CodeArts Artifact

Best Practices

Issue	01
Date	2026-03-13

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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 CodeArts Artifact Best Practices

This document summarizes the operation practices of CodeArts Artifact in common application scenarios. It provides detailed solutions for each practice, helping users easily use CodeArts Artifact in different scenarios.

Table 1-1 CodeArts Artifact best practices

Best Practice	Description
Publishing a Maven Artifact to Self-Hosted Repos via a Build Task	This practice describes how to archive a Maven artifact by version to a self-hosted repo via a build task.
Publishing/Obtaining an npm Package via a Build Task	This practice describes how to publish a private package to an npm registry via a build task and obtain a dependency from the registry to complete the build task.
Publishing/Obtaining a Go Package via a Build Task	This practice describes how to publish a private package to a Go repository via a build task and obtain a dependency from the repository to complete the build task.
Publishing/Obtaining a PyPI Package via a Build Task	This practice describes how to publish a private package to a PyPI repository via a build task and obtain a dependency from the repository to complete the build task.
Uploading/Obtaining an RPM Package Using Linux Commands	This practice describes how to use Linux commands to upload a private package to an RPM repository and obtain a dependency from the repository.
Uploading/Obtaining a Debian Package Using Linux Commands	This practice describes how to use Linux commands to upload a private package to a Debian repository and obtain a dependency from the repository.

Best Practice	Description
Migrating Repository Data from Nexus to CodeArts Artifact	CodeArts Artifact provides a batch migration tool to quickly migrate hosted, proxy, and group repositories on Nexus to self-hosted repos. This streamlines O&M for efficiency.
Migrating Local Repository Data to CodeArts Artifact	CodeArts Artifact provides a batch migration tool to quickly the migrate Maven repository and npm registry on local disks to the Maven repository and npm registry in self-hosted repos. This streamlines operations and maintenance for efficiency.
Configuring CodeArts Artifact Permissions	This practice uses self-hosted repos as an example to describe how to quickly manage permissions for individual repositories and by project.

2 Publishing a Maven Artifact to Self-Hosted Repos via a Build Task

During software development, teams often rely on private packages to protect intellectual property and promote code reuse. However, fetching dependencies from Maven repositories can introduce permission issues or network latency, leading to build failures. Secure publishing and efficient retrieval of private packages remain critical challenges. This practice describes how to publish a private package to a Maven repository via a build task and fetch its dependencies from the repository to complete the build task, resolving the challenges mentioned earlier.

Billing

This function requires CodeArts Repo and CodeArts Build. You need to **purchase a CodeArts package**.

Prerequisites

- You already have a project. If no project is available, **create one**. For example, create a project named **project_maven**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Creating a Maven Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 On the **Self-hosted Repos** page, click **Create Repository** in the upper-right corner.

Step 3 Select **Local Repository** as the repository type, enter the repository name **maven_local**, and select **Maven** as the package type. Select **project_maven** created in **Prerequisites** from the **Project** drop-down list.

Step 4 Click **OK**. The created Maven repository is displayed in the **Repo View**.

----End

Configuring Package Versions in CodeArts Repo

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 Choose **Services** > **Repo** from the top menu bar.

Step 5 In the displayed dialog box, select **Merge Requests**, and click the confirm button.

Step 6 On the Repo homepage, click **Create Repository**.

Step 7 Select **project_maven** from the **Project** drop-down list, select **Template**, and click **Next**.

Step 8 Search for the **Java Maven Demo** template, and click **Next**.

Step 9 Enter the repository name **repo01** and click **OK**.

Step 10 Go back to Repo and click **pom.xml** to view the package configuration.

Step 11 On the package configuration page, the *<version>* field displays the version number of the current package. The default version number is **1.0**.

Click ✎ in the upper-right corner of the page to change the version number. Then, click **OK** to save the changes.

----End

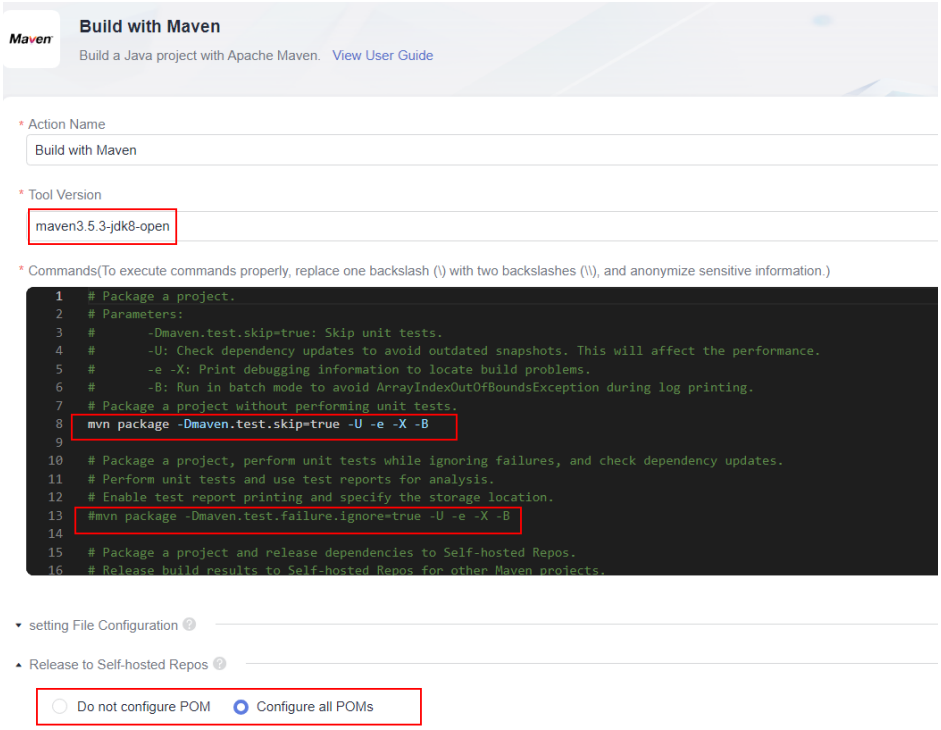
Publishing a Maven Artifact to Self-Hosted Repos via CodeArts Build

Step 1 After configuring the package version in Repo by referring to **Configuring Package Versions in CodeArts Repo**, click **Create Build Task** in the upper-right corner of the page.

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

Step 3 Edit the **Build with Maven** action.

- Select the desired tool version. In this example, **maven3.5.3-jdk8-open** is used.
- Find the following command and delete **#** in front of this command-line.
  ```
  #mvn deploy -Dmaven.test.skip=true -U -e -X -B
  ```
  Find the following command and add **#** in front of this command.
  ```
  mvn package -Dmaven.test.skip=true -U -e -X -B
  ```
- Select **Configure all POMs** under **Release to Self-hosted Repos**, and select the Maven repository **maven01** associated with the project.



Step 4 Click **Save and Execute** in the upper-right corner of the page to execute build task.

----End

Viewing Packages in the Version View of a Maven Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 Select the self-hosted repo, locate the target maven repository, and find the Maven artifact uploaded by build task.

Set the package version in Repo by referring to **Configuring Package Versions in CodeArts Repo** and archive packages of multiple versions to self-hosted repos.

Step 3 Click the **Version View** tab.

In the package list, view the number of versions and the latest version of the package obtained from the build task.

Step 4 Click a name in the **Package Name** column. The **Overview** page for the latest version of the package is displayed.

Step 5 Click the **Files** tab, click in the **Operation** column of the target package to download it to the localhost.

Step 6 After modifying a package and setting a new version number, click **Upload** in the right of the target self-hosted repo to upload the latest version of the package.

The package list in the **Version View** displays the latest uploaded version of each package and the number of archived versions.

----End

3 Publishing/Obtaining an npm Package via a Build Task

During software development, teams often rely on private packages to protect intellectual property and promote code reuse. However, fetching dependencies from npm registries can introduce permission issues or network latency, leading to build failures. Secure publishing and efficient retrieval of private packages remain critical challenges. This practice describes how to publish a private package to an npm registry via a build task and fetch its dependencies from the registry to complete the build task, resolving the challenges mentioned earlier.

Billing

This function requires CodeArts Repo and CodeArts Build. You need to **purchase a CodeArts package**.

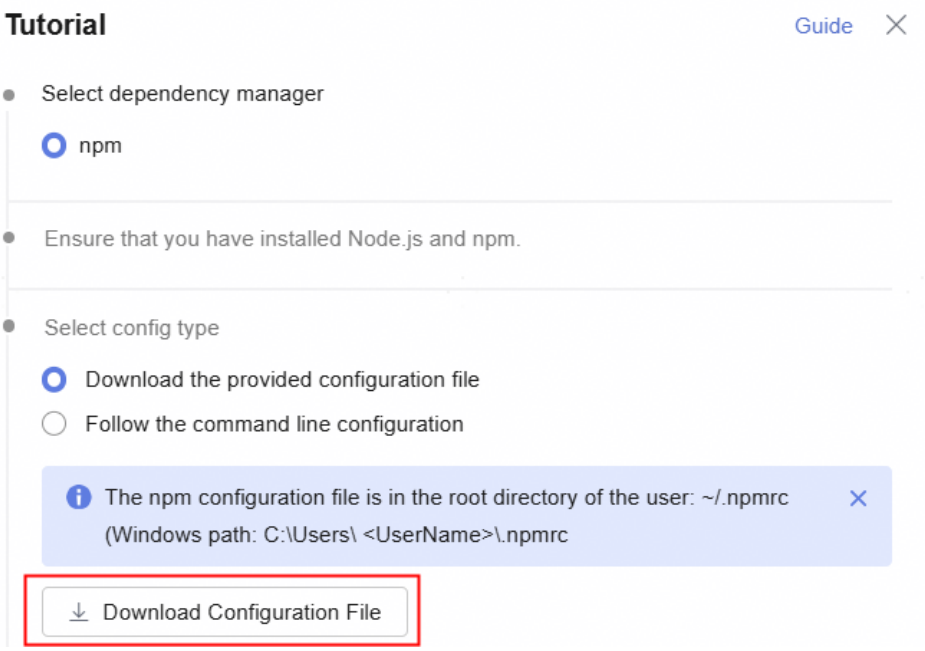
Prerequisites

- You already have a project. If no project is available, **create one**. For example, create a project named **project_npm**.
- You have created **an npm registry**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Publishing a Package to an npm Registry

Step 1 Download the configuration file.

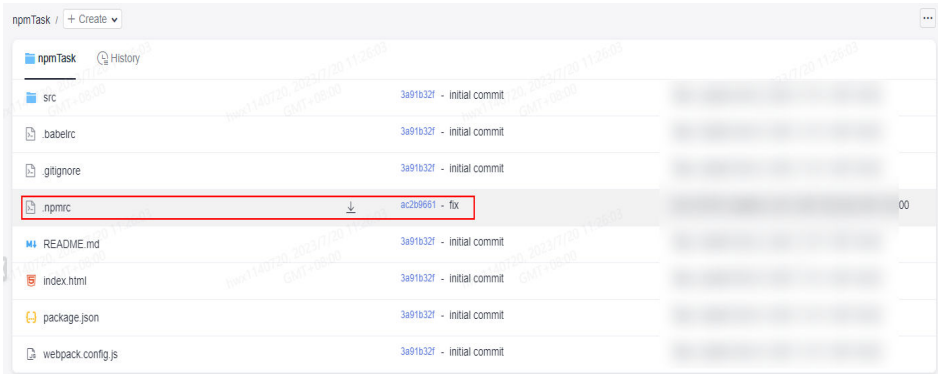
1. Use your Huawei Cloud account to access **self-hosted repos**.
2. Select the target npm registry from the repository list.
3. Click **Tutorial** on the right of the page.
4. In the displayed dialog box, click **Download Configuration File**.



5. Save the downloaded **npmrc** file as an **.npmrc** file.

Step 2 Configure a repository.

1. **Log in to the Huawei Cloud console** with your Huawei Cloud account.
2. Click ☰ in the upper-left corner and choose **Developer Services** > **CodeArts** from the service list.
3. Click **Access Service**. The homepage of CodeArts is displayed.
4. Choose **Services** > **Repo** from the top menu bar.
5. Create a Node.js repository. For details, see **Creating a Repository**. This procedure uses the **Nodejs Webpack Demo** template.
6. Go to the repository and upload the **.npmrc** file to the root directory of the repository. For details, see **Uploading Code Files to CodeArts Repo**.



Step 3 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

   Select **npm** and click **OK**.

2. Edit the **Build with npm** action.
   - Select the desired tool version. In this example, **nodejs12.7.0** is used.
   - Delete the existing commands and run the following instead.
     ```
     export PATH=$PATH:/root/.npm-global/bin
     npm config set strict-ssl false
     npm publish --verbose
     ```
3. Click **Save and Run** on the right of the page to start the build task.

   After the task is successfully executed, go to the self-hosted repo page and find the uploaded npm package.

----End

Obtaining a Dependency from an npm Registry

The following procedure uses the npm package published in **Publishing a Package to an npm Registry** as an example to describe how to obtain a dependency from an npm registry.

Step 1 Configure a repository.

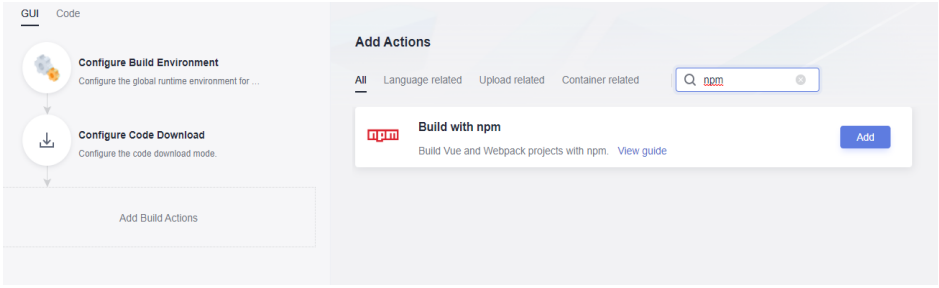
1. **Log in to the Huawei Cloud console** with your Huawei Cloud account.
2. Click  in the upper-left corner and choose **Developer Services** > **CodeArts** from the service list.
3. Click **Access Service**. The homepage of CodeArts is displayed.
4. Choose **Services** > **Repo** from the top menu bar.
5. Create a Node.js repository. For details, see **Creating a Repository**. This procedure uses the **Nodejs Webpack Demo** template.
6. Obtain the **.npmrc** file (see **Publishing a Package to an npm Registry**) and upload it to the root directory of the repository where the npm dependency is to be used.
7. Find and open the **package.json** file in the repository, and configure the dependency to the **dependencies** field. In this document, the value is as follows.
   ```
   "@test/vue-demo": "^1.0.0"
   ```

Step 2 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

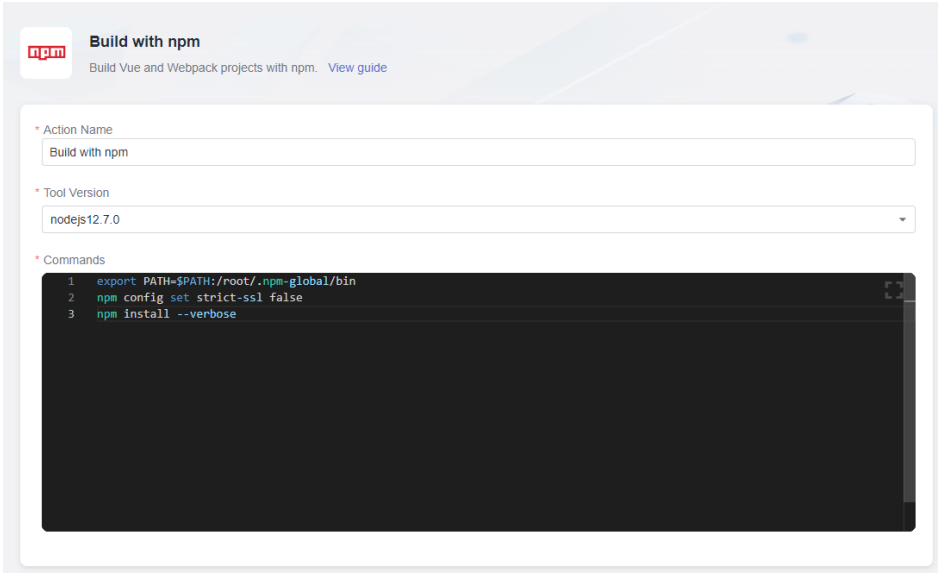
   Select **Blank Template** and click **OK**.

2. Add the **Build with npm** action.



3. Edit the **Build with npm** action.
   - Select the desired tool version. In this example, **nodejs12.7.0** is used.
   - Delete the existing commands and run the following instead.

     ```
     export PATH=$PATH:/root/.npm-global/bin
     npm config set strict-ssl false
     npm install --verbose
     ```



Step 3 Click **Save and Run** on the right of the page to start the build task.

After the task is successfully executed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the npm registry.

----End

npm Commands

When configuring build tasks, you can also run the following npm commands as required.

- Delete an existing package from the npm registry.


```
npm unpublish @scope/packageName@version
```

- Obtain tags.
  ```
  npm dist-tag list @scope/packageName
  ```
- Add a tag.
  ```
  npm dist-tag add @scope/packageName@version tagName --registry registryUrl --verbose
  ```
- Delete a tag.
  ```
  npm dist-tag rm @scope/packageName@version tagName --registry registryUrl --verbose
  ```

Command parameter description:

- *scope*: path of a self-hosted repo. For details about how to obtain the path, see **Publishing a Package to an npm Registry**.
- *packageName*: the part following *scope* in the *name* field of the **package.json** file.
- *version*: value of the *version* field in the **package.json** file.
- *registryUrl*: URL of the self-hosted repo referenced by *scope* in the configuration file.
- *tagName*: tag name.

The following uses the package published in **Publishing a Package to an npm Registry** as an example.

- *scope*: **test**
- *packageName*: **vue-demo**
- *version*: **1.0.0**

The command for deleting this package is as follows.

```
npm unpublish @test/vue-demo@1.0.0
```

4 Publishing/Obtaining a Go Package via a Build Task

In enterprise software development, teams often use private packages to build and deploy application for secure and exclusive code. However, fetching dependencies from a repository can be challenging, sometimes causing build failures. This practice describes how to publish a private package to a Go repository via a build task and fetch its dependencies from the repository to complete the build task, resolving the challenges mentioned earlier.

Billing

This function requires CodeArts Repo and CodeArts Build. You need to **purchase a CodeArts package**.

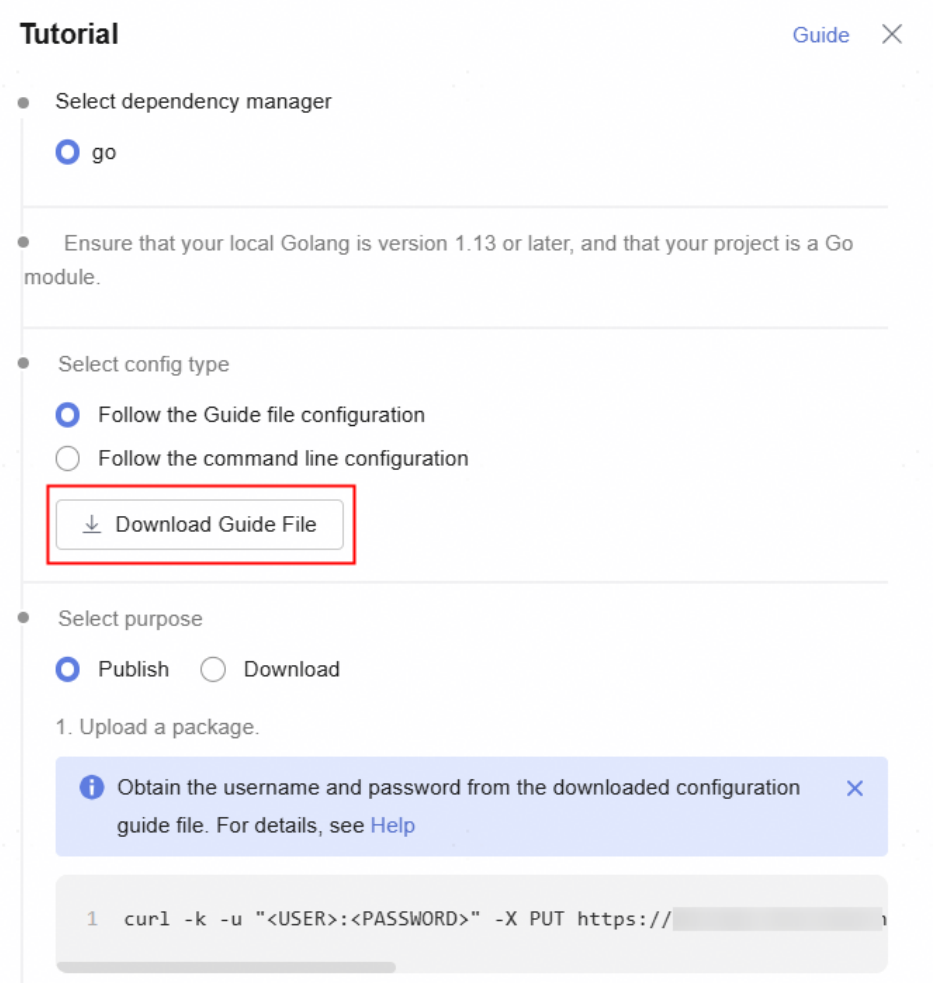
Prerequisites

- You already have a project. If no project is available, **create one**. For example, create a project named **project_go**.
- You have created **a Go repository**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Publishing a Package to a Go Repository

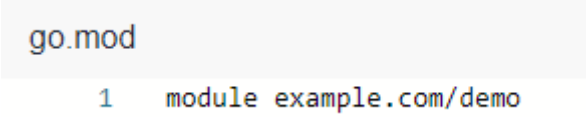
Step 1 Download the configuration file.

1. Use your Huawei Cloud account to access **self-hosted repos**.
2. Select the created Go repository. Click **Tutorial** on the right of the page.
3. In the displayed dialog box, click **Download Guide File**.



Step 2 Configure a repository.

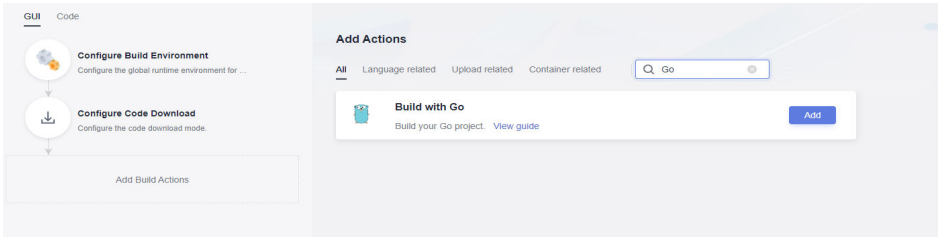
1. Go to Repo and create a Go repository. For details, see **Creating a Repository** This procedure uses the **Go web Demo** template.
2. Prepare the **go.mod** and upload it to the root directory of the repository. For details, see **Uploading Code Files to CodeArts Repo** The following figure shows the **go.mod** file used in this example.



Step 3 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

   Select **Blank Template** and click **OK**.
2. Add the **Build with Go** action.



3. Edit the **Build with Go** action.
   a. Select the desired tool version. In this example, **go-1.13.1** is used.
   b. Add the following commands to the end of the **Build with Go** action.

      ```
      cd $GOPATH/src/
      zip -D -r v1.0.0.zip  example.com/
      ```

   c. Copy the steps under **Configuring Go Environment Variables on LINUX** from the file to the command box.

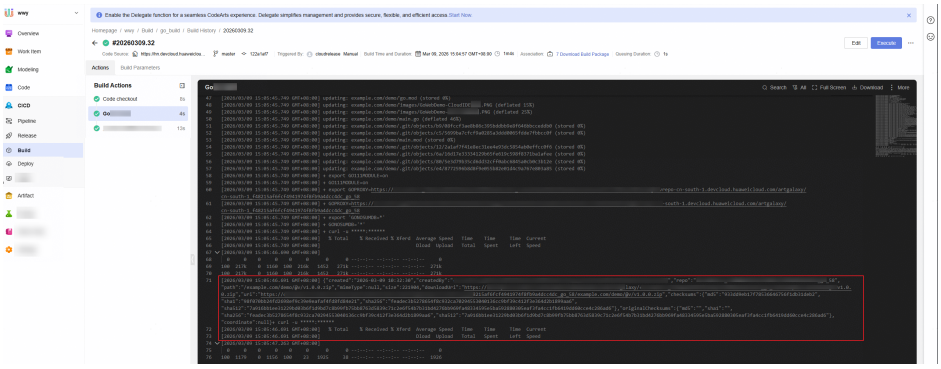
      Copy the Go publish command segment in the configuration file to the command box, and replace the parameters in the commands by referring to **Go Modules**. (In this example, the package version is **v1.0.0**.)

      Example:

      ```
      curl -u "<USER>:<PASSWORD>" -X PUT https://<URL>//example.com/demo/@v/v1.0.0.zip -T $GOPATH/src/v1.0.0.zip
      curl -u "<USER>:<PASSWORD>" -X PUT https://<URL>/example.com/demo/@v/v1.0.0.mod -T $GOPATH/src/example.com/demo/go.mod
      ```

4. Click **Save and Run** on the right of the page to start the build task.

   When the message **build successful** is displayed, go to the Go repository page and find the uploaded Go package.



----End

Obtaining a Dependency from a Go Repository

Step 1 Download the configuration file by referring to **Publishing a Package to a Go Repository**.

Step 2 Go to Repo and create a Go repository. For details, see **Creating a Repository** This procedure uses the **Go web Demo** template.

Step 3 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

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

2. Add the **Build with Go** action.
3. Edit the **Build with Go** action.
   a. Select the desired tool version. In this example, **go-1.13.1** is used.
   b. Delete the existing command. Then open the downloaded configuration file and copy the steps under **Configuring Go Environment Variables on Linux** to the command box.
   c. Copy the Go download commands in the configuration file to the command box and replace the **<modulename>** parameter with the actual value. (In this example, the parameter is set to **example.com/demo**).

      Example:

      ```
      go get -v example.com/demo@v1.0.0
      ```

Step 4 Click **Save and Run** on the right of the page to start the build task.

NOTE

If the message **can't request explicit version of path in main module** is displayed when you run the download command, check whether the package to download is built from the current repository.

When the message **build successful** is displayed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the Go repository.

----End

Go Modules

This section describes how to build and upload Go packages through Go modules.

Perform the following steps:

1. Create a source folder in the working directory.
   ```
   mkdir -p {module}@{version}
   ```
2. Copy the code to the sources folder.
   ```
   cp -rf . {module}@{version}
   ```
3. Compress the package into a ZIP package.
   ```
   zip -D -r [package name] [package root directory]
   ```
4. Upload the ZIP package and the **go.mod** file to the self-hosted repo.
   ```
   curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/{filePath} -T {{localFile}}
   ```

The package directory varies according to the package version. The version can be:

- Versions earlier than v2.0: The directory is the same as the path of the **go.mod** file. No special directory structure is required.
- v2.0 or later:
  - If the first line in the **go.mod** file ends with **/vX**, the directory must contain **/vX**. For example, if the version is v2.0.1, the directory must contain **v2**.
  - If the first line in the **go.mod** file does not end with **/vN**, the directory remains unchanged and the name of the file to be uploaded must contain **+incompatible**.

The following are examples of package directories for different versions.

- **Versions earlier than v2.0**

  The **go.mod** file is used as an example.

  go.mod

  ```
  1  module example.com/demo
  ```

  a. Create a source folder in the working directory.

     The value of **module** is **example.com/demo** and that of **version** is **1.0.0**. The command is as follows.

     ```
     mkdir -p ~/example.com/demo@v1.0.0
     ```

  b. Copy the code to the sources folder.

     The command is as follows (with the same parameter values as the previous command).

     ```
     cp -rf . ~/example.com/demo@v1.0.0/
     ```

  c. Compress the package into a ZIP package.

     Run the following command to go to the upper-level directory of the root directory where the ZIP package is located.

     ```
     cd ~
     ```

     Then, use the **zip** command to compress the code into a package. In this command, the **package root directory** is **example.com** and the **package name** is **v1.0.0.zip**. The command is as follows.

     ```
     zip -D -r v1.0.0.zip  example.com/
     ```

  d. Upload the ZIP package and the **go.mod** file to the self-hosted repo.

     Parameters *username*, *password*, and *repoUrl* can be obtained from the configuration file.

     - For the ZIP package, the value of **filePath** is **example.com/demo/@v/v1.0.0.zip** and that of **localFile** is **v1.0.0.zip**.
     - For the **go.mod** file, the value of **filePath** is **example.com/demo/@v/v1.0.0.mod** and that of **localFile** is **example.com/demo@v1.0.0/go.mod**.

     The commands are as follows (replace *username*, *password*, and *repoUrl* with the actual values).

     ```
     curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/v1.0.0.zip -T v1.0.0.zip
     curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/v1.0.0.mod -T example.com/demo@v1.0.0/go.mod
     ```

- **v2.0 and later, with the first line in go.mod ending with /vX**

  The **go.mod** file is used as an example.

  go.mod

  ```
  1  module example.com/demo/v2
  ```

  a. Create a source folder in the working directory.

     The value of **module** is **example.com/demo/v2** and that of **version** is **2.0.0**. The command is as follows.


```
mkdir -p ~/example.com/demo/v2@v2.0.0
```

b. Copy the code to the sources folder.

The command is as follows (with the same parameter values as the previous command).

```
cp -rf . ~/example.com/demo/v2@v2.0.0/
```

c. Compress the package into a ZIP package.

Run the following command to go to the upper-level directory of the root directory where the ZIP package is located.

```
cd ~
```

Then, use the **zip** command to compress the code into a package. In this command, the **package root directory** is **example.com** and the **package name** is **v2.0.0.zip**. The command is as follows.

```
zip -D -r v2.0.0.zip  example.com/
```

d. Upload the ZIP package and the **go.mod** file to the self-hosted repo.

Parameters *username*, *password*, and *repoUrl* can be obtained from the configuration file.

- For the ZIP package, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.zip** and that of **localFile** is **v2.0.0.zip**.
- For the **go.mod** file, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.mod** and that of **localFile** is **example.com/demo/v2@v2.0.0/go.mod**.

The commands are as follows (replace *username*, *password*, and *repoUrl* with the actual values).

```
curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/v2/@v/v2.0.0.zip -T v2.0.0.zip
curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/v2/@v/v2.0.0.mod -T example.com/demo/v2@v2.0.0/go.mod
```

- **v2.0 and later, with the first line in go.mod not ending with /vX**

The **go.mod** file is used as an example.

go.mod

```
1  module example.com/demo
```

a. Create a source folder in the working directory.

The value of **module** is **example.com/demo** and that of **version** is **3.0.0**. The command is as follows.

```
mkdir -p ~/example.com/demo@v3.0.0+incompatible
```

b. Copy the code to the sources folder.

The command is as follows (with the same parameter values as the previous command).

```
cp -rf . ~/example.com/demo@v3.0.0+incompatible/
```

c. Compress the package into a ZIP package.

Run the following command to go to the upper-level directory of the root directory where the ZIP package is located.

```
cd ~
```


Then, use the **zip** command to compress the code into a package. In this command, the **package root directory** is **example.com** and the **package name** is **v3.0.0.zip**. The command is as follows.

```
zip -D -r v3.0.0.zip  example.com/
```

d. Upload the ZIP package and the **go.mod** file to the self-hosted repo.

Parameters *username*, *password*, and *repoUrl* can be obtained from the configuration file.

- For the ZIP package, the value of **filePath** is **example.com/demo/@v/v3.0.0+incompatible.zip** and that of **localFile** is **v3.0.0.zip**.
- For the **go.mod** file, the value of **filePath** is **example.com/demo/@v/v3.0.0+incompatible.mod** and that of **localFile** is **example.com/demo@v3.0.0+incompatible/go.mod**.

The commands are as follows (replace *username*, *password*, and *repoUrl* with the actual values).

```
curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/
v3.0.0+incompatible.zip -T v3.0.0.zip
curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/
v3.0.0+incompatible.mod -T example.com/demo@v3.0.0+incompatible/go.mod
```

5 Publishing/Obtaining a PyPI Package via a Build Task

During software development, teams often use private packages to build and deploy applications. However, fetching dependencies from PyPI repositories can introduce permission or network issues, resulting in build failures. Ensuring seamless publishing and fetching private packages is therefore essential. This practice describes how to publish a private package to a PyPI repository via a build task and fetch its dependencies from the repository to complete the build task, supporting smooth development.

Billing

This function requires CodeArts Repo and CodeArts Build. You need to **purchase a CodeArts package**.

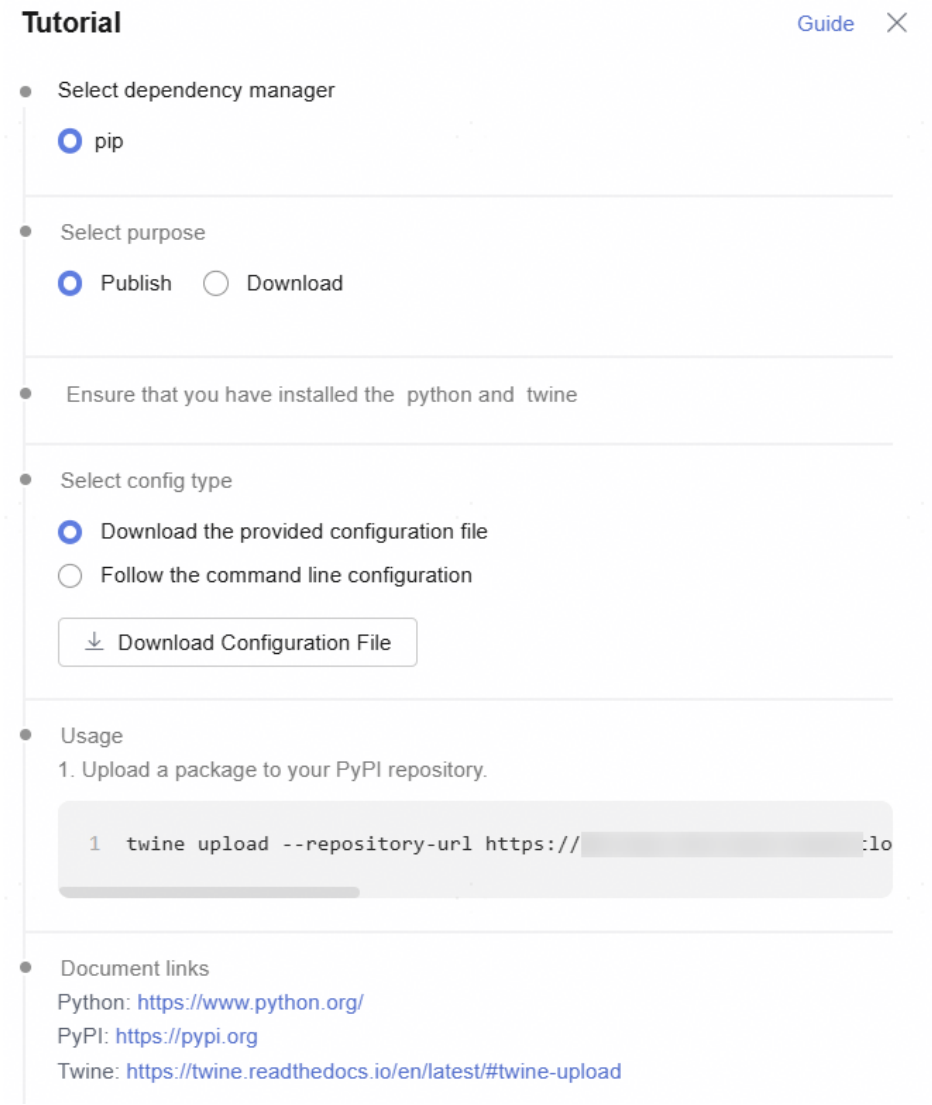
Prerequisites

- You already have a project. If no project is available, **create one**. For example, create a project named **project_PyPI**.
- You have created **a PyPI repository**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Publishing a Package to a PyPI Repository

Step 1 Download the configuration file.

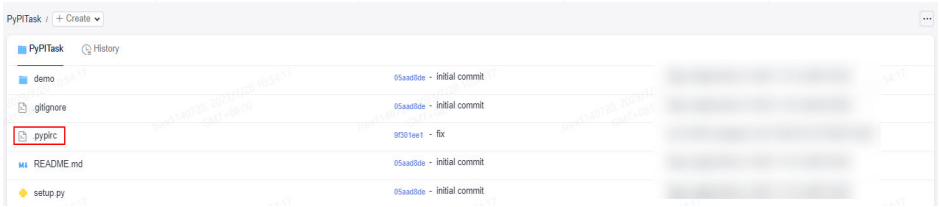
1. Use your Huawei Cloud account to access **self-hosted repos**.
2. Select a PyPI repository. Click **Tutorial** on the right of the page.
3. In the displayed dialog box, set **Select purpose** to **Publish**, select **Download the provided configuration file**, and click **Download Configuration File**.



4. Save the downloaded **PYPIRC** file as a **.pypirc** file.

Step 2 Configure a repository.

1. Go to Repo and create a Python repository. For details, see **Creating a Repository**. This procedure uses the **Python3 Demo** template.
2. Go to the repository and upload the **.pypirc** file to the root directory of the repository. For details, see
3. Create a **requirements.txt** file, change the content to **demo ==1.0**, and upload the file to the root directory of the repository. For details, see **Uploading Code Files to CodeArts Repo**.



Step 3 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

   Select **Setuptools** and click **OK**.

2. Edit the **Build with Setuptools** action.

   a. Select the desired tool version. In this example, **python3.6** is used.

   b. Delete the existing commands and run the following instead.

   ```
   # Ensure that the setup.py file exists in the root directory of the code, and run the following command to pack the project into a WHL package.
   python setup.py bdist_wheel
   # Set the .pypirc file in the root directory of the current project as the configuration file.
   cp -rf .pypirc ~/
   # Upload the package to the PyPI repository.
   twine upload -r pypi dist/*
   ```

   - If a certificate error is reported during the upload, add the following command to the top of the command in **3.b** to set environment variables to skip certificate verification. (If the Twine version is earlier than or equal to 3.8.0 and the request version is earlier than or equal to 2.27, ignore the following command.)

     ```
     export CURL_CA_BUNDLE=""
     ```

   - If a code error is reported during the upload, add the following command to the top of the command in **3.b** to set the Python code to UTF-8.

     ```
     export PYTHONIOENCODING=utf-8
     ```

3. Click **Save and Run** on the right of the page to start the build task.

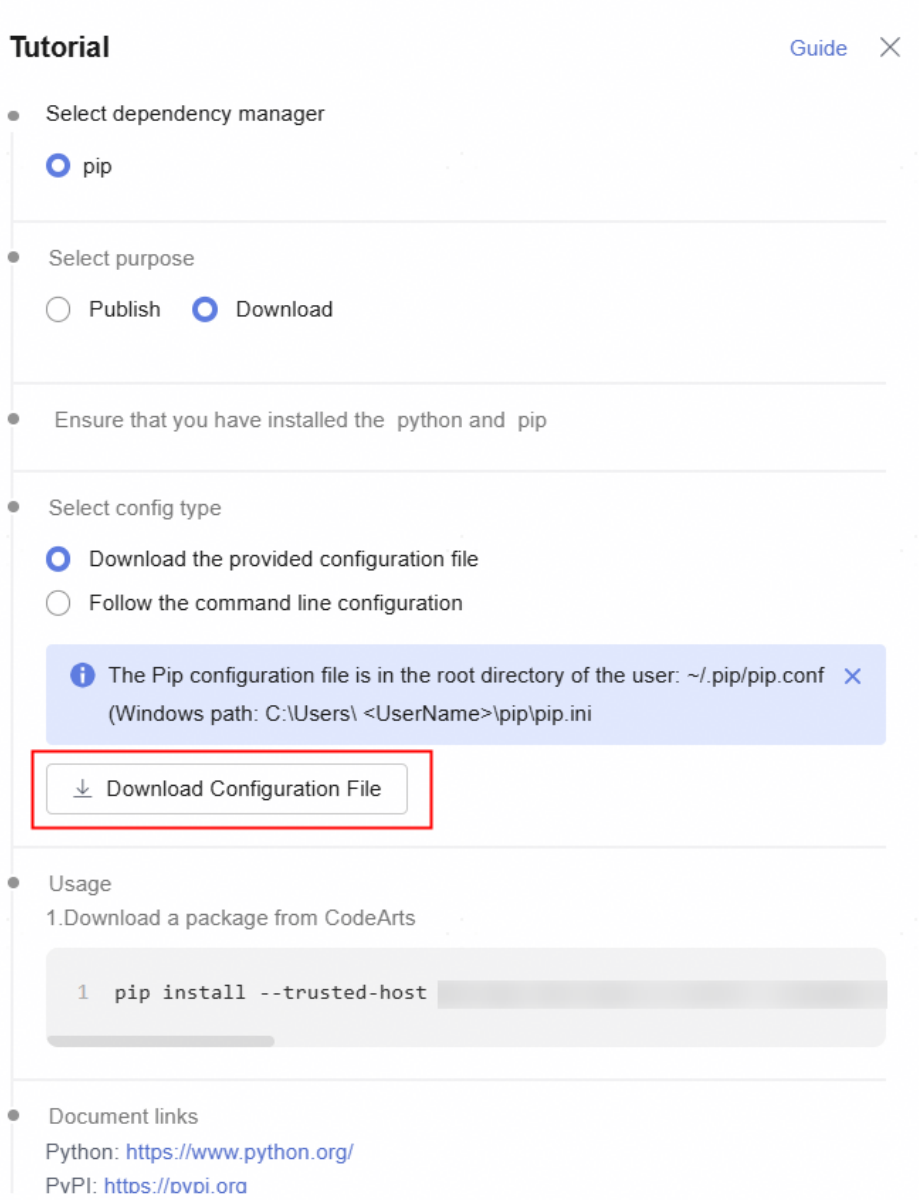
   After the task is successfully executed, go to the self-hosted repo page and find the uploaded PyPI package.

----End

Obtaining a Dependency from a PyPI Repository

Step 1 Download the configuration file.

1. Use your Huawei Cloud account to access **self-hosted repos**.
2. Select the PyPI repository and click **Tutorial** on the right of the page.
3. In the displayed dialog box, set **Select purpose** to **Download**, select **Download the provided configuration file**, and click **Download Configuration File**.

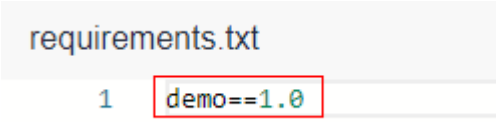


4. Save the downloaded **pip.ini** file as a **pip.conf** file.

Step 2 Configure a repository.

1. Go to Repo and create a Python repository. For details, see **Creating a Repository**. This procedure uses the **Python3 Demo** template.
2. Go to Repo, and upload the **pip.conf** file to the root directory of the repository where the PyPI dependency is to be used.
3. Find the **requirements.txt** file in the repository and open it. If the file is not found, create it by referring to **Managing Files** Add the dependency configuration to this file, as shown in the following figure.

```
demo ==1.0
```



Step 3 Configure and run a build task.

1. On the Repo page, select the repository and click **Create Build Task** in the upper right.

   Select **Setuptools** and click **OK**.

2. Edit the **Build with Setuptools** action.
   - Select the desired tool version. In this example, **python3.6** is used.
   - Delete the existing commands and run the following instead.

```
# Set the pip.conf file in the root directory of the current project as the configuration file.
 export PIP_CONFIG_FILE=./pip.conf
# Download the PyPI package.
 pip install -r requirements.txt --no-cache-dir
```

Step 4 Click **Save and Run** on the right of the page to start the build task.

After the task is successfully executed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the PyPI repository.

----End

6 Uploading/Obtaining an RPM Package Using Linux Commands

Software developers often need to upload custom private packages to an RPM repository so that other team members can easily access and use them. However, without clear guidance, developers may encounter issues when uploading packages using Linux commands, leading to upload or dependency retrieval failures. Ensuring successful upload and access to private packages is therefore a common challenge. This document describes how to upload a private package to an RPM repository and fetch dependencies from the repository using Linux commands.

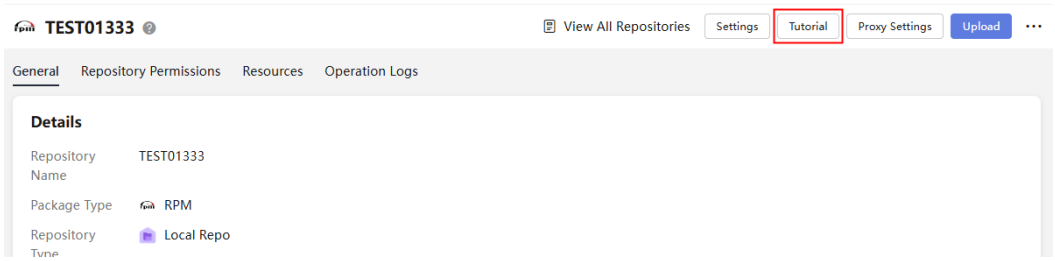
Prerequisites

- You have an RPM package available.
- You have a Linux host that can connect to the Internet.
- You have created **an RPM repository**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Publishing a Package to an RPM Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 Select an RPM repository. Click **Tutorial** on the right of the page.



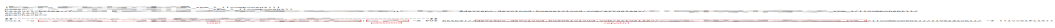
Step 3 In the displayed dialog box, click **Download Configuration File**.

Step 4 On the Linux host, run the following command to upload an RPM package.

```
curl -u {{user}}:{{password}} -X PUT https://{{repoUrl}}/{{component}}/{{version}}/ -T {{localFile}}
```

In this command, *user*, *password*, and *repoUrl* can be obtained from the **RPM upload command** in the configuration file downloaded in the **previous step**.

- *user*: string before the colon (:) between **curl -u** and **-X**
- *password*: string after the colon (:) between **curl -u** and **-X**
- *repoUrl*: string between **https://** and **/{{component}}**



component, *version*, and *localFile* can be obtained from the RPM package to be uploaded. The **hello-0.17.2-54.x86_64.rpm** package is used as an example.

- *component*: software name, for example, **hello**.
- *version*: software version, for example, **0.17.2**.
- *localFile*: RPM component, for example, **hello-0.17.2-54.x86_64.rpm**.

The following figure shows the complete commands.

Step 5 After the commands are successfully executed, go to the self-hosted repo page and find the uploaded RPM package.

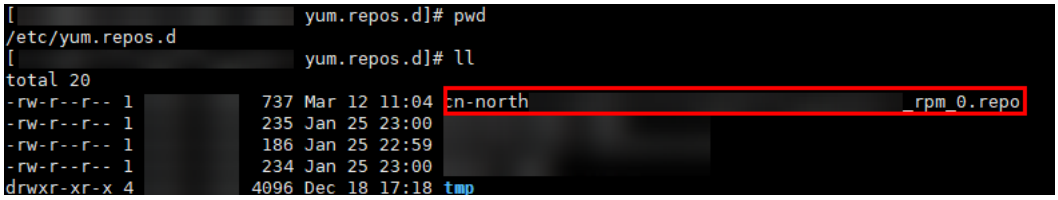
----End

Obtaining a Dependency from an RPM Repository

Step 1 Download the configuration file of the RPM repository by referring to **Publishing a Package to an RPM Repository**.

Step 2 Open the configuration file, replace all *{{component}}* in the file with the value of *{{component}}* (**hello** in this file) used for uploading the RPM file, delete the **RPM upload command**, and save the file.

Step 3 Save the modified configuration file to the **/etc/yum.repos.d/** directory on the Linux host.



Step 4 Run the following command to download the RPM package. Replace **hello** with the actual value of *component*.

```
yum install hello
```

----End

7 Uploading/Obtaining a Debian Package Using Linux Commands

Software developers often need to upload custom private packages to a Debian repository so that other team members can easily access and use them. However, without clear guidance, developers may encounter issues when uploading packages using Linux commands, leading to upload or dependency retrieval failures. Ensuring successful upload and access to private packages is therefore a common challenge. This document describes how to upload a private package to a Debian repository and fetch dependencies from the repository using Linux commands.

Prerequisites

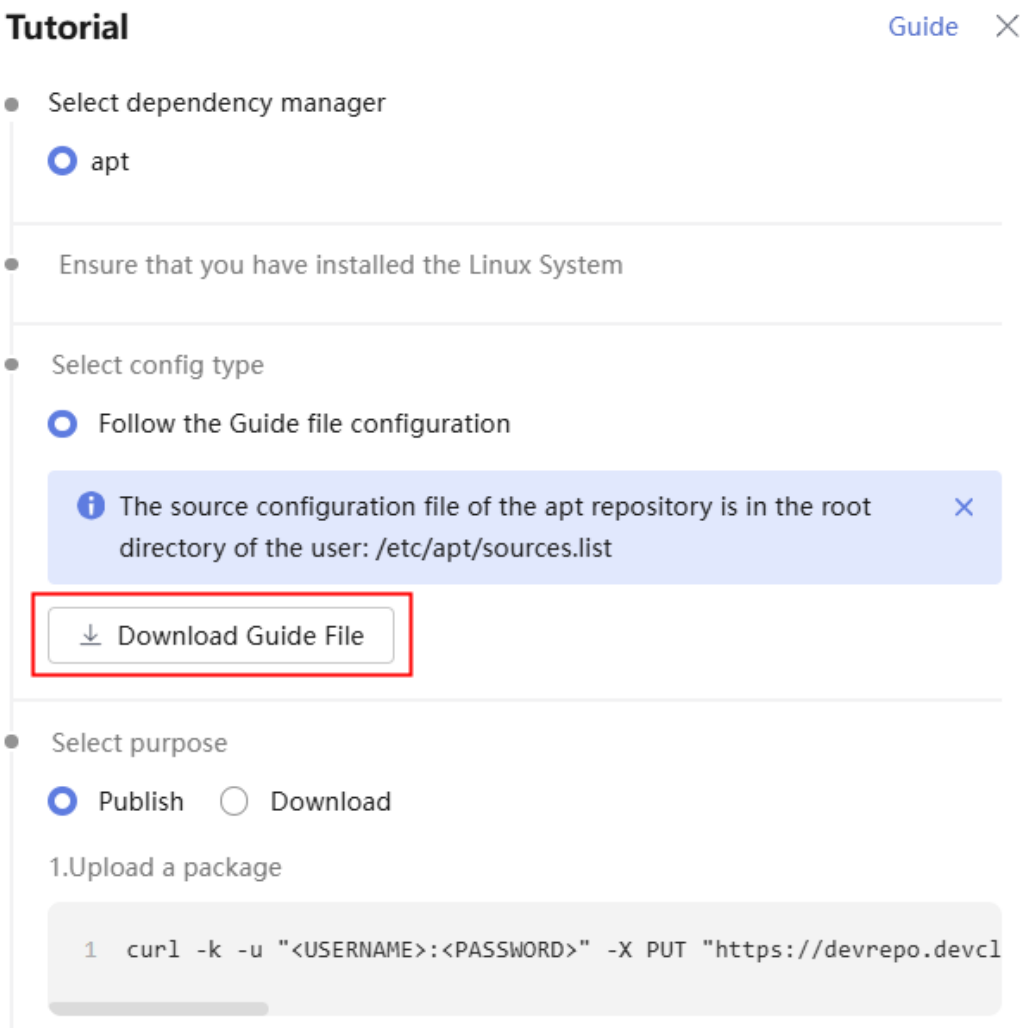
- You have a Debian package available.
- You have a Linux host that can connect to the Internet.
- You have created **a Debian repository**.
- You have been granted permissions to upload, download, and view packages in the current repository. For details, see **Configuring Repository Permissions**.

Publishing a Package to a Debian Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 Select the target Debian repository from the repository list. Click **Tutorial** on the right of the page.

Step 3 In the displayed dialog box, click **Download Guide File**.

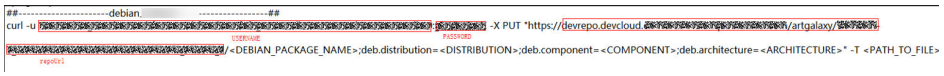


Step 4 On the Linux host, run the following command to upload a Debian package.

```
curl -u <USERNAME>:<PASSWORD> -X PUT "https:// <repoUrl>/
<DEBIAN_PACKAGE_NAME>;deb.distribution=<DISTRIBUTION>;deb.component=<COMPONENT>;deb.architecture=<ARCHITECTURE>" -T <PATH_TO_FILE>
```

In this command, *USERNAME*, *PASSWORD*, and *repoUrl* can be obtained from the **Debian upload command** in the configuration file downloaded in **Step 3**.

- *USERNAME*: username used for uploading files, which can be obtained from the Debian configuration file. For details, see the example figure.
- *PASSWORD*: password used for uploading files, which can be obtained from the Debian configuration file. For details, see the example figure.
- *repoUrl*: URL used for uploading files, which can be obtained from the Debian configuration file. For details, see the example figure.



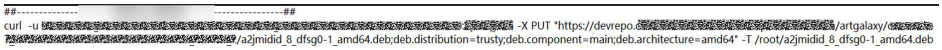
DEBIAN_PACKAGE_NAME, *DISTRIBUTION*, *COMPONENT*, and *ARCHITECTURE* can be obtained from the Debian package to be uploaded.

The **a2jmidid_8_dfsg0-1_amd64.deb** package is used as an example.

- *DEBIAN_PACKAGE_NAME*: software package name, for example, **a2jmidid_8_dfsg0-1_amd64.deb**.

- *DISTRIBUTION*: published version, for example, **trusty**.
- *COMPONENT*: component name, for example, **main**.
- *ARCHITECTURE*: system architecture, for example, **amd64**.
- *PATH_TO_FILE*: local storage path of the Debian package, for example, **/root/a2jmidid_8_dfsg0-1_amd64.deb**.

  The following figure shows the complete commands.



Step 5 After the commands are successfully executed, go to the self-hosted repo page and find the uploaded Debian package.

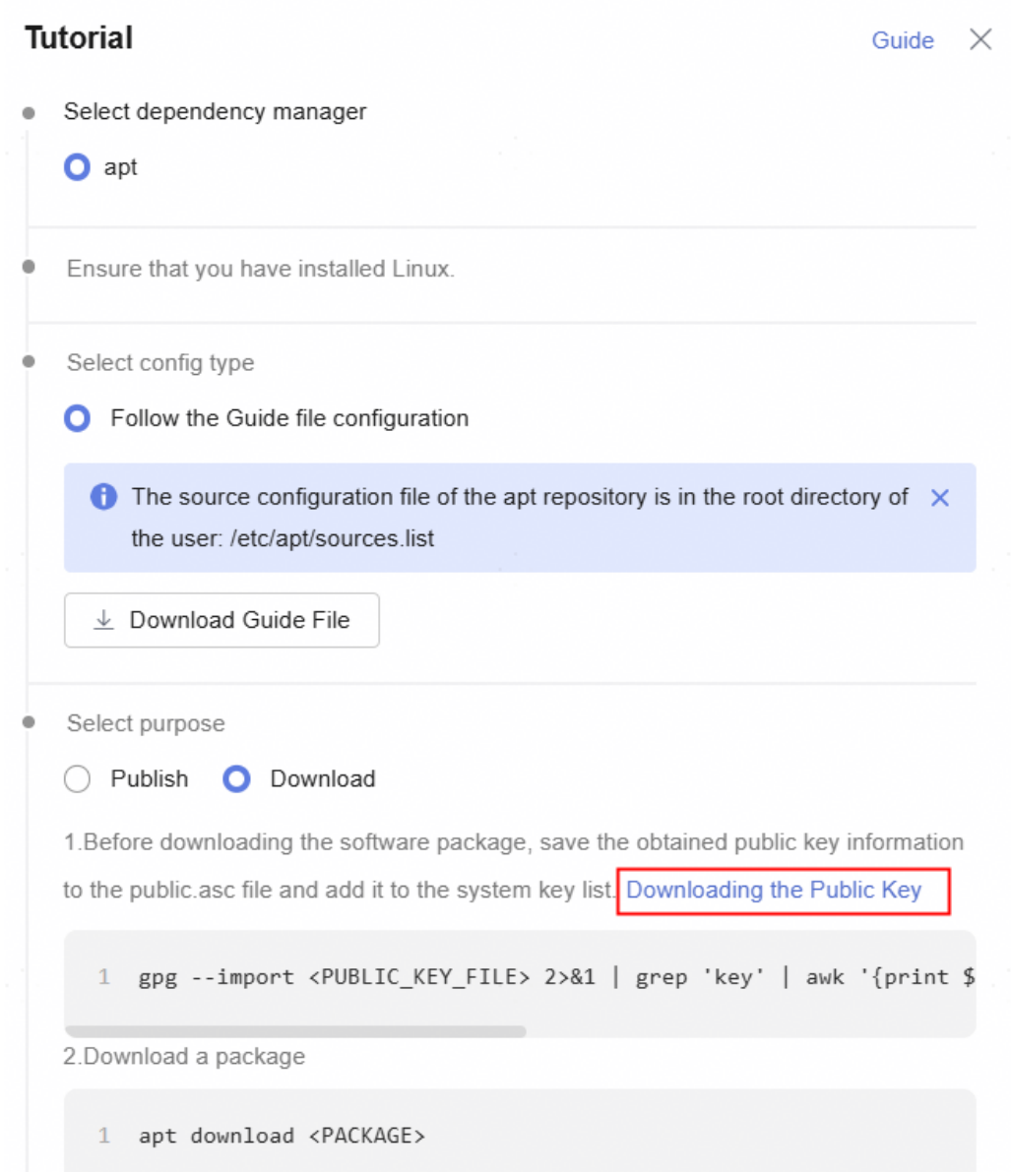
----End

Obtaining a Dependency from a Debian Repository

The following procedure uses the Debian package published in **Publishing a Package to a Debian Repository** as an example to describe how to obtain a dependency from a Debian repository.

Step 1 Go to the Debian repository by referring to **Publishing a Package to a Debian Repository**. Click **Tutorial** on the right of the page.

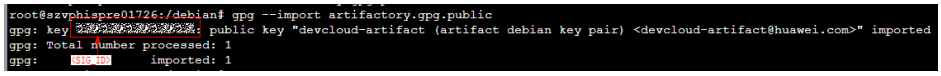
Step 2 In the displayed dialog box, set **Select purpose** to **Download** and click **Downloading the Public Key** to download the Debian repository public key.



Step 3 Import the **gpg** public key.

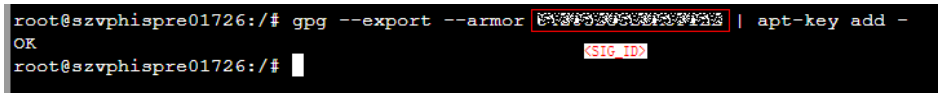
```
gpg --import <PUBLIC_KEY_PATH>
```

PUBLIC_KEY_PATH: local path for storing the Debian public key, for example, **artifactory.gpg.public**.



Step 4 Add the public key to the list of keys used by apt to authenticate packages.

```
gpg --export --armor <SIG_ID> | apt-key add -
```



Step 5 Add the apt repository source.

Open the configuration file (for details about how to obtain the file, see **Publishing a Package to a Debian Repository**), replace all *DISTRIBUTION* fields with the value of *COMPONENT* (for example, **main**) used for uploading the Debian file, and add the repository source based on the downloaded configuration file **sources.list**.

Step 6 After the repository source is added, run the following command to update the repository source.

```
apt-get update
```

Step 7 Run the following command to download the Debian package. Replace **a2jmidid** with the actual value of *PACKAGE*.

```
apt download a2jmidid
```

To obtain *<PACKAGE>*, perform the following steps.

Download the Packages source data of the Debian package. The following uses the **a2jmidid** package as an example.

----End

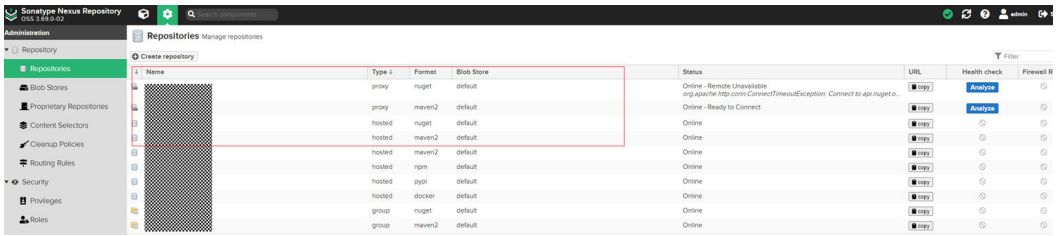
8 Migrating Repository Data from Nexus to CodeArts Artifact

8.1 Migration Preparations

In enterprise software development, teams often need to manage numerous dependencies, which may be spread across different repositories, such as hosted, proxy, and group repositories in Nexus. However, this distributed management complicates operations and reduces efficiency. How can we manage all repositories centrally and streamline operations? CodeArts Artifact provides a batch migration tool to quickly migrate hosted, proxy, and group repository data on Nexus to self-hosted repos. This streamlines operations and management for greater efficiency.

Confirming Repository and Package Types

The following uses Nexus3 as an example. Log in to Nexus3 and go to the administration page. Confirm the repository type (in the **Type** column) and package type (in the **Format** column) to be migrated, as shown in the following figure.



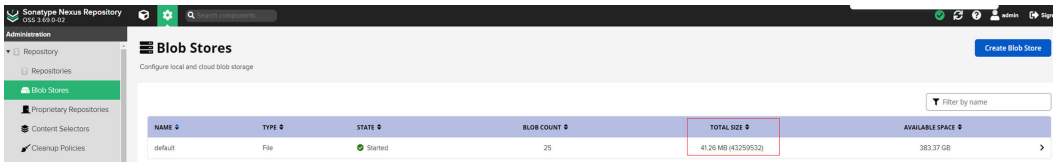
The migration method varies depending on the repository type (in the **Type** column).

- **hosted** repository: Perform operations in **Migrating Hosted Repository Data from Nexus to CodeArts Artifact**.
- **proxy** repository: Perform operations in **Migrating Proxy Repository Data from Nexus to CodeArts Artifact**.

- **group** repository: Perform operations in **Migrating Group Repository Data from Nexus to CodeArts Artifact**.

Confirming CodeArts Artifact Repository Capacity

Check the number and size of files displayed in **Blob Stores** (as shown in the following figure) to evaluate whether the remaining repository capacity of CodeArts Artifact meets the requirements.



Confirming Repository Usage Scenario

Repositories are provided for all projects. You are advised to create a project separately. For details about how to create a project, see **Creating a CodeArts Project**.

If projects are planned with separate repositories, you do not need to create a project.

8.2 Migrating Hosted Repository Data from Nexus to CodeArts Artifact

You can use the migration tool of CodeArts Artifact to migrate hosted repositories from Nexus to CodeArts Artifact. Migration tools work as follows: Read the packages on Nexus to the input stream, and then call the CodeArts Artifact APIs to upload the packages.

Prerequisites

- You have completed the operations in **Migration Preparations**.
- The runtime environment is JDK 8. Run the **java -version** command to verify your environment. For details about how to install JDK, see **https://www.java.com/en-us/**.
- The host running the migration tool is connected to both Nexus and CodeArts Artifact. That is, the PC can access the network addresses of Nexus and CodeArts Artifact. You can use either of the following methods to verify the connection:
  - Open a browser and enter the *Nexus domain name or IP address:port* and the CodeArts Artifact *repository address* (for details about how to obtain the repository address, see **Step 1** in **Step 3: Obtain Repository URL and Configuration File**).
  - Run the following commands:

    telnet *Nexus domain name or IP address: Port*

    telnet *Repository address of CodeArts Artifact* (For details about how to obtain the repository address, see **Step 1** in **Step 3: Obtain Repository URL and Configuration File**.)

Step 1: Identify the Original Repository to Migrate

Check the repository and package types of the repository to be migrated. If the repository is **hosted**, perform the operations in this section.

Step 2: Create a Local Repository in Self-Hosted Repos

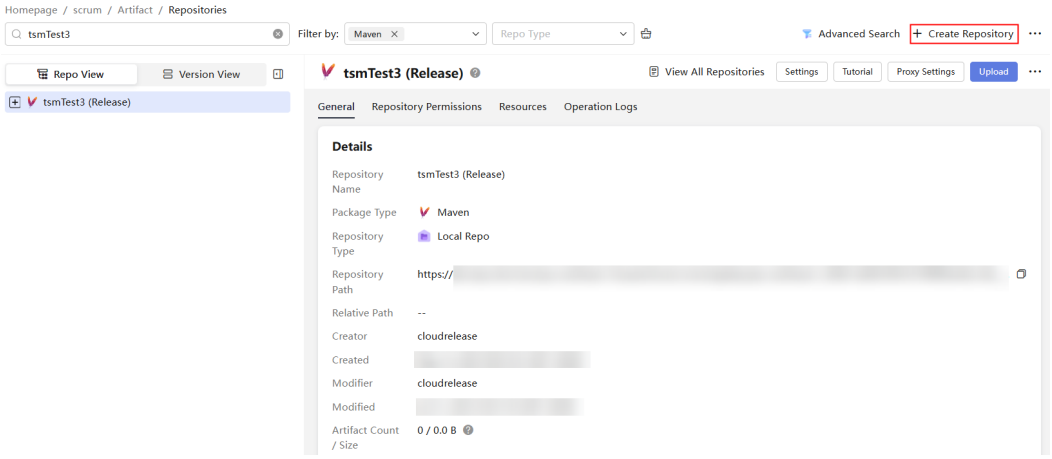
Create a repository based on the repository type in **Confirming Repository and Package Types**. The following describes how to create a local Maven repository.

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 On the **Self-hosted Repos** page, click **Create Repository** in the upper-right corner.

Step 3 Set **Repository Type** to **Local Repo**, **Package Type** to **Maven**, and **Project** to an existing project, and click **OK**. The created Maven repository is displayed in the **Repo View**.

Step 4 Click **Create Repository** in the upper-right corner to create another repository as required.

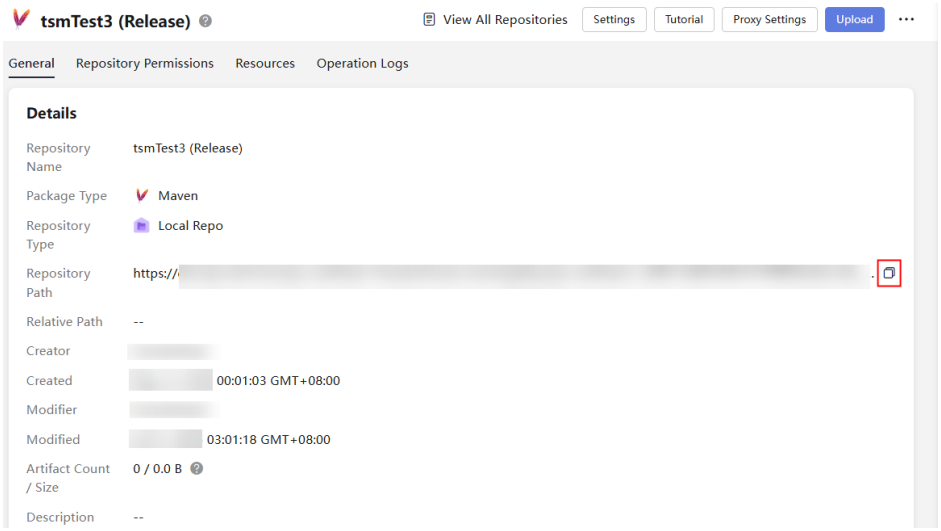


----End

Step 3: Obtain Repository URL and Configuration File

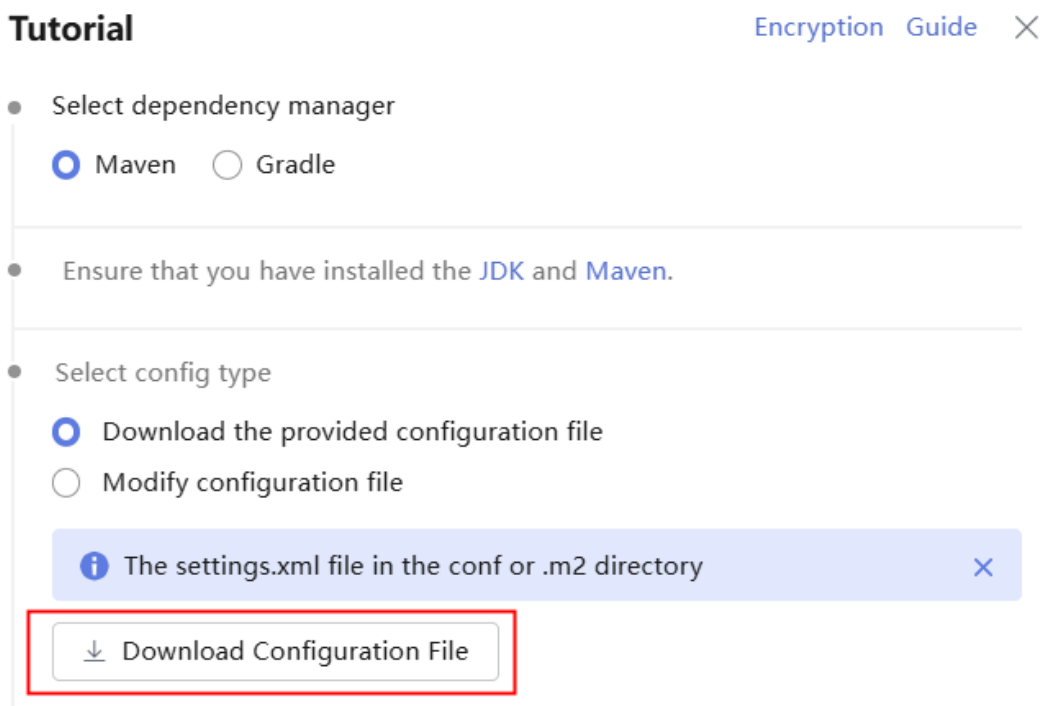
Step 1 Obtain the repository URL.

1. Find the repository created in **Step 2: Create a Local Repository in Self-Hosted Repos** and click **Repo View**.
2. Click the repository name. The **Repo URL** is displayed on the **General** tab. Click to copy the repository URL.

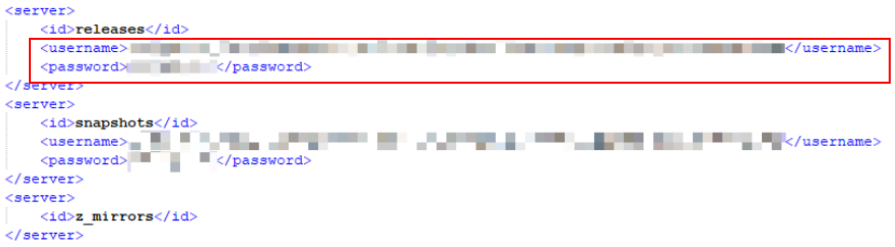


Step 2 Obtain the configuration file.

1. Click **Tutorial** on the right of the page.
2. In the **Tutorial** dialog box, click **Download Configuration File** to download the **settings.xml** file to the localhost.



3. Open the **settings.xml** file on the localhost, locate the <*username*> and <*password*> lines, and extract the username and password as shown in the following figure.



----End

Step 4: Configure Migration Tools for Local Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, click the target Maven repository name.

Step 3 Click ••• in the upper-right corner of the page, and select **Download Migration Tool** to download the tool package (migration tool name: **relocation-jfrog-20251016.1.jar**; configuration file: **application-nexus.yaml**) to the localhost.

Step 4 The following is an example of the **application-nexus.yaml** file. The example contains only mandatory parameters. You can directly search for the parameter name in the file. These parameters are under **relocation**.

Table 8-1 Key configurations in **application.yaml**

Parameter	Example Value	Description
name	nexus-to-artifact	Name of the migration task (only for display).
package_type	maven	Migration package type: maven, npm, PyPI, and Go **NOTICE** Maven repositories are classified into Release and Snapshot repositories. If the original repositories are mixed, you need to migrate the original repositories twice to migrate Release and Snapshot repositories to different Maven repositories.
migrate_type	nexus3	Migration type. The value can be **nexus3** or **nexus2**, depending on the Nexus version.
save_temp_dir	D:/tmp/xxx/	Cache path, which is mandatory for migrating Maven. The path ends with a slash (/). The last uploaded snapshot version of the Maven is stored in this path.
domain	http://{ip}:{port}	Domain name of the original repository. The path must not end with a slash (/).

Parameter	Example Value	Description
repo	test_maven	Original repository name, which is the repository name on Nexus. You can obtain the name of the repository to be migrated from the **NAME** field on the Nexus page.
user_name	username	Original repository username.
password	password	Original repository password.
target_domain	https://{domain}/artgalaxy https://{domain}/artgalaxy/api/npm	Domain name of the target repository. Obtain it from the repository URL on the self-hosted repo page in **Step 3: Obtain Repository URL and Configuration File**. • If the repository URL is https://{domain}/artgalaxy/*xx*-north-*xxx_xxxxxxxx*_maven_1_388/ enter **https://{domain}/artgalaxy**. • If the repository URL is https://{domain}/artgalaxy/api/npm/*xx*-north-*xx_xxxxxxx*_npm_6944/ enter **https://*{domain}*/artgalaxy/api/npm**.
target_repo	*xx*-north-*xxx_xxxxxxxx*_maven_1_388 *xx*-north-*xx_xxxxxxx*_npm_6944	Target registry name. Obtain it from the repository URL in **Step 3: Obtain Repository URL and Configuration File**. • If the repository URL is https://{domain}/artgalaxy/xx-north-*xxx_x*xxxxxxx*_maven_1_388/ enter *xx*-north-*xxx_xxxxxxxx*_maven_1_388. • If the repository URL is https://{domain}/artgalaxy/api/npm/*xx*-north-*xx_xxxxxxx*_npm_6944/ enter *xx*-north-*xx_xxxxxxx*_npm_6944.
target_user_name	username	Username of the target repository, which can be obtained from **Step 3: Obtain Repository URL and Configuration File**.

Parameter	Example Value	Description
target_password	password	Password of the target repository, which can be obtained from **Step 3: Obtain Repository URL and Configuration File**.

Simple **application-nexus.yaml**

```
spring:
  main:
    web-application-type: none
relocation:
  # Name of the migration task (only for display)
  name: nexus-to-artifact
Migration package type: maven, npm, PyPI, and Go
  package_type: maven
# Migration type. Governance, JFrog, Nexus3, and Nexus2 are available.
  migrate_type: "nexus3"
# Cache path, which is mandatory for migrating Maven. The path ends with a slash (/). The last uploaded snapshot version of the Maven is stored in this path.
  save_temp_dir: "D:/tmp/xxx/"

  # Original repository parameter. It is mandatory only in the JFrog and Nexus scenarios.
  # Original repository URL. The URL must not end with a slash (/).
  domain: 'http://{domain}/artifactory'
  # Original repository name
  repo: 'test-maven'
  # Original repository username
  user_name: "username"
  # Original repository password
  password: "password"

  # Target repository parameter
  # Target repository URL. The path must not end with a slash (/). Obtain the path from the page.
  target_domain: 'https://{domain}/artgalaxy/xxxx/xxxx'
  # Target repository name
  target_repo: xxxxx
  # Target repository username
  target_user_name: 'username'
  # Target repository password
  target_password: 'password'
```

Step 5 Configure parameters in the **application.yaml** file.

```
spring:
  main:
    web-application-type: none
relocation:
  # General configuration of the migration program
  # Number of core threads of the migration program
  corePoolSize: 20
  # Maximum number of thread pools of the migration program
  maxPoolSize: 40
  # Thread pool queue size of the migration program
  queueCapacity: 99999999
  # Maximum migration speed of the migration program (MB/s). The default value is 20.
  speed_limit: 20
  # JFrog migration scenario parameter. This parameter indicates the migration packages between the time of modifiedFrom and modifiedTo (timestamp).
  modifiedFrom:
  modifiedTo:
  # Name of the migration task (only for display)
  name: jfrog-to-artifact
  # Package type of the repository to be migrated
```



```
  package_type: pypi
  # Migration type. Governance, JFrog, and Nexus are available.
  migrate_type: "jfrog"
  # Cache path, which is mandatory for migrating Maven. The value ends with a slash (/).
  save_temp_dir: "/xxxx/"

  # Original repository parameter. It is mandatory only in the JFrog and Nexus scenarios.
  # Original repository URL. The URL must not end with a slash (/).
  domain: 'http://{domain}/artifactory'
  # Original repository name
  repo: 'test-pypi-source'
  # Original repository type. The default value is artifactory.
  repo_type: artifactory
  # Original repository username
  user_name: "username"
  # Original repository password
  password: "password"
  # In the JFrog scenario, enter the sub-path of the original repository. Sub-path migration is supported.
  source_sub_path:

  # Target repository parameter
  # Target repository URL. The path must not end with a slash (/). Obtain the path from the page.
  target_domain: 'https://{domain}/artgalaxy/xxxx/xxxx'
  # Target repository name
  target_repo: xxxxx
  # Target repository type. The default value is artifactory.
  target_repo_type: artifactory
  # Target repository username
  target_user_name: 'username'
  # Target repository password
  target_password: 'password'

  # Governance migration parameters (ignore for Python migration)
# domain_id in the governance migration scenario
  domain_id: test009
  # Whether to call CodeArts Governance in the governance scenario. If the value is true, CodeArts
Governance is not called.
  call_governance_use_local: true
  # Package type of the governance migration repository
  governance_type: npm
  # Path for storing the CodeArts Governance entity package.
  migrate_local_path: ""
  # Governance metadata file, indicating the metadata file to be migrated
  migrate_metadata_path: ""
  # Callback governance path. An empty path needs to be set for this path.
  governance_save_path: ""
  #  Callback CodeArts Governance URL
  governance_url: ""
  # User AK. It is used to call back the CodeArts Governance API.
  access_key_id: ""
  # User SK. It is used to call back the CodeArts Governance API.
  secret_access_key: ""
  # Whether to call back CodeArts Governance only through the cache information in
governance_save_path.
  only_update_governance: false
  # Maximum number of files to be migrated in the CodeArts Governance. The number is the number of
metadata records in migrate_metadata_path.
  migrate_max_num: -1
```

----End

Step 5: Migrate Data

Step 1 Run the following migration script:

```
java -jar relocation-jfrog-20251016.1.jar --spring.config.location=application-nexus.yaml > /log/relocation-jfrog.log 2>&1 &
```


Step 2 Go to the target self-hosted repo (local repository) and check whether the package is successfully uploaded to the **hosted** repository.

----End

8.3 Migrating Proxy Repository Data from Nexus to CodeArts Artifact

To migrate proxy repository data from Nexus to CodeArts Artifact, you only need to configure proxy repositories on CodeArts Artifact. Then you can use the new proxy repository.

The principle is as follows: Configure proxy settings for open-source repositories or third-party repositories to replace proxy repositories on Nexus.

Prerequisites

You have completed the operations in **Migration Preparations**.

Procedure

Step 1 **Add a proxy to the virtual repository in the self-hosted repo.**

Step 2 Use your Huawei Cloud account to access **self-hosted repos**.

Step 3 In the left pane, select the repository where the proxy is configured.

----End

8.4 Migrating Group Repository Data from Nexus to CodeArts Artifact

To migrate group repository data from Nexus to CodeArts Artifact, you only need to configure virtual repositories on CodeArts Artifact. Then you can use the new virtual repository.

The principle is as follows: Aggregate the local and proxy repositories through configuration to replace the group repository on Nexus.

Prerequisites

- You have completed the operations in **Migration Preparations**.
- You have created a virtual repository. For details, see **Creating a Repository**.
- You have configured the network for the proxy mirror repository. You can contact the environment administrator, O&M personnel, or technical support.

Step 1: Migrate the Proxy and Hosted Repositories in the Group Repository in the Nexus

Migrate the hosted and proxy repositories in a group repository in the Nexus by referring to **Migrating Hosted Repository Data from Nexus to CodeArts**

Artifact and **Migrating Proxy Repository Data from Nexus to CodeArts Artifact**.

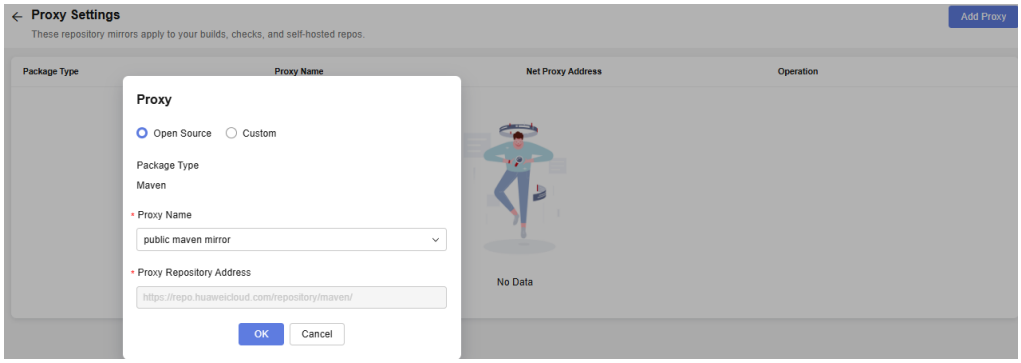
Step 2: Configure a Virtual Repository

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, select the target virtual repository.

Step 3 Click **Proxy Settings** in the upper-right corner of the page.

Step 4 Click **Add Proxy** and select **Open Source** or **Custom**.



You can select **Third-party Repository** or **Huawei Local Repository** from **Custom**.

- **Third-party repository**: Set a third-party repository or a repository created by a user as the proxy source.

  After selecting a third-party repository, select a custom proxy source (proxy on Nexus) from the **Proxy Name** drop-down list.
- **Huawei Local Repository**: Set a Huawei local repository as the proxy source. Only repositories with access permissions can be selected. The local repository corresponds to the hosted repository in Nexus.

  You can select a local repository from the **Proxy Name** drop-down list.

Step 5 Click **OK**. The proxy is added.

- In the proxy list, click in the **Operation** column to change the proxy name, proxy username, and proxy password.

  NOTE

  You cannot edit the proxy source of the local repository.
- Click in the **Operation** column to delete the proxy.

----End

9 Migrating Local Repository Data to CodeArts Artifact

9.1 Migrating Local Maven Repository Data to CodeArts Artifact

A local repository is a copy of software packages or dependencies stored on your computer. When you use Maven to manage dependencies, the tools download dependencies from remote repositories to your local repository. However, CodeArts Artifact enforces strict permissions for storing, pushing, and pulling dependencies and final products generated during development to support effective team collaboration. Therefore, after migrating Maven repository data from your local disk to CodeArts Artifact, you can manage operations and maintenance more efficiently in one place. To meet this need, CodeArts Artifact provides a tool to help you quickly migrate Maven repository data in batches to the Maven repository in self-hosted repos.

Constraints

Only local Maven and npm repository data can be migrated to self-hosted repos in CodeArts Artifact. This section describes how to migrate local Maven repository data to self-hosted repos in CodeArts Artifact.

Prerequisites

- You already have a project. If no project is available, **create one**.
- You have permissions for the current Maven repository. For details, see **Configuring Repository Permissions**.
- You have created **a Maven repository** in self-hosted repos.
- You have the Python3 environment available.
- The local PC running the migration tool is connected to CodeArts Artifact. That is, the PC can access the network address of CodeArts Artifact. You can use either of the following methods to verify the connection:
  - Open a browser and access the *repository address* of CodeArts Artifact. (For details about how to obtain the repository address, see steps **1** to **3**

in **Step 1: Obtain Target Maven Repository Information from CodeArts Artifact**.)

- Run the following command:

  telnet *Repository address of CodeArts Artifact* (For details about how to obtain the repository address, see steps **1** to **3** in **Step 1: Obtain Target Maven Repository Information from CodeArts Artifact**.)

Step 1: Obtain Target Maven Repository Information from CodeArts Artifact

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, click the target Maven repository name to go to its details page, and view the **Repo URL**.

Step 3 Click next to the repository URL to copy it.

Step 4 Click **Tutorial** in the upper-right corner of the page. In the displayed dialog box, click **Download Configuration File** to download the **settings.xml** file to the localhost.

Open the file on the localhost and find the username and password.

----End

Step 2: Configure Migration Tool

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, select the target Maven repository.

Step 3 Click the repository name. In the upper-right corner of the page, click ••• and select **Download Migration Tool** to download the **MigrateTool.zip** package to the localhost. Then decompress it to obtain **uploadArtifact.py** (migration tool) and **artifact.conf** (configuration file).

Step 4 Configure the **artifact.conf** file using the example below. Only the required parameters are listed. Other parameters can be deleted.

[artifact]

packageType = Component type. Set it to Maven.

userInfo = username:password (username and password obtained in **Step 4** of **Step 1: Obtain Target Maven Repository Information from CodeArts Artifact**)

repoRelease = Repository URL (repository URL obtained in **Step 3** of **Step 1: Obtain Target Maven Repository Information from CodeArts Artifact**)

repoSnapshot = Repository URL (repository URL obtained in **Step 3** of **Step 1: Obtain Target Maven Repository Information from CodeArts Artifact**). This parameter is involved when the Maven artifact type is Snapshot.

srcDir = Directory path of the local Maven repository to be migrated. The value is user-defined, for example, **C:\Users***xxxxxx***\repository**.

----End

Step 3: Migrate Data

Run the migration tool obtained in **Step 3** by executing this command:

```
python uploadArtifact.py
```

Step 4: Verify Migration Results

Go to the target Maven repository in self-hosted repos and verify whether the local Maven repository data was uploaded successfully.

If the migration fails, try again or contact customer service.

9.2 Migrating Local npm Registry Data to CodeArts Artifact

A local repository is a copy of software packages or dependencies stored on your computer. When you use npm to manage dependencies, the tools download dependencies from remote repositories to your local repository. However, CodeArts Artifact enforces strict permissions for storing, pushing, and pulling dependencies and final products generated during development to support effective team collaboration. Therefore, after migrating npm registry data from your local disk to CodeArts Artifact, you can manage operations and maintenance more efficiently in one place. To meet this need, CodeArts Artifact provides a tool to help you quickly migrate npm registry data in batches to the npm registry in self-hosted repos.

Constraints

Only local Maven and npm repository data can be migrated to self-hosted repos in CodeArts Artifact. This section describes how to migrate local npm registry data to CodeArts Artifact.

Prerequisites

- You already have a project. If no project is available, **create one**.
- You have permissions for the current Maven repository. For details, see **Configuring Repository Permissions**.
- You have created **an npm registry** in self-hosted repos.
- You have the Python3 environment available.
- The local PC running the migration tool is connected to CodeArts Artifact. That is, the PC can access the network address of CodeArts Artifact. You can use either of the following methods to verify the connection:

- Open a browser and access the *repository address* of CodeArts Artifact. (For details about how to obtain the repository address, see steps **1** to **3** in **Step 1: Obtain Target npm Registry Information from CodeArts Artifact**.)
- Run the following command:

  telnet *Repository address of CodeArts Artifact* (For details about how to obtain the repository address, see steps **1** to **3** in **Step 1: Obtain Target npm Registry Information from CodeArts Artifact**.)

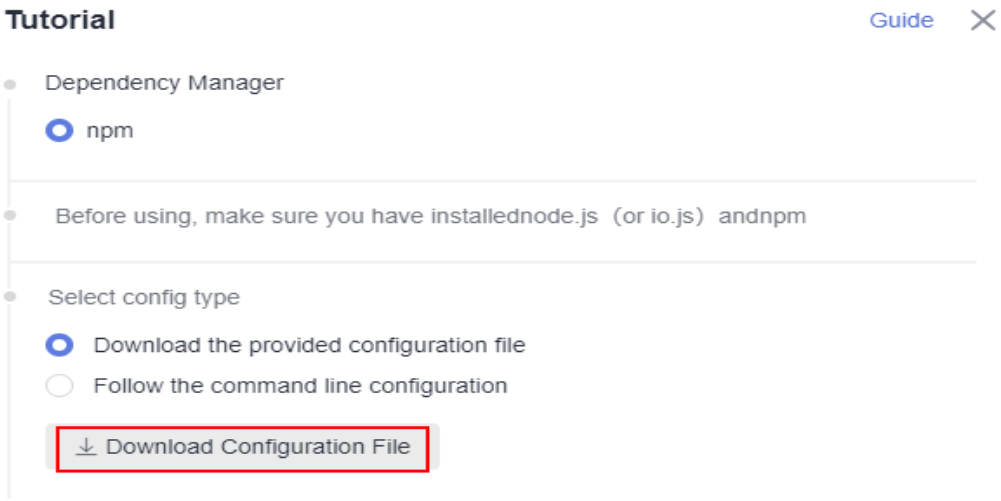
Step 1: Obtain Target npm Registry Information from CodeArts Artifact

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, click the target npm registry name to go to its details page and view the **Repo URL**.

Step 3 Click next to the repository URL to copy it.

Step 4 Click **Tutorial** in the upper-right corner of the page. In the displayed dialog box, click **Download Configuration File** to download the **npmrc** configuration file to the localhost.



Step 5 Open the configuration file on the localhost, find the value of the **_auth** field, and decode the value using Base64.

----End

Step 2: Configure Migration Scripts

Step 1 Use your Huawei Cloud account to access **self-hosted repos**.

Step 2 In the left pane, select the target npm registry.

Step 3 Click the repository name. In the upper-right corner of the page, click and select **Download Migration Tool** to download the **MigrateTool.zip** package to the localhost. Then decompress it to obtain **uploadArtifact.py** (migration tool) and **artifact.conf** (configuration file).

Step 4 Configure the **artifact.conf** file using the example below. Other parameters can be deleted.

[artifact]

packageType = Component type. Set it to npm.

userInfo = Value of the _auth field decoded using Base64 in the npmrc configuration file of the npm registry. (For details, see **Step 5** in **Step 1: Obtain Target npm Registry Information from CodeArts Artifact**.)

repoRelease = Repository URL (repository URL obtained in **Step 3** of **Step 1: Obtain Target npm Registry Information from CodeArts Artifact**)

repoSnapshot = Left empty

srcDir = Directory path of the local npm registry to be migrated. The value is user-defined, for example, **C:\Users*xxxxxx*\repository**.

----End

Step 3: Migrate Data

Run the migration tool obtained in **Step 3** by executing this command:

```
python uploadArtifact.py
```

Step 4: Verify Migration Results

Go to the target npm registry in self-hosted repos and verify whether the local npm registry data was uploaded successfully.

If the migration fails, try again or contact customer service.

10 Configuring CodeArts Artifact Permissions

Overview

CodeArts Artifact provides two repository types: release repos and self-hosted repos. Release repos support project-level permission management, while self-hosted repos support both project-level and repository-level permission management. For details, see **Configuring Permissions for Release Repos** and **Configuring Permissions for Self-Hosted Repos**.

This practice uses self-hosted repos as an example to describe how to quickly manage permissions for individual repositories and by project.

Constraints

- **Custom roles** created in CodeArts Artifact do not have preset permissions. You can contact the project administrator to configure roles and permissions on corresponding resources.
- By default, the project administrator, project manager, and test manager can assign permissions for other roles in self-hosted repos. If other roles can assign permissions, they can continue to manage permissions for other roles in self-hosted repos.

Prerequisites

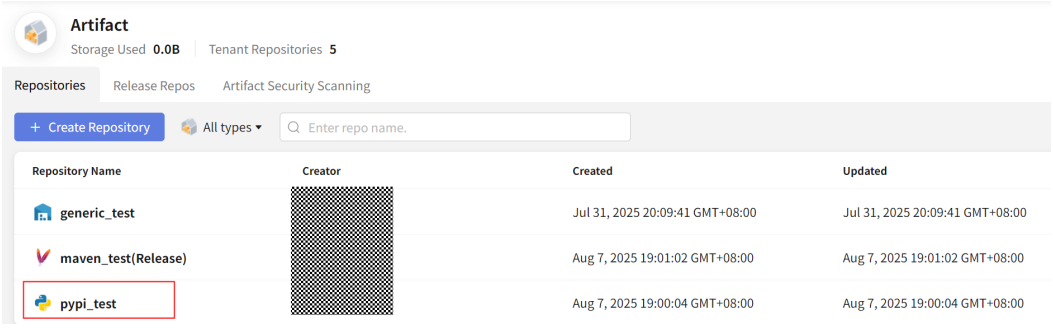
- You have created a **CodeArts project** (Select the **Scrum** template and name it **Scrum**.)
- You have added users to the CodeArts project **Scrum** and assigned roles to them. For details, see **Adding Project Members**.
- To assign permissions to other roles in the self-hosted repo, you must have the **Privilege Config** permission. By default, the project administrator, project manager, and test manager roles have this permission.
- You have created a self-hosted repo named **pypi_test** in the Scrum project. For details, see **Creating a Repository**.

Configuring Project-Level Permissions

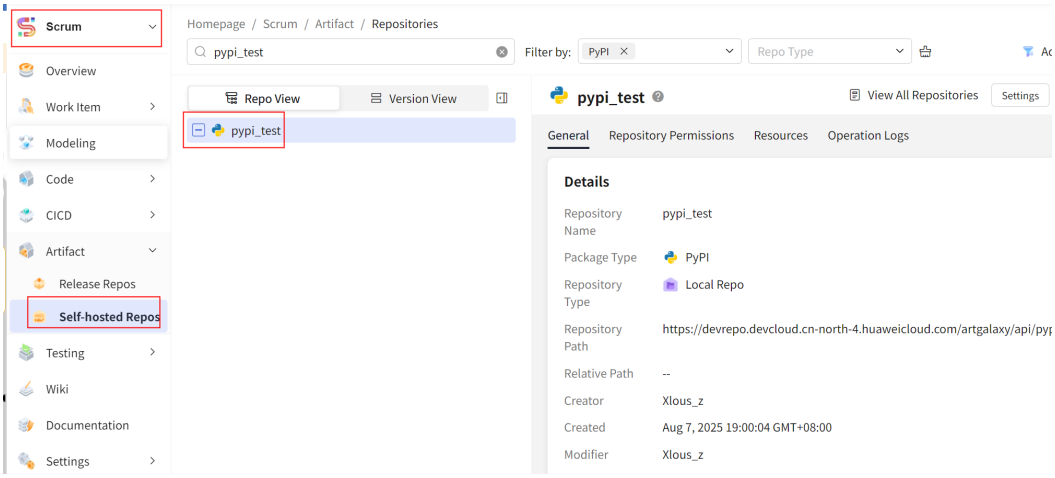
CodeArts Artifact allows you to configure permissions on self-hosted repos for each role in a project. For details, see **Configuring Project-Level Permissions**.

Step 1 Log in as a user with the **Privilege Config** permission, and access the **self-hosted repo**.

Step 2 Click the **Repositories** tab. All self-hosted repos created are displayed.

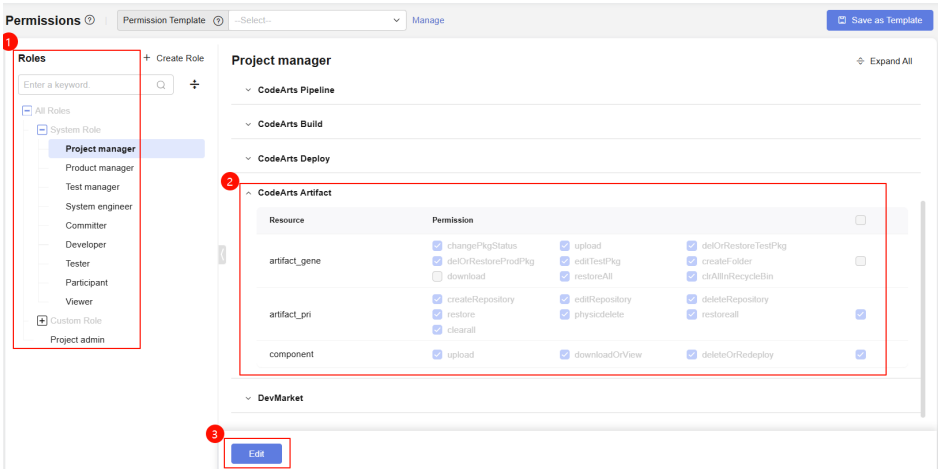


Step 3 In the repository list, click **pypi_test** to go to its details page of the project, as shown in the following figure.

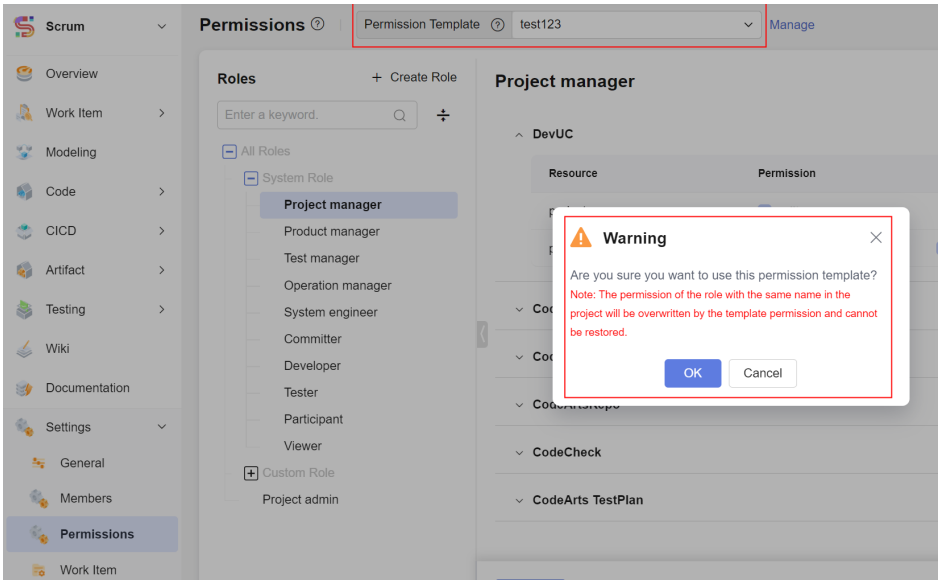


Step 4 Choose **Settings** > **General** > **Permissions** from the navigation pane. The **Permissions** page is displayed. You can configure project-level permissions in either of the following ways:

- Method 1: In the **Roles** area, click the role you want to configure permissions, select **CodeArts Artifact** on the right, and click **Edit** at the bottom of the page. On the displayed page, select or deselect the required permissions, and then click **Save**.



- Method 2: Select the template to apply from the **Permission Template** drop-down list on the top of the page (for details about how to create and manage permission templates, see **Managing Project Permission Templates**). In the displayed dialog box, click **OK**. The permissions from the template are then applied in one click, as shown in the following figure. **After a permission template is applied, the permissions of roles with the same name in the project will be overwritten and cannot be restored. Proceed with caution.**



----End

Configuring Repository-Level Permissions

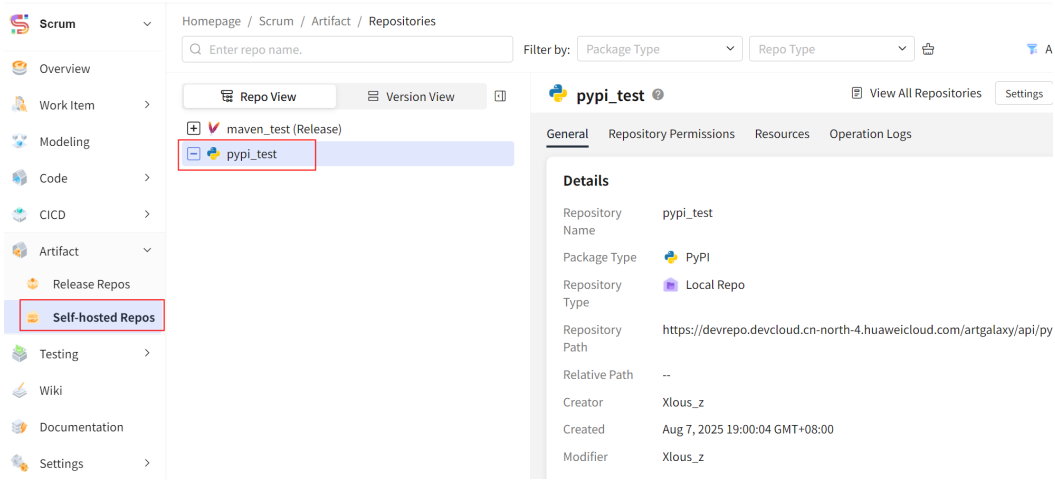
CodeArts Artifact allows you to configure permissions on all self-hosted repos in a project. For details, see **Configuring Project-Level Permissions**. You can also configure permissions for a single self-hosted repo.

- By default, new self-hosted repos inherit role permissions from **Settings** > **Permissions** in the project. Any changes made to these role permissions in **Configuring Project-Level Permissions** will be synced to the repo's permissions.

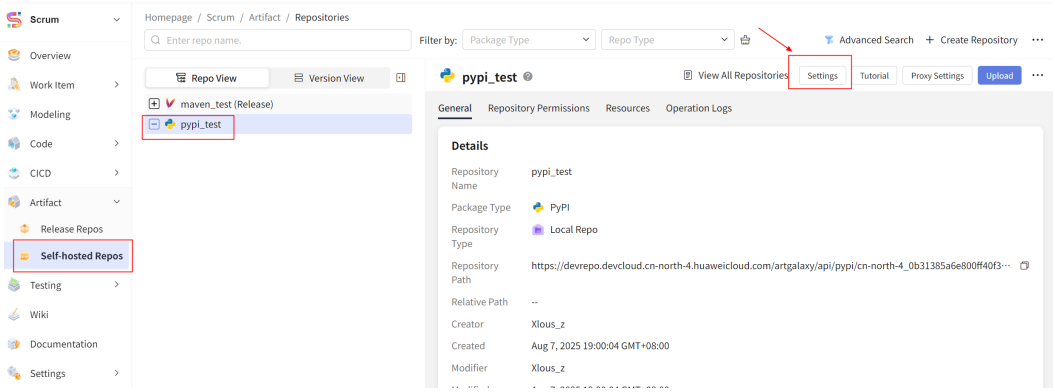
- If you do not change the repository permissions of a role in the self-hosted repo, any changes made on **Configuring Project-Level Permissions** will be synced to the repo's permissions as well.
- If you change the repository permissions of a role directly in the self-hosted repo, any changes made on **Configuring Project-Level Permissions** will not be synced to the repo's permissions. You need to change the permissions of the role in the repo.

Step 1 **Access the self-hosted repo**.

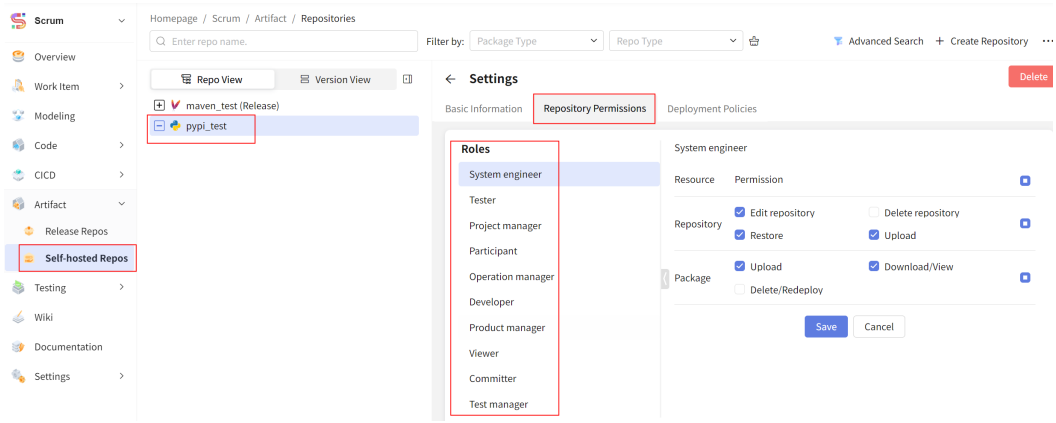
Step 2 Select the target repository from the repository list.



Step 3 Click **Settings** on the right of the page.



Step 4 Click the **Repository Permissions** tab. The roles in the current project are displayed.



Step 5 In the **Roles** area, select or deselect the desired permissions for the target role.

- By default, new self-hosted repos inherit role permissions from **Settings** > **Permissions** in the project. Any changes made to these role permissions in **Configuring Project-Level Permissions** will be synced to the repo's permissions.
- If you do not change the repository permissions of a role in the self-hosted repo, any changes made on **Configuring Project-Level Permissions** will be synced to the repo's permissions as well.
- If you change the repository permissions of a role directly in the self-hosted repo, any changes made on **Configuring Project-Level Permissions** will not be synced to the repo's permissions. You need to change the permissions of the role in the repo.

Step 6 Click **Save** to complete the repository-level permission configuration.

Each role can access the **self-hosted repo** and perform operations specified by their permissions.

----End

