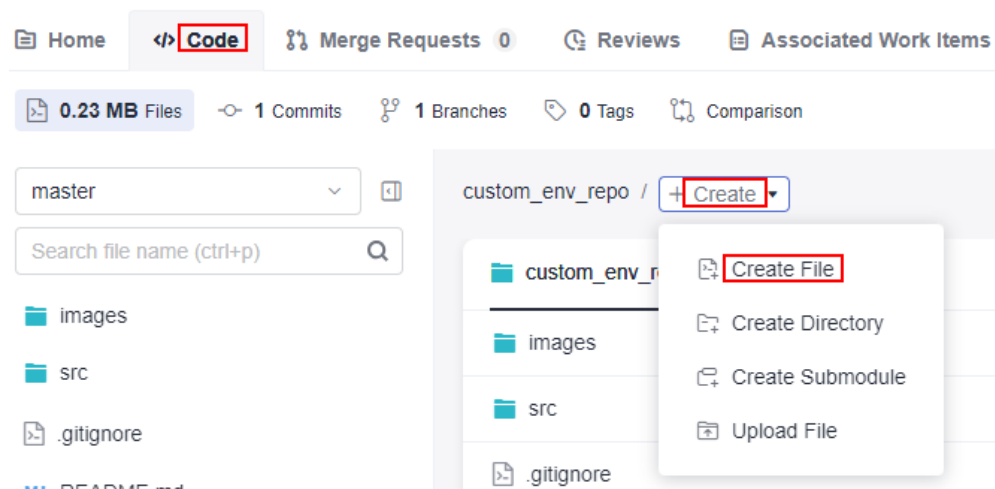
**Step 3** On the template selection page, select the **Java Maven Demo** template and click **Next**.

**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

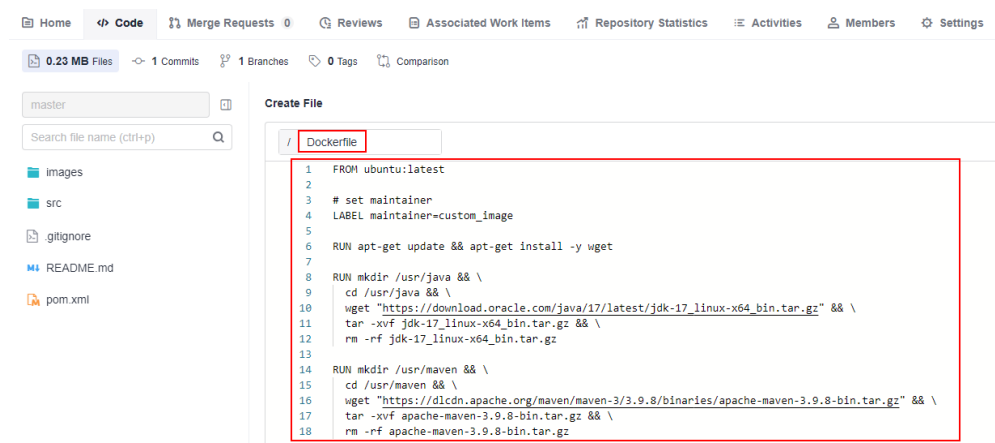
RUN mkdir /usr/maven && \
  cd /usr/maven && \
  wget "https://dlcdn.apache.org/maven/maven-3/3.9.8/binaries/apache-maven-3.9.8-bin.tar.gz" && \
  tar -xvf apache-maven-3.9.8-bin.tar.gz && \
  rm -rf apache-maven-3.9.8-bin.tar.gz

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



**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.
RUN	They are run when <b>docker build</b> creates an image. In this practice, <b>RUN</b> 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, <b>ENV</b> instructions set environment variables for JDK and Maven and add their directory to the <b>PATH</b> 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 <b>build</b> 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**.

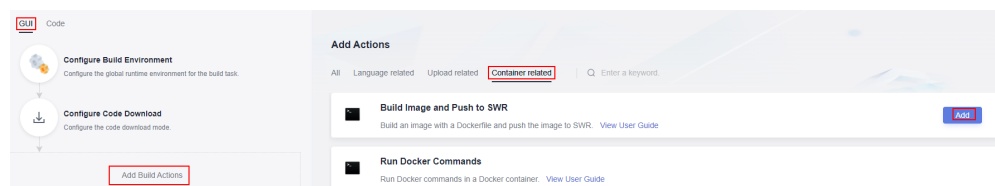
**Table 9-3** Basic information

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

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

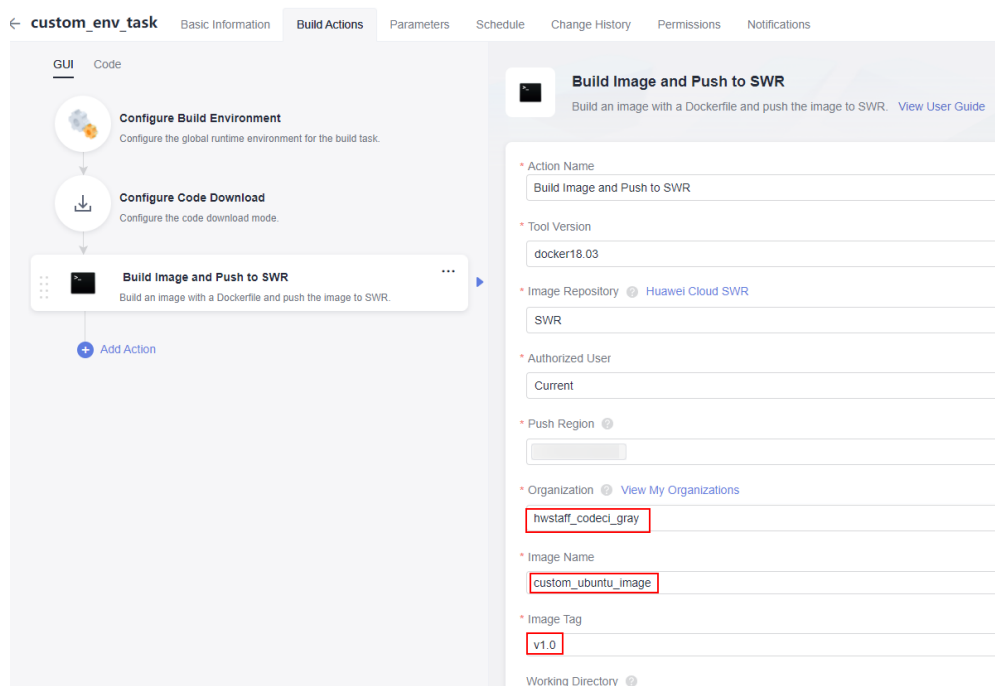
**Step 4** On the **Build Actions** page, click the **GUI** tab and click **Add Build Actions**.

**Figure 9-3** Adding a build action



**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



**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.

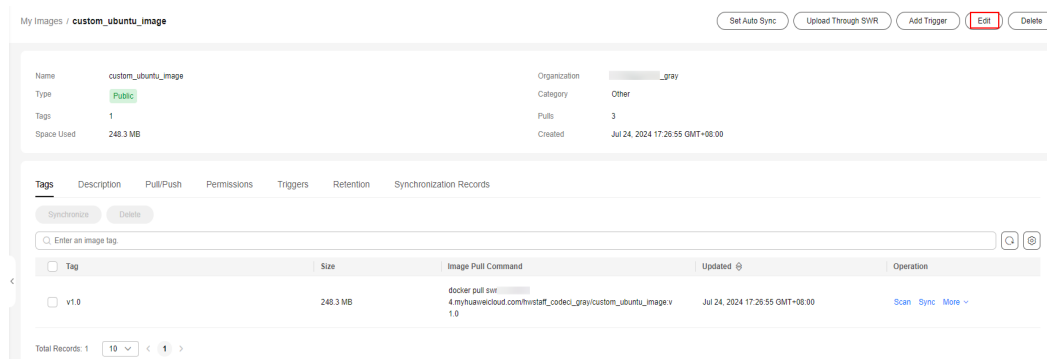
Figure 9-5 Build task run logs

```
80 [2024/07/24 17:15:58.627 GMT+08:00] Login Succeeded
81 [2024/07/24 17:15:58.923 GMT+08:00] Sending build context to Docker daemon 377.3kB
82 [2024/07/24 17:15:58.927 GMT+08:00] Step 1/10 : FROM ubuntu:latest
83 [2024/07/24 17:16:14.610 GMT+08:00] latest: Pulling from library/ubuntu
84 [2024/07/24 17:16:15.033 GMT+08:00] 7b1a6ab2e44d: Pulling fs layer
85 [2024/07/24 17:16:16.350 GMT+08:00] 7b1a6ab2e44d: Download complete
86 [2024/07/24 17:16:17.171 GMT+08:00] 7b1a6ab2e44d: Pull complete
87 [2024/07/24 17:16:17.177 GMT+08:00] Digest: sha256:626ffe58f0e756ee0254b638eb7e0f3b11d4da9675888f4781a58ae288f3322
88 [2024/07/24 17:16:17.184 GMT+08:00] Status: Downloaded newer image for ubuntu:latest
89 [2024/07/24 17:16:17.184 GMT+08:00] ----> ba6accdd29
90 [2024/07/24 17:16:17.184 GMT+08:00] Step 2/10 : LABEL maintainer=custom_image
91 [2024/07/24 17:16:17.360 GMT+08:00] ----> Running in db709e7459ae
92 [2024/07/24 17:16:17.428 GMT+08:00] Removing intermediate container db709e7459ae
93 [2024/07/24 17:16:17.428 GMT+08:00] ----> f0327e20cf8b
94 [2024/07/24 17:16:17.428 GMT+08:00] Step 3/10 : RUN apt-get update && apt-get install -y wget
95 [2024/07/24 17:16:17.456 GMT+08:00] ----> Running in 9dab0c202c86
96 [2024/07/24 17:16:18.201 GMT+08:00] Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
97 [2024/07/24 17:16:18.818 GMT+08:00] Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [128 kB]
98 [2024/07/24 17:16:20.098 GMT+08:00] Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [128 kB]
99 [2024/07/24 17:16:20.461 GMT+08:00] Get:4 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [3780 kB]
100 [2024/07/24 17:16:20.942 GMT+08:00] Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [128 kB]
101 [2024/07/24 17:16:21.840 GMT+08:00] Get:6 http://archive.ubuntu.com/ubuntu focal/multiverse amd64 Packages [177 kB]
102 [2024/07/24 17:16:22.370 GMT+08:00] Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
103 [2024/07/24 17:16:22.410 GMT+08:00] Get:8 http://archive.ubuntu.com/ubuntu focal/universe amd64 Packages [11.3 MB]
104 [2024/07/24 17:16:23.486 GMT+08:00] Get:9 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [3809 kB]
105 [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]
106 [2024/07/24 17:16:24.782 GMT+08:00] Get:11 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1249 kB]
107 [2024/07/24 17:16:40.566 GMT+08:00] Get:12 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
108 [2024/07/24 17:16:41.554 GMT+08:00] Get:13 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [3932 kB]
109 [2024/07/24 17:16:45.156 GMT+08:00] Get:14 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [4274 kB]
110 [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]
111 [2024/07/24 17:16:47.624 GMT+08:00] Get:16 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [1535 kB]
112 -----
```

**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 [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



**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

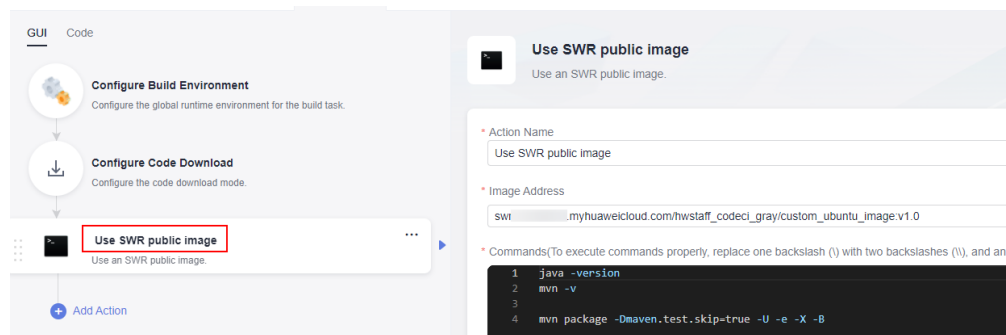
docker pull swr.
4.myhuaweicloud.com/hwstaff_codeci_gray/custom_ubuntu_image:v
1.0
```

----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



- 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

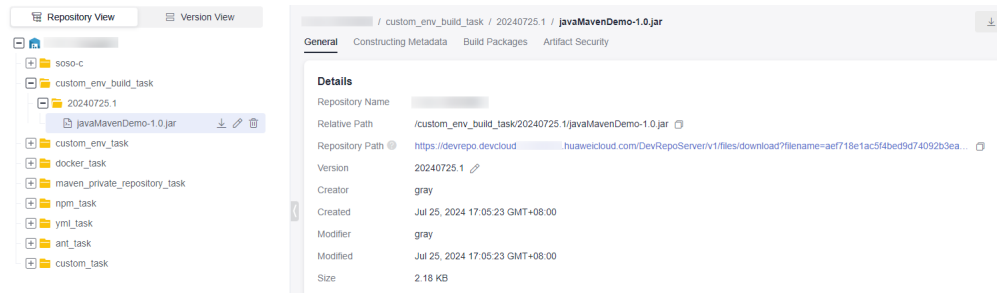
## Viewing the Build Results

- 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



----End