CodeArts Artifact

User Guide

Issue 03

Date 2023-08-07

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 Release Repo (New Version)

1.1 Overview

CodeArts Artifact is a general, unified repository used to manage software artifacts in different formats. In addition to basic storage functions, it also provides important functions such as build and deployment tool integration, version control, access permission control. It is a standardized way for enterprises to process all artifact types generated during software development.

Operations pertaining to a release repo include:

- Basic operations: You can upload, download, edit, search for, and delete software packages, as well as create, edit, search for, and delete folders.
- Viewing and editing software package details: Software package details include basic, constructing metadata, and build packages. The folder name, software package name and status, and release version are editable.
- Managing the recycle bin: After a software package is deleted, it is moved to the recycle bin. You can restore the package to the folder before the deletion or permanently delete the package from the recycle bin.

1.2 Accessing a Release Repo

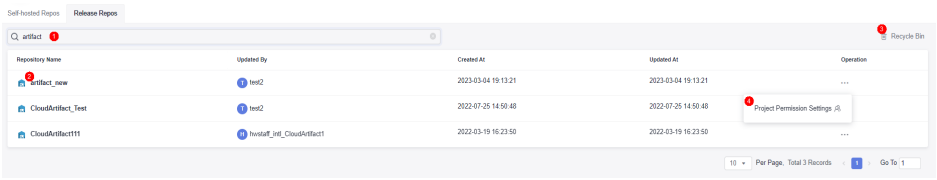
You can access the release repo page in either of the following ways: homepage entry and project entry.

Accessing Through the Homepage

Step 1 Log in to CodeArts.

Step 2 Choose **Services** > **Artifact**.

Step 3 View the project name list of the current tenant. You can perform the following operations as required:



No.	Operation	Description
1	Search for repos	Enter the project name in the search box to find the existing software release repo of the project.
2	View folder details	Click any folder to view the list of archived software packages in the folder or folders. You can upload, download, and edit software packages or folders.
3	Manage recycle bin	Click **Recycle Bin** to go to the recycle bin page. You can delete or restore the software package or folder as required.
4	Set project permissions	Click ··· to go to the **Set Project Permissions** page and edit member permissions. For details, see **1.7 Setting Permissions**.

NOTE

On the **Release Repos** page, you will view a project list, but you cannot upload files or create folders there. To perform such operations, click a project name to access the specific project first.

----End

Accessing Through a Specific Project

Step 1 Log in to the CodeArts homepage and click a card to access a project.

Step 2 Choose **Artifact** > **Release Repos**.

Step 3 The list of software packages and folders archived in the current project is displayed.

Perform the operations described in the following sections as required.

----End

1.3 Performing Basic Operations

Follow instructions in **Accessing Through a Specific Project** to access the release repo homepage. You can upload, create, download, edit, search for, and delete software packages.

Viewing Basic Information About Release Repos

Step 1 On the release repo page, click **Settings** in the upper right corner of the page.

Step 2 The repository name, artifact type, and description of the release repo are displayed.

- The release repo's name is the same as that of the project and cannot be modified.
- The description of release repos is synchronized with that of the project, as shown in the following figure. Click **here** to go to the basic project information page and modify the description.

----End

Creating a Folder

Step 1 On the **Release Repos** tab page, click **Create Folder** on the right of the page to create a folder. Folders can be nested.

Click to change a folder name.

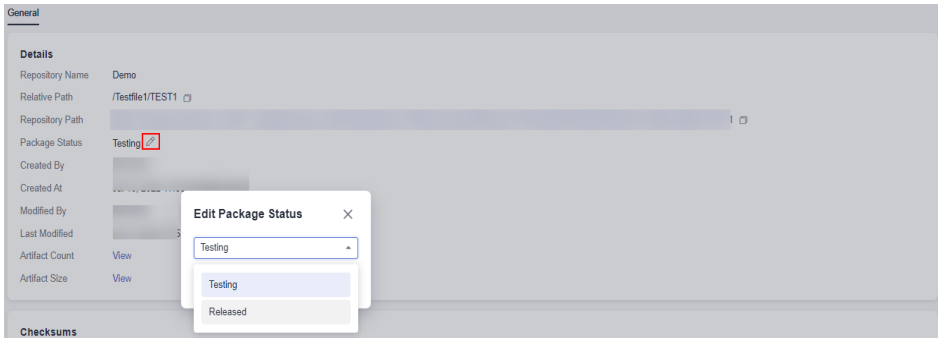
Click to delete a folder and software package in the folder. You can choose to permanently delete the folder or move the deleted folder to the recycle bin.

Step 2 Select a folder and click **Create Folder** to create a level-2 folder.

----End

Setting Status

Step 1 After entering a level-1 folder, you can click next to **Package Status** and select a status from a drop-down list to change the status of a level-2 folder (**Testing** by default).



If the folder status is **Released**, the folder cannot be changed or edited (changing the folder name, changing the file name in the folder, uploading the folder, changing the version number, or creating a subfolder). You can only download or delete it.

NOTICE

The folder status can be changed from **Not Released** to **Released**, but such change is irreversible. Exercise caution when performing this operation.

----End

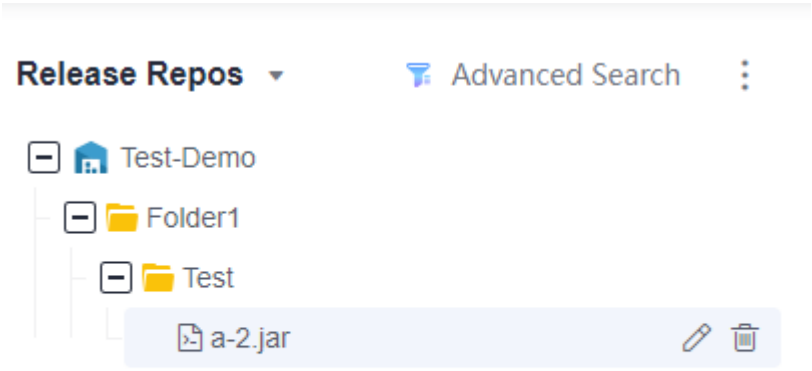
Uploading a Software Package

Step 1 Click **Upload** in the upper right corner of the page to manually upload local software packages to the release repo.

After selecting a folder, click **Upload** to manually upload a local software package to the corresponding folder.

Step 2 Click to change a software package name.

Click to delete the software package permanently or move it to the recycle bin.



NOTE

You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to the release repo.

----End

Searching

Step 1 Access the release repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Enter a keyword (a folder or file name) in the search box on the left of the page to search for the software package whose name contains the keyword.

Step 3 Click a file name to go to the file details page.

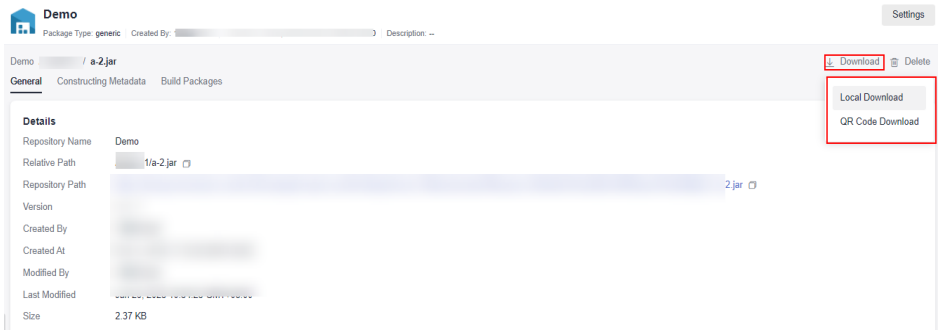
----End

Downloading a Software Package

Scenario 1

Step 1 Follow instructions in **Accessing Through a Specific Project** to access the release repo, select the software package to be downloaded, and click **Download**.

Step 2 In the displayed dialog box, select a method to download.



- **Local Download**: Download the software package to a local PC.
- **QR Code Download**: Use a mobile phone to scan the QR code to download the file.

----End

Scenario 2

Step 1 Access the release repo through **Accessing Through a Specific Project** and select the software package to be downloaded.

Step 2 Click  on the right of the software package.

----End

1.4 Viewing or Editing Software Package Details

On the release repo page, you can view or edit software package details, including basic information, constructing metadata information, and build packages.

Follow instructions in **Accessing Through a Specific Project** to access the release repo and click the software package name. The details about the selected software package are displayed. The software package details are displayed on three tab pages: **Basic Information**, **Constructing Metadata**, **Build Packages**.

- **Basic Information**: displays the repository name, relative path, download address, released version, creator, creation time, size, and checksum.

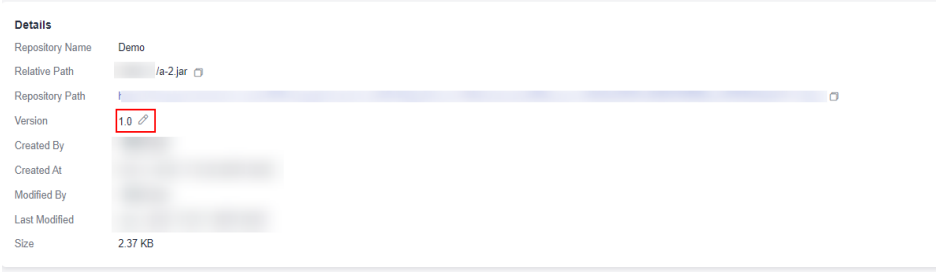
  Click  to change the release version of the software package. (The release version archived by CodeArts Build is the build sequence number by default.)
- **Constructing Metadata**: displays the build task, size, sequence number, builder, code repo, and code branch of the generated software package. Click **Build Task** to link to the task in CodeArts Build.
- **Build Packages**: displays the archiving records of software packages uploaded through build tasks. You can click  to download a package.

1.5 Version View

CodeArts Artifact displays software packages by version. In **Version View** tab page, artifact packages can be displayed by artifact package name and version number, and files can be sorted by update time.

Step 1 Access the release repo by following instructions in **Accessing Through a Specific Project**.

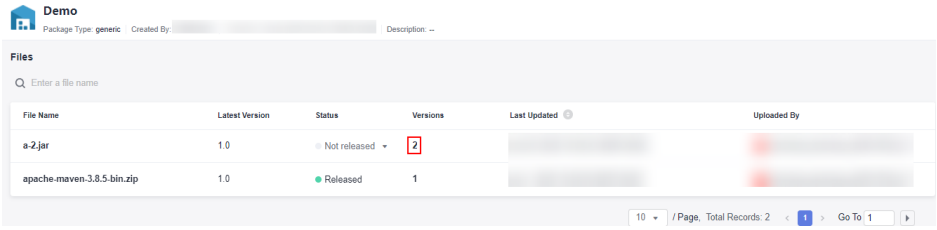
Step 2 You need to edit the version of the uploaded software package. By default, the release version of the software package archived by CodeArts Build is the version number set when build task is executed.



Step 3 Click **Version View** in the upper left corner of the page. The list of software packages with version numbers is displayed.

Step 4 Release repos store the software packages of different versions with the same name in the same file. Click a file name. The overview information about the latest version of the software package is displayed.

Step 5 Click versions. The version list of the software package is displayed.



Click a version number. The **Overview** and **Files** of the software package are displayed. In the **Files** page, click a file name. The storage location of the software package is displayed.

Step 6 After you edit the version of a software package, the status is **Not Released** by default. You can change the status.

- In the files, set the status of the file to **Released**. The software package of the latest version in the file is set to **Released**.
- Click **Versions** to go to version list. You can set the status of different versions to **Released**.

  NOTE

  The status changes from **Not Released** to **Released**. The status change is irreversible. Exercise caution when performing this operation. Files in **Released** status cannot be modified or edited (file names or version numbers), but can only be downloaded or deleted.

----End

1.6 Managing the Recycle Bin

Software packages or folders deleted from the release repo are moved to the recycle bin, where you can manage them.

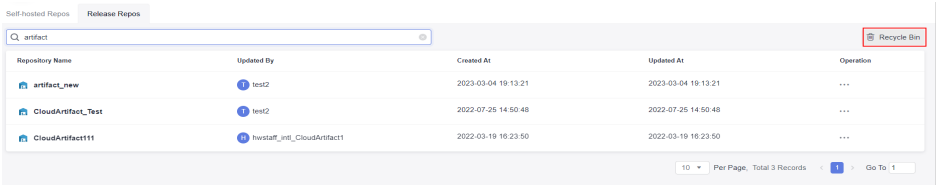
CodeArts Artifact provides an **overall recycle bin** and **project-level recycle bins**.

Overall Recycle Bin

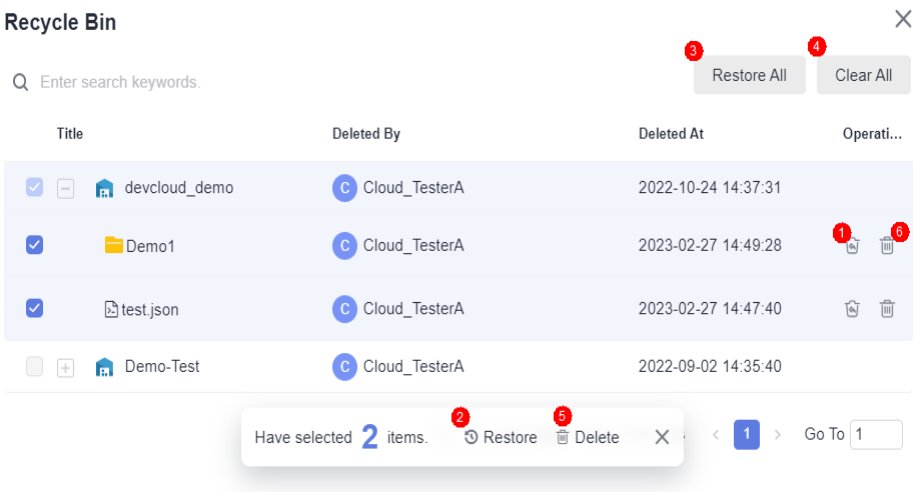
In the overall recycle bin, you can manage software packages and folders deleted from all projects.

Step 1 Log in to CodeArts and choose **Services** > **Artifact**.

Step 2 Click the **Release Repos** tab and click **Recycle Bin**.



Step 3 The deleted files of different projects are displayed on the page. Perform the following operations on the software package or folder as required:

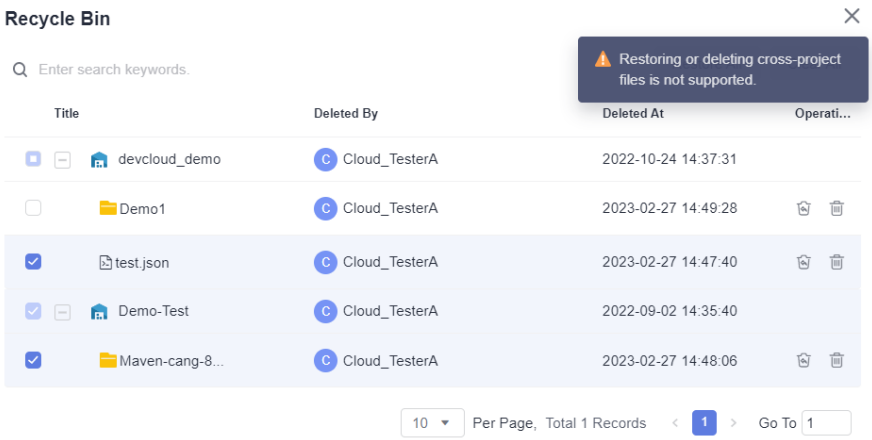


No.	Operation	Description
1	Restore	Click in the **Operation** column to restore a software package or folder.
2	Batch restore	Select multiple software packages or folders and click **Restore** below the list to restore all the selected software packages or folders.
3	Restore all	Click **Restore All** to restore all software packages or folders in the recycle bin by one click.
4	Clear recycle bin	Click **Clear All** to delete all software packages or folders from the recycle bin.

No.	Operation	Description
5	Batch delete	Select multiple software packages or folders and click **Delete** below the list to delete all the selected software packages or folders.
6	Delete	Click in the **Operation** column to delete a software package or folder.

NOTICE

1. If you choose to delete a software package or folder in the recycle bin, it cannot be retrieved anymore. Exercise caution when performing this operation.
2. When selecting multiple files to restore or delete in batches, the overall recycle bin does not allow you to restore or delete files across projects.



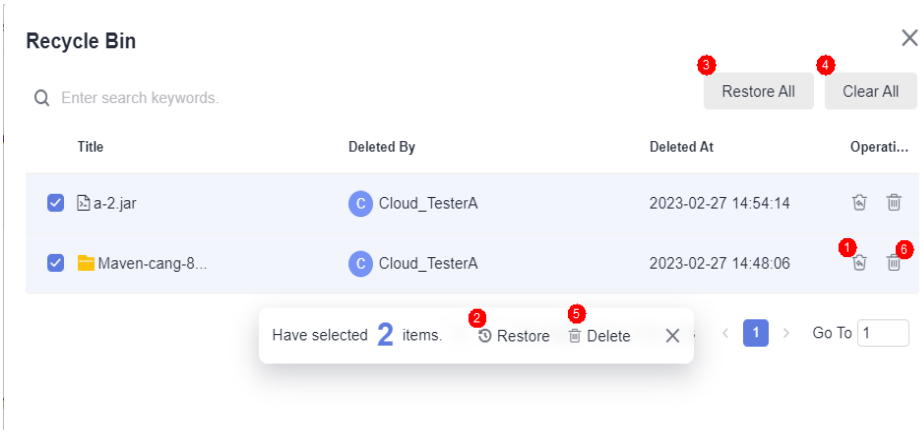
----End

Recycle Bin in a Project

You can process deleted software packages or folders in a project.

Step 1 Follow instructions in **Accessing Through a Specific Project** to access the **Release Repos** page and click **Recycle Bin** in the lower left corner of the page.

Step 2 The deleted files of this project are displayed on the page. Perform the following operations on the software package or folder as required:



No.	Operation	Description
1	Restore	Click in the **Operation** column to restore a software package or folder.
2	Batch restore	Select multiple software packages or folders and click **Restore** below the list to restore all the selected software packages or folders.
3	Restore all	Click **Restore All** to restore all software packages or folders in the recycle bin by one click.
4	Clear recycle bin	Click **Clear All** to delete all software packages or folders from the recycle bin.
5	Batch delete	Select multiple software packages or folders and click **Delete** below the list to delete all the selected software packages or folders.
6	Delete	Click in the **Operation** column to delete a software package or folder.

NOTICE

If you choose to delete a software package or folder in the recycle bin, it cannot be retrieved anymore. Exercise caution when performing this operation.

----End

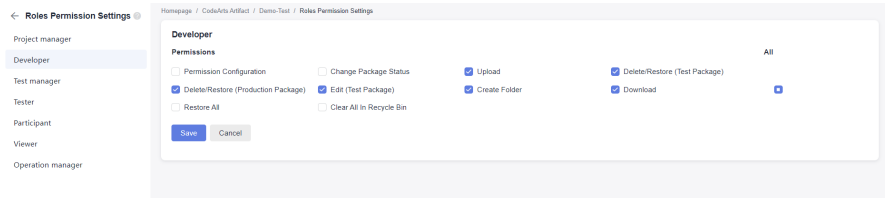
1.7 Setting Permissions

In the release repo, project roles have different operation permissions. Members who have the **Permission Settings** permission can edit the permission scope.

Step 1 Access the release repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Click ⋮ in the upper left corner of the page and choose **Set Project Permissions** from the drop-down list.

Step 3 Click the role for which you want to set permissions, select the permissions as required, and click **Save**.



The table below lists the default permission matrix provided by a release repo.

Operation/ Role	Project Creator	Project Manager	Developer	Test Manager	Tester	O&M Manager	Participant	Viewer
Set permissions	√	√	-	-	-	-	-	-
Change the package status	√	√	-	-	-	√	-	-
Upload	√	√	√	√	√	√	-	-
Delete/ Restore (test package)	√	√	√	√	-	√	-	-
Delete/ Restore (production package)	√	-	-	-	-	√	-	-
Edit (test package)	√	√	√	√	-	√	-	-
Create a folder	√	√	√	√	√	√	-	-
Download	√	√	√	√	√	√	√	√
Restore all	√	√	-	√	-	-	-	-
Clear recycle bin	√	√	-	√	-	-	-	-

NOTE

- By default, the project creator has all operation permissions and is not displayed on the page. The creator's permission scope cannot be changed.
- Viewers have only the download permission. A viewer's permission scope cannot be changed.

----End

1.8 Clearing Policies

A release repo automatically clear files on a scheduled basis. You can set a clearing policy to move expired files from the repository to the recycle bin or delete them permanently from the recycle bin based on the specified retention periods.

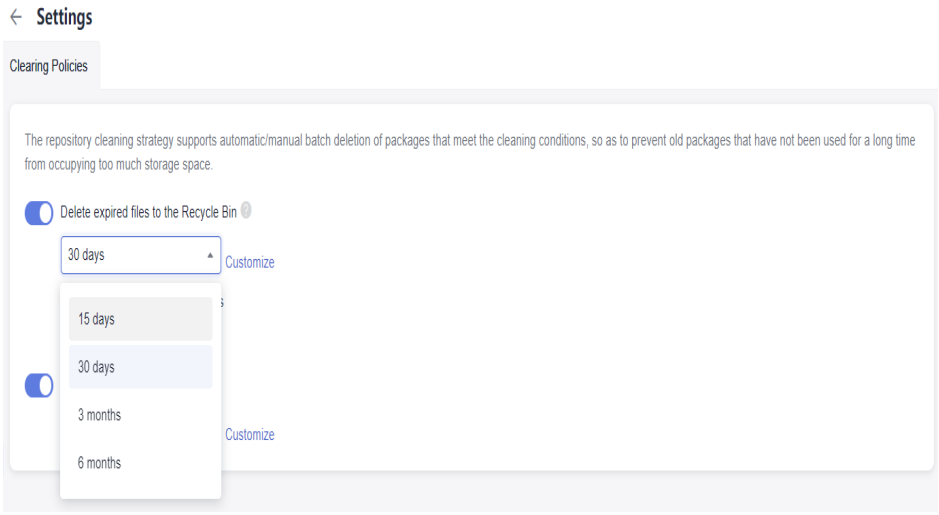
Step 1 Access the release repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Click **Settings** at the upper right corner of the page. The **Clearing Policies** page is displayed.

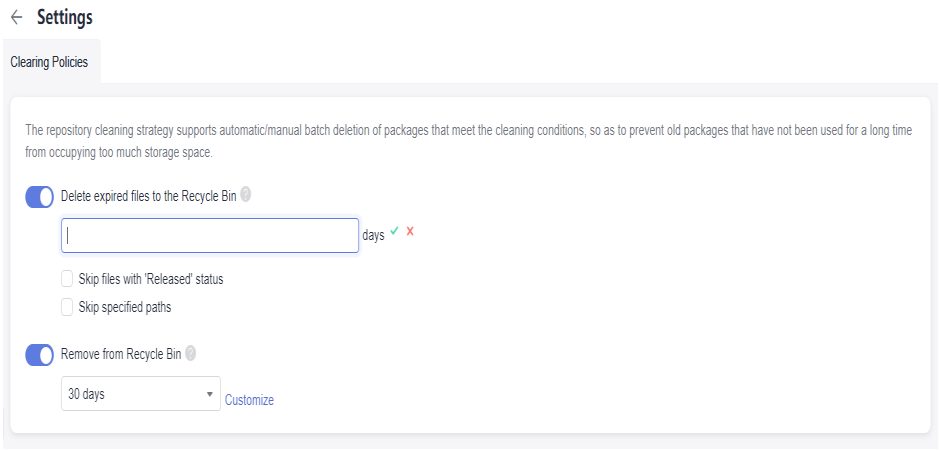
Step 3 Enable **Move expired files to the Recycle Bin** or **Clear from Recycle Bin** as required, and select a retention period from the drop-down list.

Default retention periods:

- **Move expired files to the Recycle Bin**: 30 days
- **Clear from Recycle Bin**: 30 days



You can also customize a period. Click **Customize**, enter a number, and click **√** to save.



NOTE

Parameters below are optional.

- **Skip Released files**: The system retains files in the production package state when deleting files. For details, see **Setting Status**.
- **Skip specified paths**: When cleaning up files, the system retains the software package that matches the file path set by the user. You can set multiple file paths (starting with a slash (/) and separated by semicolons (;)).

----End

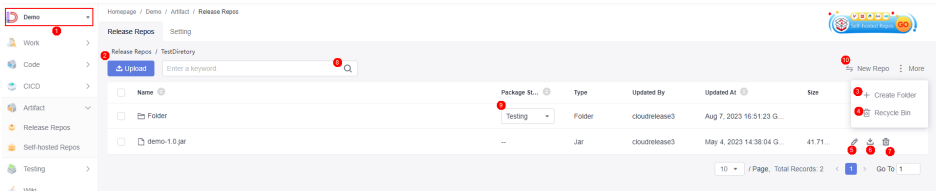
2 Release Repo (Legacy Version)

2.1 Accessing a Release Repo (Legacy Version)

Step 1 Access the release repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Click **Old Repo** in the lower left corner.

Step 3 View the existing file list of the current project. You can perform the following operations as required:



No.	Operation	Description
1	Switch project	Click the project name drop-down list to switch the project to view the legacy release repo.
2	Upload	Click **Upload** in the upper left corner of the list to manually upload local software packages to the release repo. The size of each file to upload cannot exceed 2 GB. **NOTE** You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to the release repo.
3	Create a folder	Choose **More** > **Create Folder** in the upper right corner of the list to create a folder on the current page.

No.	Operation	Description
4	Recycle bin	Choose **More** > **Recycle Bin** in the upper right corner of the list to access the recycle bin of the release repo, where you can manage deleted software packages or folders.
5	Edit	Click in the **Operation** column to edit an individual software package or folder. • Software package: You can change the name and release version of the software package. (The release version archived by CodeArts Build is the build sequence number by default.) • Folder: You can edit the folder name.
6	Download	Click in the **Operation** column to download the software package.
7	Delete	• Click in the **Operation** column. In the displayed dialog box, click **Remove to Recycle Bin** to move the corresponding software package or folder to the recycle bin, or click **Clear** to delete it permanently. • Select multiple software packages or folders. In the displayed dialog box, click **Remove to Recycle Bin** to move them to the recycle bin, or click **Clear** to delete them permanently. **NOTE** Software packages in the recycle bin still occupy the space used by the release repo. Once a software package is permanently deleted from the recycle bin, the occupied space will be released.
8	Search	Enter a keyword (a folder or file name) in the search box above the list and click to search for the software package whose name contains the keyword.
9	Change status	After entering a level-1 folder, you can change the status of a level-2 folder by clicking the drop-down list in the **Package Status** column. The status can be **Not Released** or **Released**. A folder can change from **Not Released** to **Released**, but such a change is irreversible. If a folder is **Released**, the folder cannot be changed or edited (changing the folder name, changing the file name in the folder, uploading the folder, changing the version number, or creating a subfolder). You can only download or delete it.
10	New repo	Click **New Repo** to return to the new version of release repo.

----End

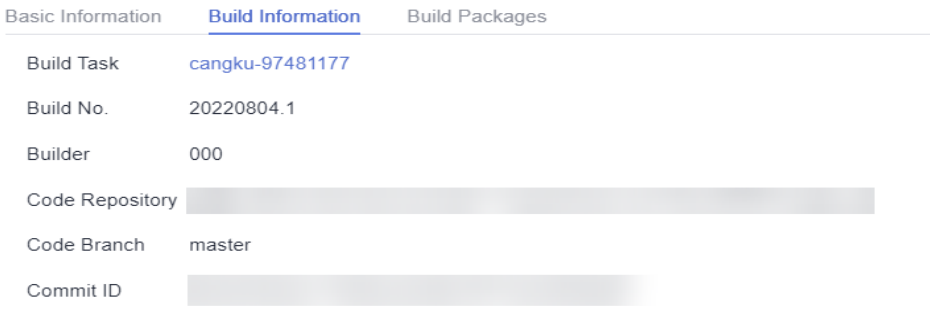
2.2 Viewing or Editing Software Package Details

On the release repo page, you can view or edit the software package details, including basic information, build information, and build packages.

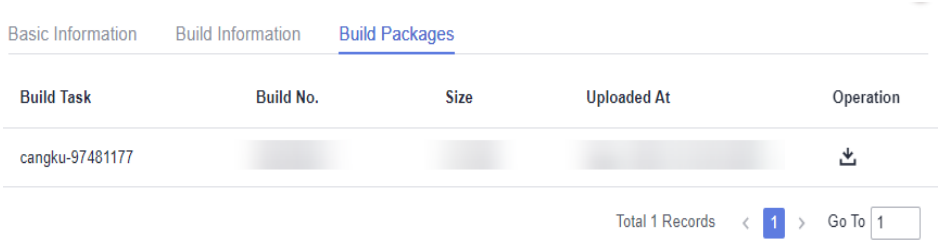
Access the release repo homepage and click the name of a software package. A drawer is displayed, showing the software package details. The release repo details are displayed on the **Basic Information**, **Build Information**, and **Build Packages** tab pages.

- The **Basic Information** tab page displays the software package name, version, size, path, download address, creator, creation time, checksum, and other information.

  You can click  in the upper right corner to change the name and release version of the software package. (The release version archived by CodeArts Build is the build sequence number by default.)
- The **Build Information** tab page displays the build task, build No., builder, code repository, code branch, and commit ID of the generated software package. Click **Build Task** to link to the task in CodeArts Build.



- The **Build Packages** tab page displays the archiving records of software packages uploaded through build tasks. You can click  to download a package.



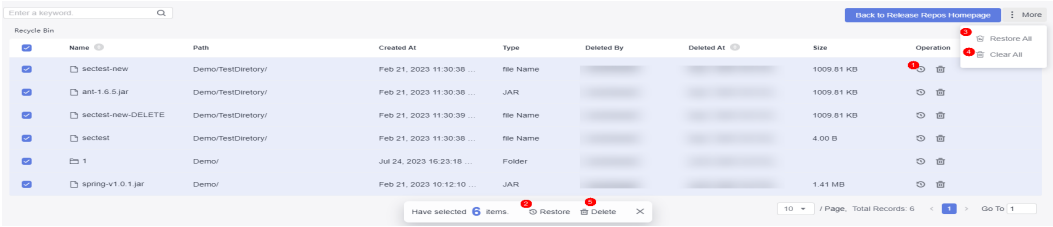
2.3 Managing the Recycle Bin

Software packages or folders deleted from the release repo are moved to the recycle bin, where you can manage them.

Step 1 Access a release repo of the legacy version.

Step 2 Choose **More** > **Recycle Bin**.

Step 3 Delete or restore software packages or folders as required.



No.	Operation	Description
1	Restore	Click in the **Operation** column to restore a software package or folder.
2	Batch restore	Select multiple software packages or folders and click **Restore** below the list to restore all the selected software packages or folders.
3	Restore all	Choose **More** > **Restore All** to restore all software packages or folders in the recycle bin.
4	Clear recycle bin	Choose **More** > **Clear All** to delete all software packages or folders from the recycle bin.
5	Batch delete	Select multiple software packages or folders and click **Delete** below the list to delete all the selected software packages or folders.
6	Delete	Click in the **Operation** column to delete a software package or folder.

NOTICE

If you choose to delete a software package or folder in the recycle bin, it cannot be retrieved anymore. Exercise caution when performing this operation.

----End

3 Self-hosted Repo (New Version)

3.1 Introduction

A self-hosted repo manages private components, supporting Maven, npm, Go, PyPI, RPM, Debian, Conan, and NuGet.

A self-hosted repo provides the following functions:

- Repository management: includes creating a repository, editing basic repository information, managing repository permissions, and connecting the repository to the local development environment.
- Private component management: includes uploading, downloading, searching for, and deleting private components, and managing the recycle bin.

3.2 Accessing a Self-hosted Repo

You can access the self-hosted repo page in either of the following ways: homepage entry and project entry.

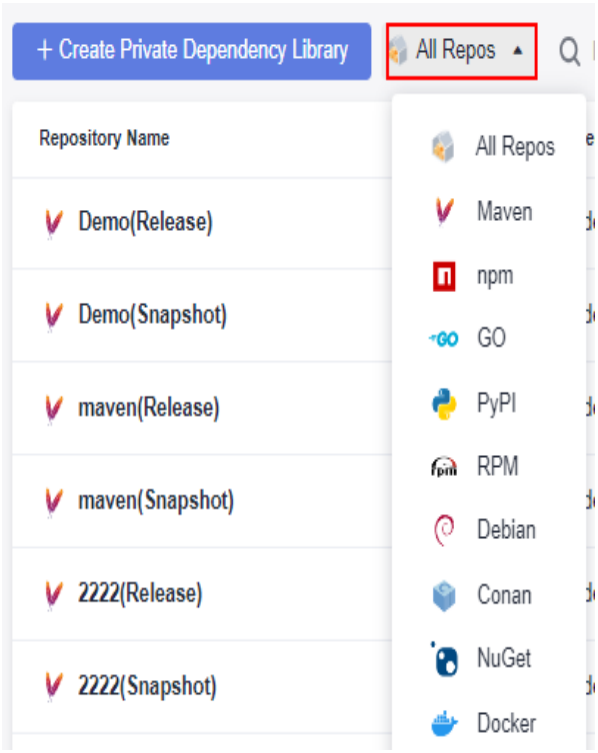
Accessing Through the Homepage

Step 1 Log in to CodeArts.

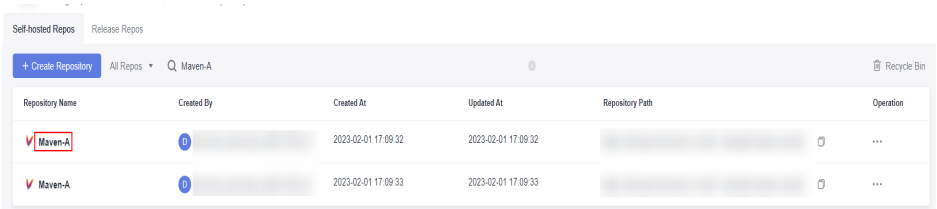
Step 2 Choose **Services** > **Artifact**.

Step 3 Click the **Self-hosted Repos** tab. All self-hosted repos created for the service are displayed.

Click the filter drop-down list box above to view repos by types.



Click the name of a repository to go to the self-hosted repo page of the project where the repository is located.



----End

Accessing Through a Specific Project

Step 1 Log in to the CodeArts homepage and click a card to access a project.

Step 2 Choose **Artifact** > **Self-hosted Repos**.

Step 3 View the list of different types of repositories archived in the current project.

Perform the operations described in the following sections as required.

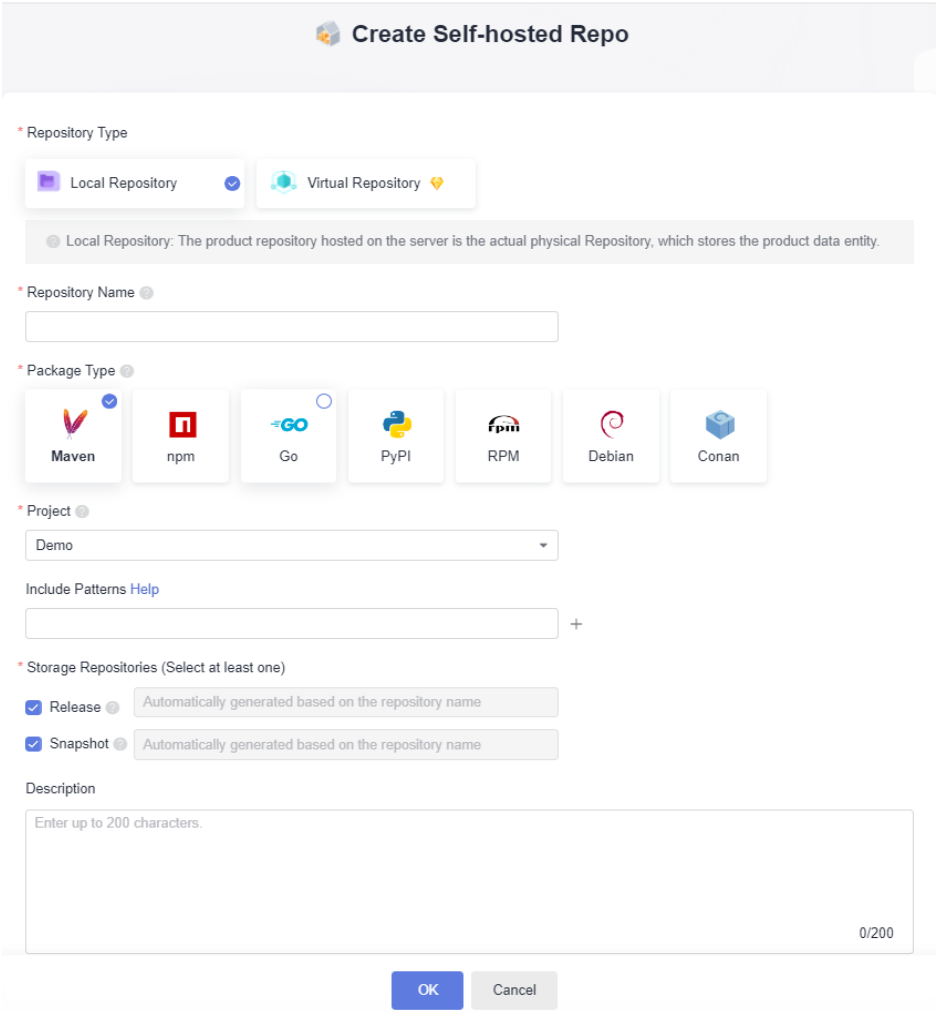
----End

3.3 Creating a Self-hosted Repo

If you use this service for the first time, you need to create a repo. Only tenant administrators have the permission to create self-hosted repos.

- Follow instructions in **Accessing Through the Homepage** to access the self-hosted repo. You can click **Create Self-hosted Repo** in the upper left corner of the page.
- Follow instructions in **Accessing Through a Specific Project** to access the self-hosted repo. You can click + 新建 in the upper left corner of the page.

Step 1 The **Create Self-hosted Repo** page is displayed.

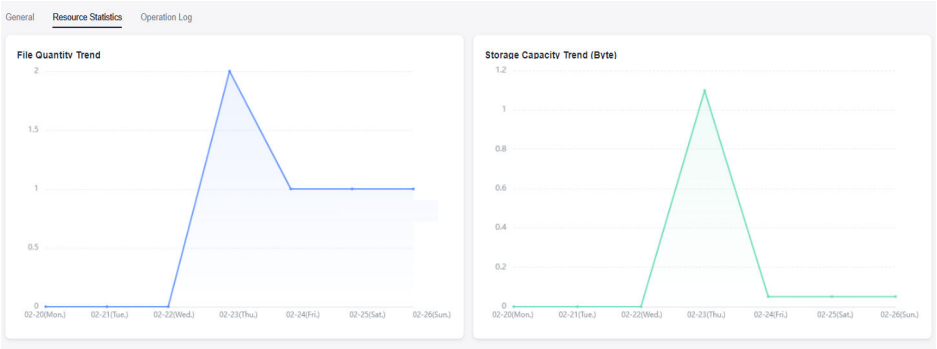


Step 2 Configure basic information and click **OK**.

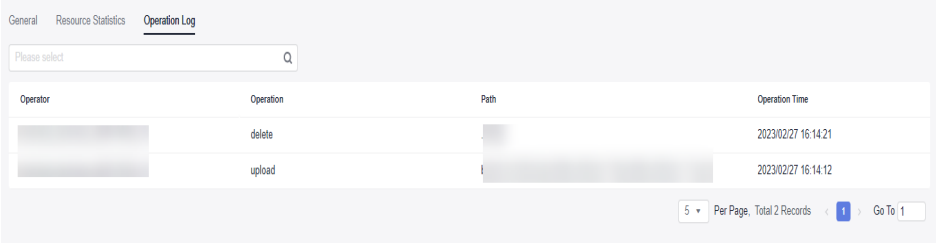
Configuration Item	Required	Description
Name	Yes	Enter up to 20 characters: letters, numbers, underscores (_), hyphens (-), and periods (.). **NOTE** After a self-hosted repo is created, the repository name cannot be changed.
Package Type	Yes	: supports Maven, npm, Go, PyPI, RPM, Debian, Conan, and NuGet artifact repositories. Complete the configuration for the selected format by following instructions in **Configuring Repository Items**.
Project	Yes	Select a project for the created repository. After the setting is complete, the project to which the user belongs cannot be changed.
Description	No	Enter up to 200 characters.

Step 3 View the name of the created self-hosted repo displayed in the list on the left of the page. Click the repository name to view the repository details. The repository details are displayed on the **General**, **Resources**, and **Operation Logs** tab pages.

- **General**: displays the repository name, repository type, repository path, relative path, creator, creation time, modifier, modification time, artifact count, and artifact size.
- **Resources**: collects statistics on artifacts uploaded to the repository by **File Counts** and **Storage Capacity (Byte)**.



- **Operation Logs**: displays the operation history of uploading, deleting, and restoring data from the recycle bin in the repository.



----End

Configuring Repository Items

The following table describes the configuration items specific to each type of repository.

Type	Configuration Item	Required	Description
Maven	Storage repositories	Yes	The options are **Release** and **Snapshot**. You are advised to select both. If so, two repositories will be generated: **Release** and **Snapshot**. If you select one, a **Release** or **Snapshot** repository will be generated.
	Include patterns	No	Enter the required path, and click **+**. During package builds, only the Maven files whose path starts with this path can be uploaded to the self-hosted repo.
npm	Include patterns	No	Enter the required path, and click **+**. During package builds, only the npm files whose path starts with this path can be uploaded to the self-hosted repo.
Go	Include patterns	No	Enter the required path, and click **+**. During package builds, only the Go files whose path starts with this path can be uploaded to the self-hosted repo.
PyPI	Include patterns	No	Enter the required path, and click **+**. During package builds, only the PyPI dependency packages in which the **name** value in the **setup.py** file matches this path can be uploaded to the self-hosted repo.
RPM	Include patterns	No	Enter the required path, and click **+**. During package builds, only the RPM binary files whose path starts with this path can be uploaded to the self-hosted repo.

Type	Configuration Item	Required	Description
Conan	Include patterns	No	Enter the required path, and click **+**. Only the Conan files whose path starts with this path can be uploaded from a local client to the self-hosted repo.

3.4 Managing Self-hosted Repos

You can edit repository descriptions, add paths, delete repositories, and manage user permissions.

Editing Repository Descriptions and Paths

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be edited.

Step 2 Click **Settings** on the right of the page to display the basic information about a repository.

Step 3 Edit the repository description as required and click **Submit**.

NOTE

On the basic information page, the repository name, package type, home project, and permission scope cannot be modified.

On the **Basic Information** page of the repository, enter the path and click + to add paths for the Maven, npm, Go, PyPI, RPM and Conan repositories.

Click to delete a path.

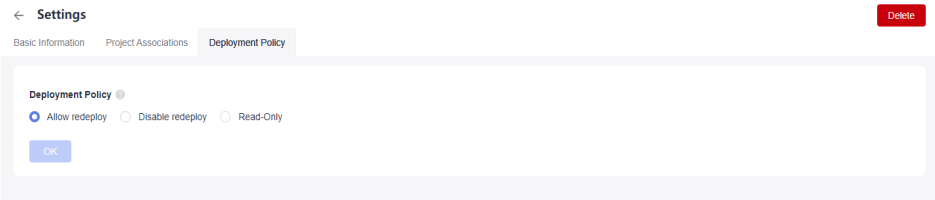
----End

Configuring Deployment Policies

The self-hosted repo supports three version policies: **Allow redeploy**, **Disable redeploy**, and **Read-only**. You can set whether to allow artifacts in the same path to be uploaded and overwrite the original package.

Step 1 Access the self-hosted repo homepage. In the left pane, click the repository name.

Step 2 Click **Settings** on the right of the page. The basic information about the repository is displayed. Click the **Deployment Policies** tab.



- **Allow redeploy** (selected by default): Artifacts in the same path can be uploaded. After being uploaded, the original package will be overwritten.
- **Disable redeploy**: Artifacts in the same path cannot be uploaded.
- **Read-only**: Artifacts cannot be uploaded, updated, or deleted. You can download an uploaded artifact.

Step 3 Click **OK**.

----End

Deleting a Repository

You can delete a self-hosted repo. Deleted repositories are moved to the recycle bin.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be deleted.

Step 2 Click **Settings** on the right of the page to display the basic information about a repository.

Step 3 Click **Delete**. Check that the deleted repository is no longer displayed in the repository list in the left pane.

----End

Managing Repository Permissions

After a repository is created, the mapping between project members and repository roles is as follows:

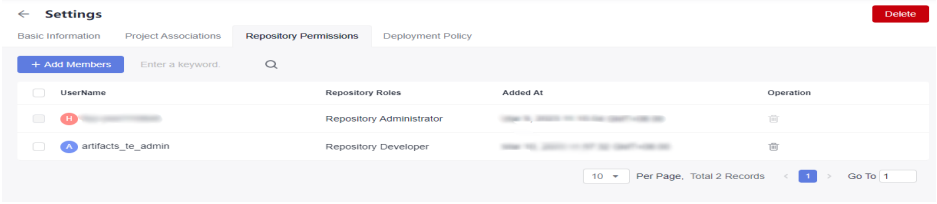
- The project creator and project manager are repository administrators.
- The developer, test manager, tester, and O&M manager are repository developers.
- The participant, viewer, and customized roles are repository viewers.

To add or delete permissions for self-hosted repo members, perform the following steps:

Step 1 Go to the self-hosted repo page and select the target repository from the list.

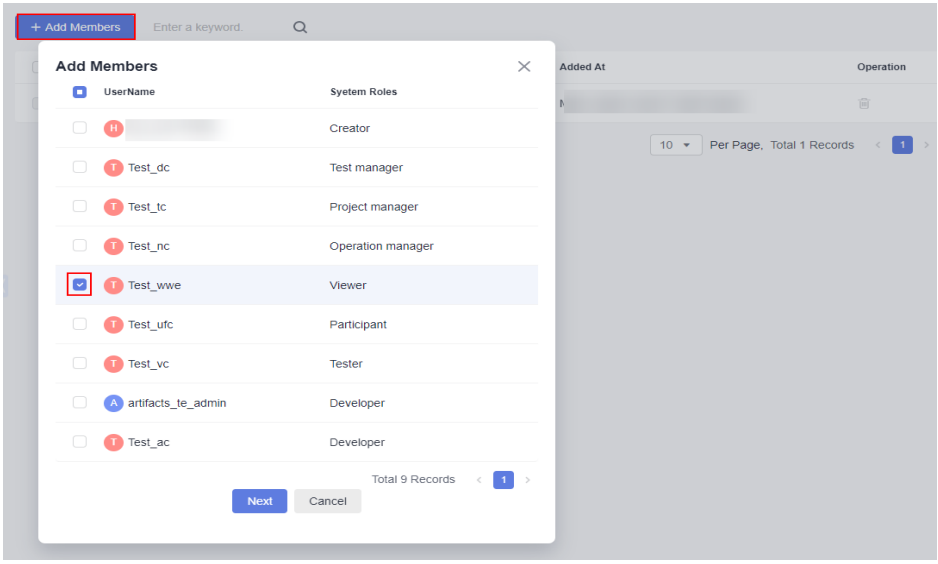
Step 2 Click **Settings** on the right of the page.

Step 3 Click the **Repository Permissions** tab. The added repository members are displayed in the list.



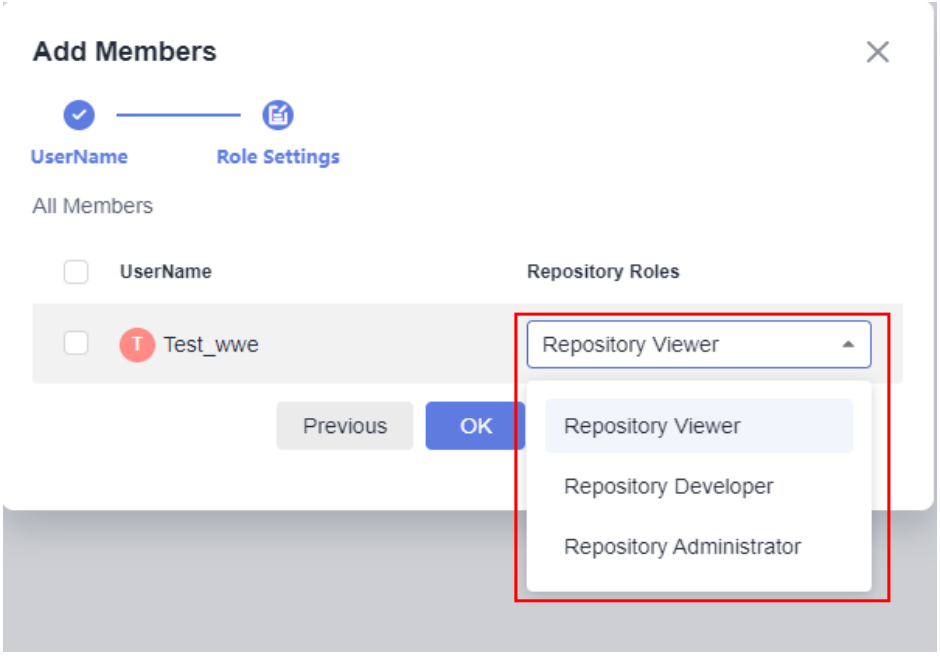
Step 4 Add members.

Click **Add Members** in the upper left corner, select a member, and click **Next**.



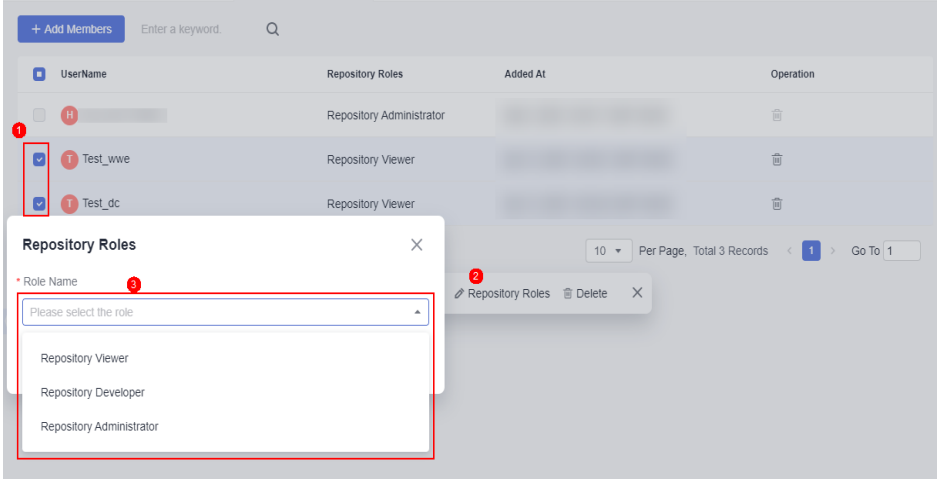
Step 5 Assign roles to the member.

Select **Repository Administrator**, **Repository Developer**, or **Repository Viewer** from the **Repository Roles** drop-down list.



Step 6 Click **OK**. The repository member is added and the repository role is configured. The newly added member is displayed in the list.

Step 7 In the member list, select multiple repository members and click **Repository Roles** to configure repository roles in batches.



----End

The following table lists the operations of each repository permission.

Operation/Role	Tenant Administrator			Non-Tenant Administrator		
	Repository Administrator	Developer	Viewer	Repository Administrator	Developer	Viewer
Create a self-hosted repo	√	√	√	×	×	×
Edit a self-hosted repo	√	√	√	×	×	×
Manage the association between repositories and projects	√	√	√	×	×	×
Upload a private component	√	√	×	√	√	×

Download a component	√	√	√	√	√	√
Delete a component	√	√	×	√	√	×
Restore a component	√	√	×	√	√	×
Permanently delete a component	√	√	×	√	√	×
Delete a repository	√	×	×	×	×	×
Restore a repository	√	√	×	√	√	×
Permanently delete a repository	√	×	×	×	×	×
Clear recycle bin	√	√	√	×	×	×
Restore all	√	√	√	×	×	×
Manage user permissions	√	√	√	√	×	×

NOTE

1. Tenant administrators (IAM users with the **Tenant Administrator** permissions) can manage all projects of a tenant.
2. Project creators who are not tenant administrators have the permissions listed in the preceding table.
3. IAM users created without being added to any groups do not have permissions. Administrators can assign permissions to these IAM users on the IAM console. Then users can use cloud resources in the account based on the assigned permissions. For details, see **Creating a User Group and Assigning Permissions**.

Set-Up

You can connect the self-hosted repo to a local development environment so that private components in the self-hosted repo can be used during local development.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be connected to the local development environment.

Step 2 Click**Tutorial** on the right of the page.

Step 3 In the displayed dialog box, click **Download Configuration File** to download the configuration file to your local directory.

Step 4 Copy the downloaded file to the corresponding directory based on the instructions in the **Information** dialog box.

----End

Configuring the Encryption Mode of a Self-hosted Maven

Selecting Maven as the dependency management tool

- Ensure that you have installed **JDK** and Maven.
- Download the provided configuration file, or modify the Maven **settings.xml** file in the **conf** or **.m2** directory according to the following procedure.

To encrypt a password, perform the following operations:

Step 1 Create a master password.

```
mvn --encrypt-master-password <password>
```

The following encrypted password is generated: {jSMOWnoPFgsHVpMvz5VrIt5kRbzGpI8u+9EF1iFQyJQ=}.

Save it to the **${user.home}/.m2/settings-security.xml** file. For example:

```
<settingsSecurity>
<master>{jSMOWnoPFgsHVpMvz5VrIt5kRbzGpI8u+9EF1iFQyJQ=}</master>
</settingsSecurity>
```

Step 2 Encrypt the password.

```
mvn --encrypt-password <password>
```

The following encrypted password is generated: {COQLCE6DU6GtcS5P=}.

Save it to the **settings.xml** file. For example:


```
<settingsSecurity>
<master>{jSMOWnoPFgsHVpMvz5VrIt5kRbzGpI8u+9EF1iFQyJQ=}</master>
</settingsSecurity>
```

----End

Selecting Gradle as the dependency management tool

- Ensure that you have installed **JDK** and **Gradle**.

Step 1 Add the **nu.studer.credentials** plug-in to the **build.gradle** file in the Gradle project.

```
plugins{
 id 'nu.studer.credentials' version '2.1'
}
```

Step 2 Create an encryption password.

```
gradle addCredentials -PcredentialsKey=accountPassword -PcredentialsValue="**"
```

The following encrypted password is generated: cVe******kg\=\=.

By default, the data is stored in the **GRADLE_USER_HOME/gradle.encrypted.properties** file. For example:

```
accountPassword=cVe******kg\=\=
```

Step 3 Configure the repository with an encrypted password in the **build.gradle** file.

```
def password = credentials.accountPassword
            repositories {
              maven {
                credentials {
                  username 'cn-
north-1_f9e40463c23845438ca9efd3a7ec854e_67902fb51ded48c2868310a07e1569e7'
                  password password
                }
                url 'https://devrepo.devcloud.huaweicloud.com/artgalaxy/cn-
north-1_f9e40463c23845438ca9efd3a7ec854e_maven_1_7/'
              }
            }
```

----End

Resetting the Repository Password

You can reset the password in the self-hosted repo configuration file. After the password is reset, download the configuration file again to replace the original file.

Step 1 Access the self-hosted repo homepage. Click ⋮ above the repository list on the left and choose **Reset Repository Password**.

Step 2 In the displayed dialog box, click **OK**. Check that a message is displayed indicating the password has been reset.

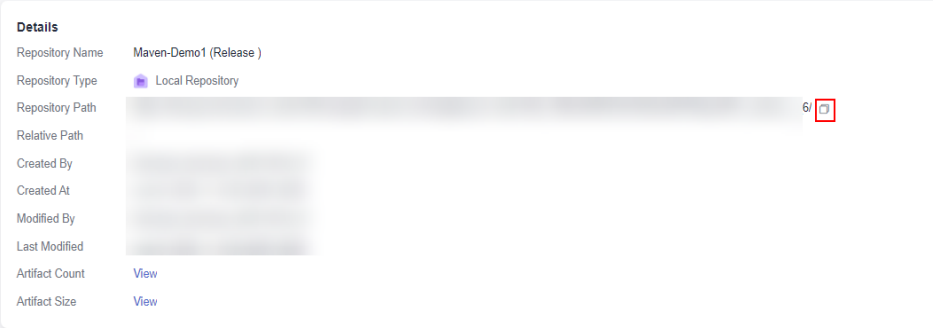
----End

Obtaining the Self-hosted Repo Path

The path of the self-hosted repo will be used when you connect the repository to the local development environment. You can perform the following operations to obtain the path:

Step 1 Access the self-hosted repo homepage. In the left pane, click the repository name.

Step 2 The path of the self-hosted repo is displayed in the repository details on the page. You can click to obtain the path.

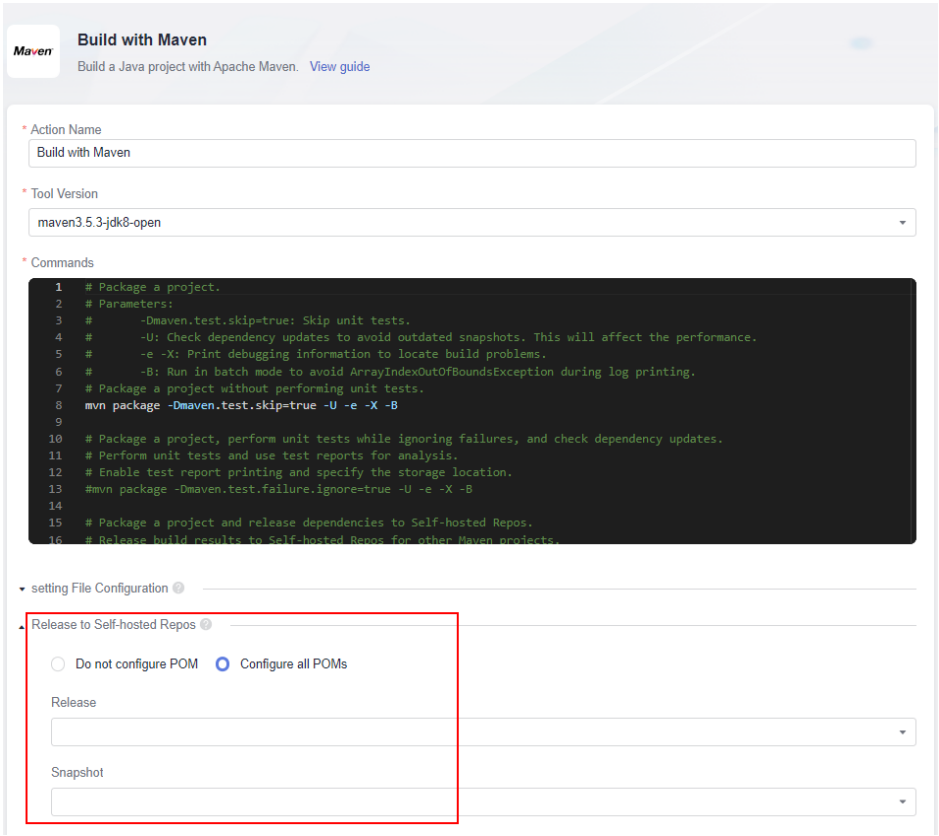


----End

Obtaining the Association Between the Self-hosted Maven Repo and Project

When uploading a Maven component to the self-hosted repo through a build task, specify the repository path in the **Build with Maven** step.

- **Do not configure POM**: The dependency package is not released to the self-hosted repo.
- **Configure all POMs**: If you run the **mvn deploy** command, the dependency package is released to the specified release repository and snapshot repository.



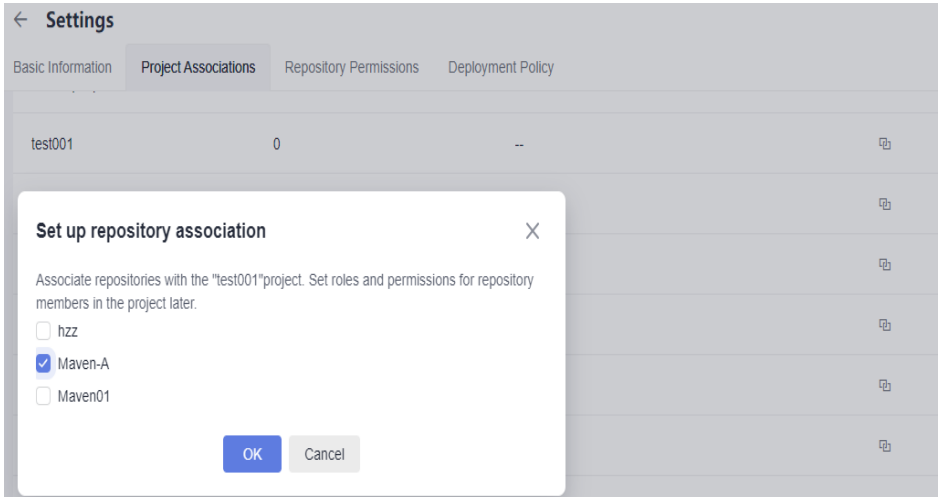
After the self-hosted Maven repo is associated with a project, you can select the repository in the build step of the build task in the project.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of a self-hosted Maven repo.

Step 2 Click **Settings** on the right of the page, and choose **Project Associations**.

Step 3 In the **Operation** column of the target project in the self-hosted Maven repo, click .

Step 4 In the displayed dialog box, select the repository name, and click **OK**.



After an "Operation successful" message is displayed, the value of **Associated Repositories** for the project will be updated according to the number of selected repositories.

----End

3.5 Version View

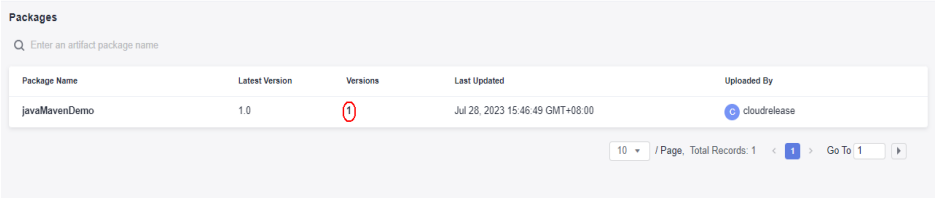
A self-hosted repo displays different types of private components by version. In **Version View** tab page, artifact packages can be filtered and displayed by artifact package name and version number, and files can be sorted by update time.

Step 1 Access the self-hosted repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Select **Version View** in the upper left corner of the page and click a repository name in the list on the left. The software package version list of the repository is displayed. For details about how to set versions for different types of private components, see **3.8 Uploading a Private Component**.

Step 3 Self-hosted repos store the software packages of different versions with the same name in the same file. Click a package name. The overview information about the latest version of the software package is displayed.

Step 4 Click versions. The version list of the software package is displayed.



Click a version number. The **Overview** and **Files** of the software package are displayed. In the **Files** page, click a file name. The storage location of the software package is displayed.

----End

3.6 Configuring the Artifact Cleanup Strategy

You can automatically and manually delete artifacts that meet deletion conditions in batches. When a user creates a self-hosted Maven repo, the storage repositories include **Release** and **Snapshot**.

The snapshot of a Maven artifact is a special version that specifies a copy of the current development progress, and is different from a common version. Maven checks a new snapshot in the remote repository during each build and provides the maximum number of snapshot versions that can be retained and the function of automatically clearing expired snapshot versions.

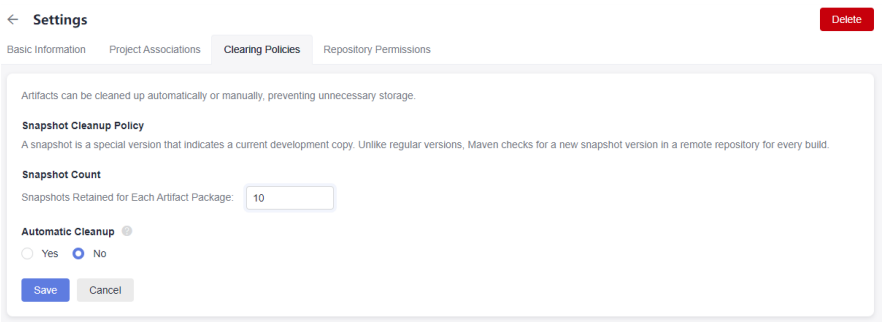
The artifact cleanup strategy reduces the waste of storage space, makes artifacts in the repository clear, and ensures that artifacts are transferred in order during development, testing, deployment, and release.

Procedure

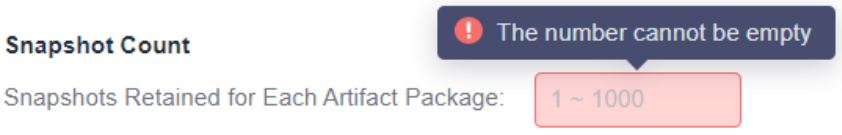
Step 1 Access the self-hosted repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Select the corresponding **Snapshot** self-hosted Maven repo from the repository list on the left and click **Settings** in the upper right corner of the page.

Step 3 Click the **Clearing Policies** tab.



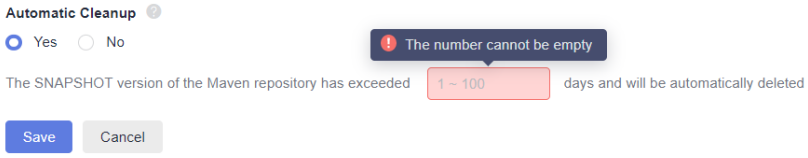
Step 4 Set the maximum number of **Snapshot Count**. The value ranges from 1 to 1000.



When the version of an artifact package exceeds this value, the package of the earliest version is overwritten by the package of the latest version.

Step 5 Enable automatic cleanup (**No** by default). Click **Yes** and enter the number of days. Snapshot versions that have been stored for more than the specified number of days will be automatically cleaned up.

The automatic cleanup time cannot be less than 1 day or greater than 100 days.



Step 6 Click **Save** to complete configuration.

----End

3.7 Virtual Repository

3.7.1 Introduction

The custom agent repository function is new in CodeArts Artifact. You can customize agent repositories to proxy open-source community repositories and third-party dependent repositories. After files are downloaded from the agent repository, the corresponding files can be cached to CodeArts Artifact, solving the pain point of slow download of third-party dependency. The third-party dependency has the same download experience as the local repository.

NOTE

The proxy settings are available for Maven, npm, and PyPI in self-hosted repos.

3.7.2 Custom Source

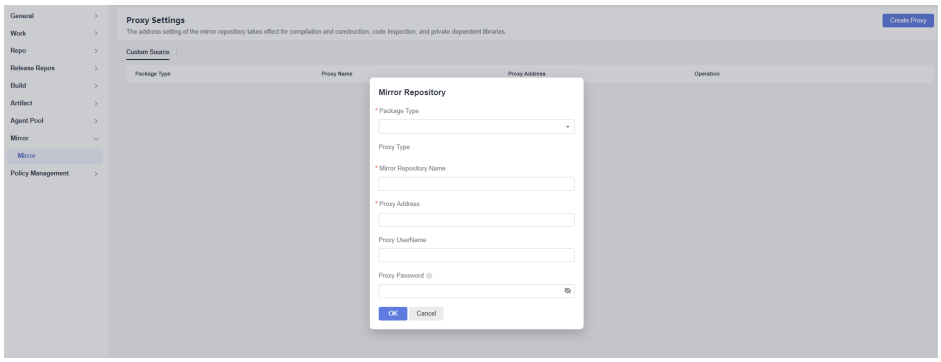
In the self-hosted repo, you can add custom mirror sources to the Maven, npm, and PyPI virtual repositories. To configure a custom mirror source, perform the following procedure:

Step 1 Log in to the CodeArts homepage, click the username in the upper right corner of the page, and choose **All Account Settings** from the drop-down list.

Step 2 In the navigation pane on the left, choose **Mirror** > **Mirror**.

Step 3 Click the **Custom Source** tab and click **Create Proxy** at the upper right corner of the page.

Step 4 In the displayed dialog box, choose the package type, and enter the mirror repository name (required), proxy address (required), proxy username, and proxy password.



NOTE

1. The proxy address of the mirror repository must start with **https://** or **http://**. Otherwise, an error message is displayed, indicating that the URL is invalid.
2. If you do not set the proxy password, the password set last time is used by default.

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

Step 6 You can perform the following operations on a custom proxy source that has been added.

Operation	Description
Edit	Click in the **Operation** column to change the mirror repository name, proxy username, and proxy password.
Delete	Click in the **Operation** column to delete the custom proxy source. If the custom proxy source to be deleted has been associated with a self-hosted repo, remove the proxy source on the **Proxy Settings** page of the corresponding repository and return to this page to delete the proxy source.

----End

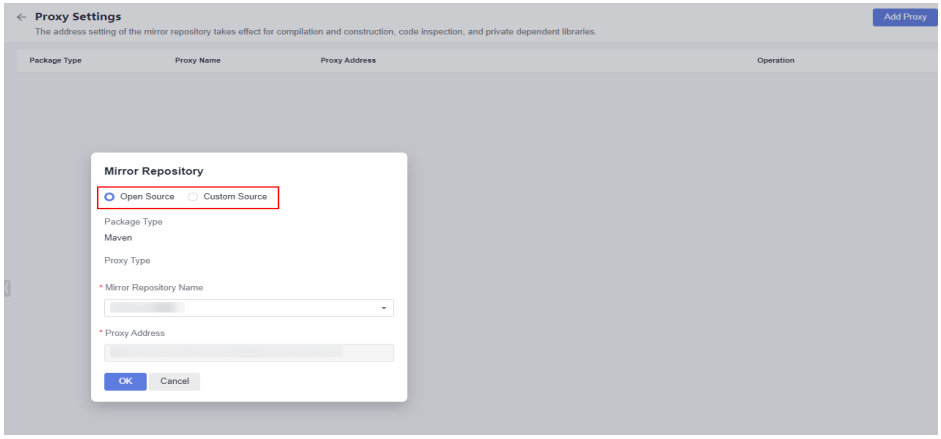
3.7.3 Proxy Settings

Step 1 For details about how to create a virtual repository, see **3.3 Creating a Self-hosted Repo**.

Step 2 Access the self-hosted repo homepage. In the left pane, select the target virtual repository.

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

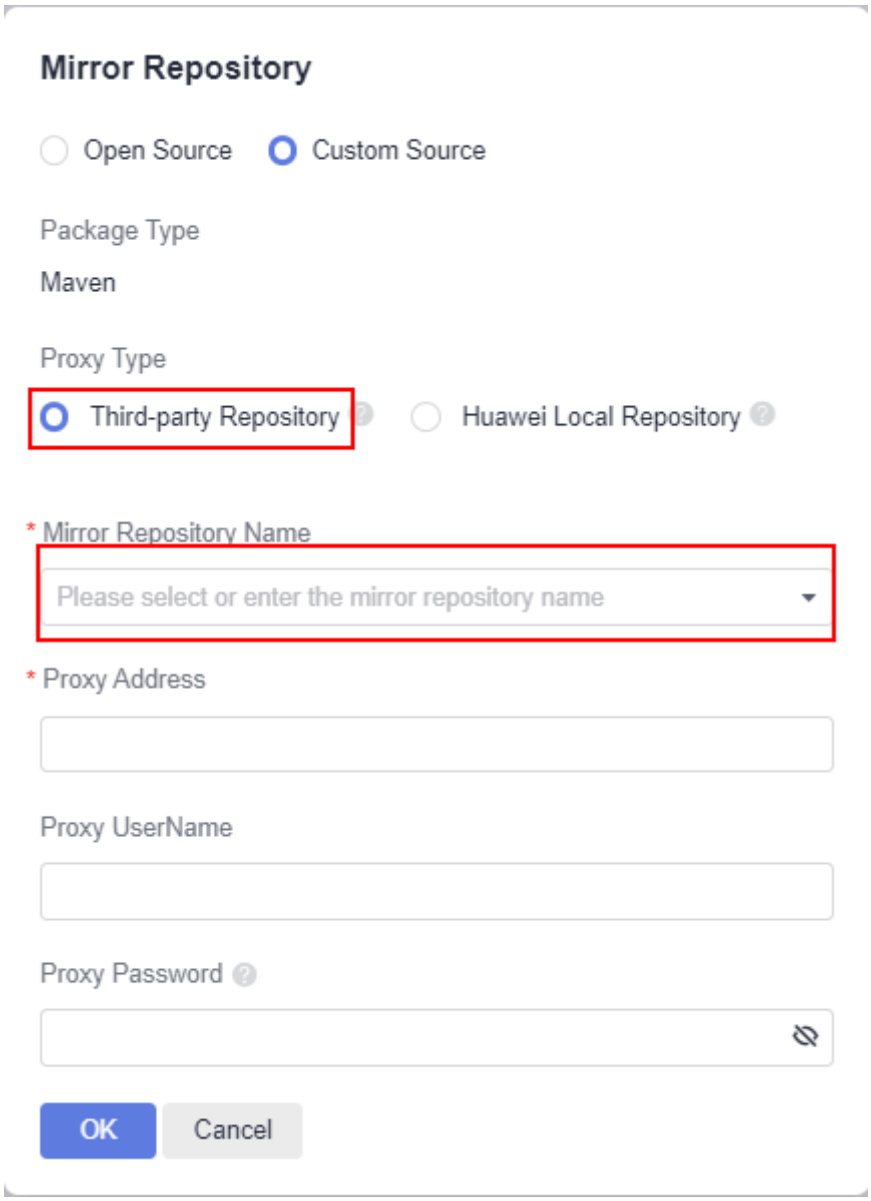
Step 4 Click **Add Proxy** and select **Open Source** or **Custom Source**.



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

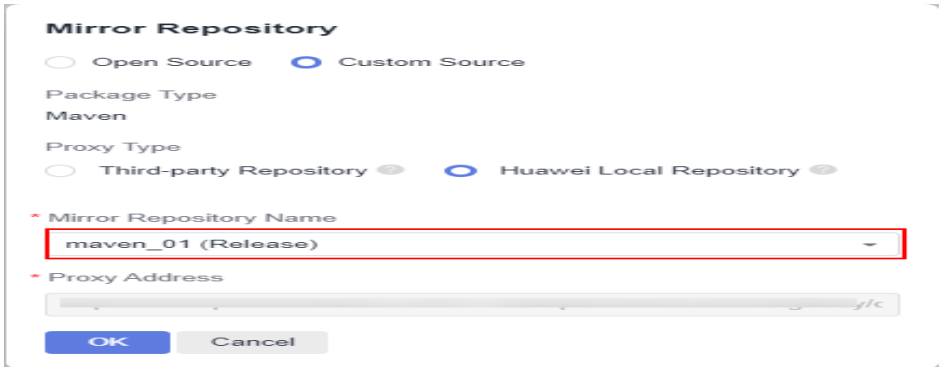
- **Third-party repository**: Set a third-party repository or a repository created by a user as the proxy source.

  After selecting a third-party repository, click the **Mirror Repository Name** drop-down list and select a custom proxy source. For details about how to create a custom proxy source, see **3.7.2 Custom Source**.



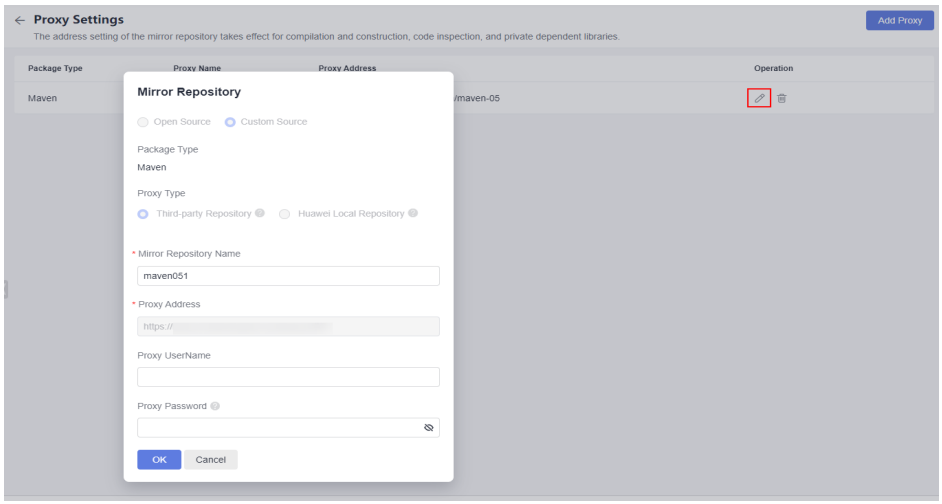
- **Huawei local repository**: Set Huawei local repository as the proxy source. Users can only select the local repository of which they are the repository administrator.

  You can select a local repository of a self-hosted repo from the **Mirror Repository Name** drop-down list.



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

- Click in the **Operation** column to change the mirror repository name, proxy username, and proxy password.



NOTE

Users cannot edit the proxy source of the Huawei local repository.

- Click in the **Operation** column to delete a proxy.

----End

3.8 Uploading a Private Component

Only repository administrators and developers can upload private components. You can set repository roles on the **Repository Permissions** page.

Procedure

Step 1 Access the self-hosted repo homepage. In the left pane, choose the repository to which the private component is to be uploaded.

Step 2 Click **Upload**.

Step 3 Set the component parameters, select the file, and click **Upload**. The detailed configuration for each type of component is described below.

NOTE

1. The maximum size of a file uploaded to the self-hosted repo is 100 MB for the Maven/npm/PyPI/RPM/Debian and 20 MB for the NuGet.
2. You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to the self-hosted repo.

----End

Introduction to Maven Components

- POM: Project Object Model (POM) is the basic working unit of a Maven project. It is an XML file that contains basic project information to describe

how to build a project and declare project dependencies. When a build task is executed, Maven searches for POM in the current directory, reads the POM, obtains the required configuration information, and constructs the target component.

- Maven coordinates: X, Y, and Z are used to uniquely identify a point in the three-dimensional space. In Maven, GAV is used to identify a unique Maven component package. GAV is short for **groupId**, **artifactId**, and **version**. **groupId** indicates a company or organization. For example, Maven core components are in the **org.apache.maven** organization. **artifactId** indicates the name of a component package. **version** indicates the version of the component package.
- Maven dependency: The dependency list is the cornerstone of POM. The building and running of most projects depend on the dependency on other components. Add the dependency list to the POM file. If the App component depends on the App-Core and App-Data components, the configuration is as follows:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.companyname.groupname</groupId>
    <artifactId>App</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Core</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Data</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
</project>
```

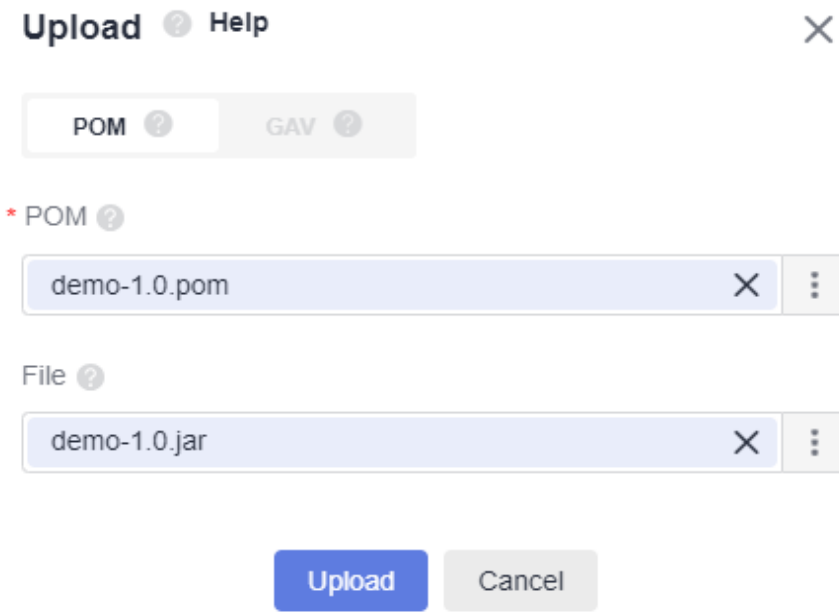
Uploading a Maven Component

A self-hosted repo supports two upload modes: POM and GAV.

Upload Mode	Description
POM	GAV parameters are obtained from the POM file. The system retains transitive dependencies of components.
GAV	GAV, short for **Group ID**, **Artifact ID**, and **Version**, is the unique identifier of a JAR package. In this mode, GAV parameters are manually specified. The system automatically generates a POM file without any transitive dependency.

- POM

In POM mode, you can upload only the POM file or upload the POM file and related components. The name of the uploaded file must be the same as the **artifactId** value and **version** value in the POM file. As shown in the following figure, the **artifactId** value is **demo** and the **version** value is **1.0** in the POM. The uploaded file must be **demo-1.0.jar**.



The POM file structure is as follows:

```
<project>
 <modelVersion>4.0.0</modelVersion>
 <groupId>demo</groupId>
 <artifactId>demo</artifactId>
 <version>1.0</version>
</project>
```

NOTE

The **modelVersion** tag must exist and the value must be **4.0.0**, indicating that **Maven2** is used.

If you upload files in both the **POM** and **File** area, the **artifactId** and **version** parameter values in the uploaded POM file must match the name of the file uploaded in the **File** area. For example, if **artifactId** is **demo** and **version** is **1.0** in the POM file, the name of the file to be uploaded in the **File** area must be **demo-1.0**. Otherwise, the upload will fail.

- GAV

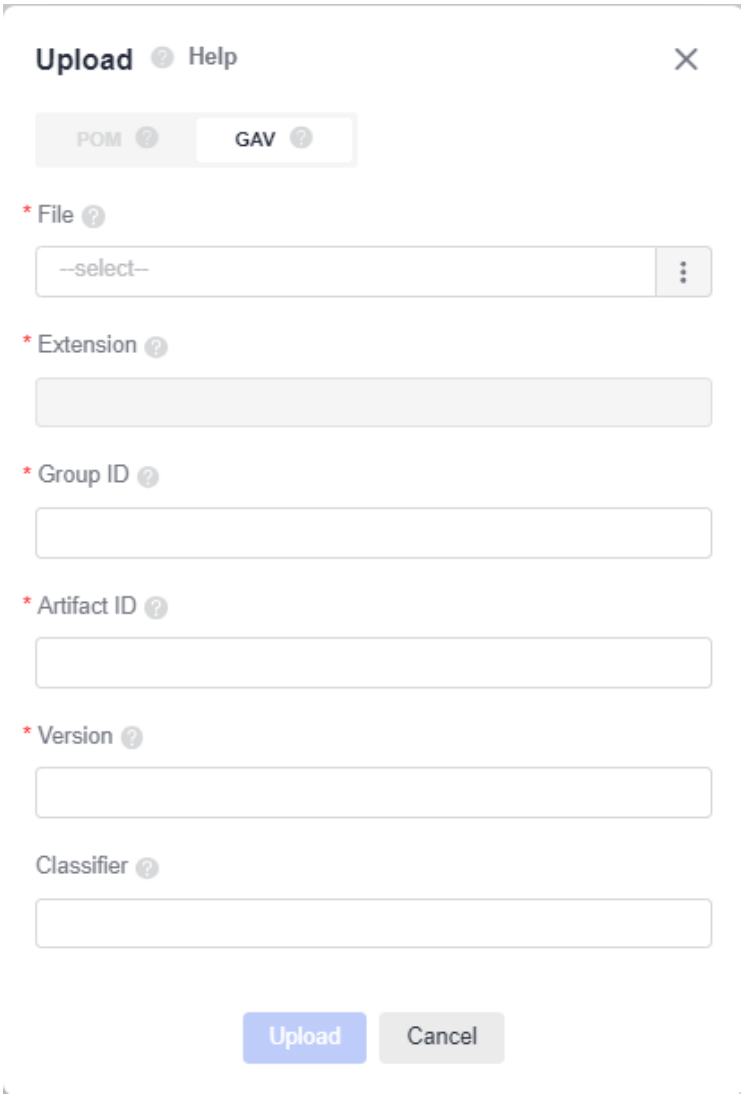
  In the GAV mode, the **Group ID**, **Artifact ID**, and **Version** parameters must be manually specified and they determine the name of the file to be uploaded. **Extension** indicates the packaging type, which determines the type of the file to be uploaded.

  Classifiers are used to distinguish artifacts that are constructed from the same POM and have different contents. This field is optional. It can contain letters, digits, underscores (_), hyphens (-), and dots (.). If you enter a value, it will be appended to the file name.

  Common Usage Scenario

  – Differentiate versions by names, such as **demo-1.0-jdk13.jar** and **demo-1.0-jdk15.jar**.

- Differentiate usage by names, such as **demo-1.0-javadoc.jar** and **demo-1.0-sources.jar**.



Introduction to npm Components

Node Package Manager (npm) is a JavaScript package management tool. The npm component package is the object managed by the npm, and the self-hosted npm repo is for managing and storing the npm component package.

The npm component package consists of the structure and file description.

- Package structure: organizes various files in a package, such as source code files and resource files.
- Description file: describes package information. Example: **package.json**, **bin**, and **lib** files

The **package.json** file in the package is a description file of a project or module package. It contains information such as the name, description, version, and author. The **npm install** command downloads all dependent modules based on this file.

An example of the **package.json** file is as follows:

```
{
  "name": "third_use",            //Package name
  "version": "0.0.1",           //Version number
"description": "this is a test project", // Description
  "main": "index.js",           //Entry file
  "scripts": {                  //Script commands
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [                 //Keyword
    "show"
  ],
  "author": "f",                //Developer name
  "license": "ISC",             //License agreement
  "dependencies": {             //Project production dependencies
    "jquery": "^3.6.0",
    "mysql": "^2.18.1"
  },
  "devDependencies": {          //Project development dependencies
    "less": "^4.1.2",
    "sass": "^1.45.0"
  }
}
```

The **name** and **version** are the most important fields and must exist. Otherwise, the current package cannot be installed. The two attributes together form the unique identifier of an **npm** package.

name indicates the name of a package. The first part of the name, such as **@scope**, is used as the namespace. The other part is **name**. Generally, you can search for the **name** field to install and use the required package.

```
{
  "name": "@scope/name"
}
```

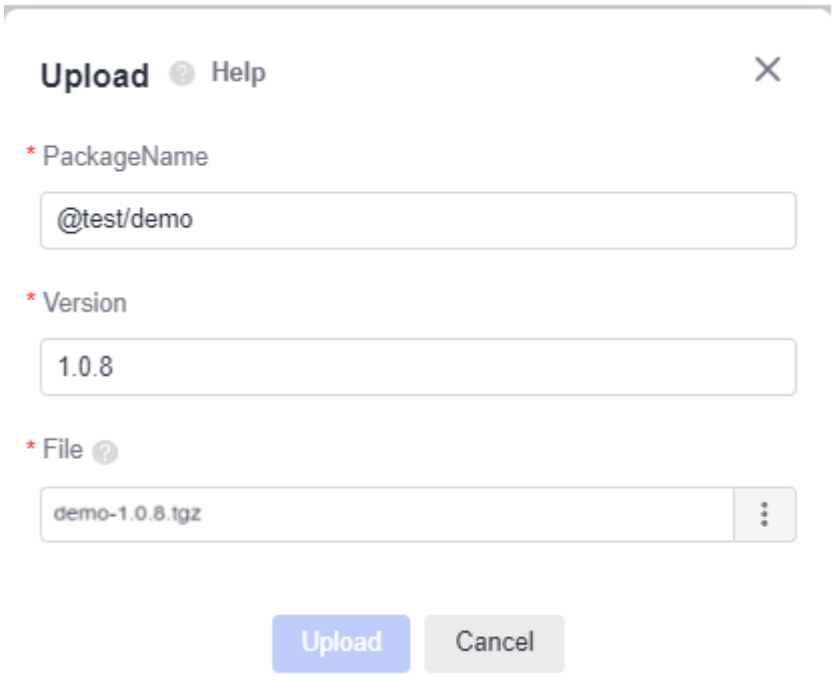
version indicates the version of a package, which is in the x.y.z format.

```
{
  "version": "1.0.0"
}
```

Uploading an npm Component

A self-hosted repo allows you to upload npm component packages in .tgz format. When uploading a package, you need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as the value of **name** in the **package.json** file.
Version	The value must be the same as that of **version** in the **package.json** file.



NOTE

When uploading a component, ensure that the package name starts with a path in the path list added during repository creation. For details, see **Configuring Repository Items** in the help guide.

Example:

The path **@test** is added during the creation of an npm repository.

When uploading an npm component to the repository, make sure that the value of **PackageName** starts with **@test**. If a path outside the path list is used, for example, **@npm**, the upload will fail.

After the upload is successful, you can view the component package in .tgz format in the repository component list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Component

Go (also called Golang) is a programming language developed by Google. Golang 1.11 and later versions support modular package management tools. A module is a unit for source code exchange and versioning of Go. A MOD file is used to identify and manage a module. A ZIP file is a source code package. There are two types of Go modules: v2.0 and later versions and v2.0 and earlier versions. The management of the Go module is different between the two versions.

To upload a Go component, upload a ZIP file and a MOD file. You need to set the following parameters.

Parameter	Description
Zip Path	Complete path of the ZIP file. Valid path formats are: • Versions earlier than v2.0: {moduleName}/@v/{version}.zip • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the file path format is: {moduleName}/vX/@v/vX.X.X.zip – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the file path format is: {moduleName}/@v/vX.X.X+incompatible.zip
Zip File	Directory structure of the ZIP file. Valid directory structure formats are: • Versions earlier than v2.0: {moduleName}@{version} • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the directory structure format is: {moduleName}/vX@{version} – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the directory structure format is: {moduleName}@{version}+incompatible
Mod Path	Complete path of the MOD file. Valid path formats are: • Versions earlier than v2.0: {moduleName}/@v/{version}.mod • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the file path format is: {moduleName}/vX/@v/vX.X.X.mod – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the file path format is: {moduleName}/@v/vX.X.X+incompatible.mod
Mod File	MOD file content. Valid content formats are: • Versions earlier than v2.0: module {moduleName} • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the content format is: module {moduleName}/vX – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the content format is: module {moduleName}

Uploading a PyPI Component

You are advised to go to the project directory (which must contain the **setup.py** configuration file) and run the following command to compress the components

to be uploaded into a wheel (.whl) installation package. By default, the installation package is generated in the **dist** directory of the project directory. The Python software package management tool pip supports only wheel installation packages.

```
python setup.py sdist bdist_wheel
```

You need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as the value of **name** in the **setup.py** file.
Version	The value must be the same as the value of **version** in the **setup.py** file.

After the upload is successful, you can view the installation package in **.whl** format in the repository component list. In addition, the corresponding metadata is generated in the **.pypi** directory, which can be used for pip installation.

Uploading an RPM Component

Introduction to RPM

- Red Hat Package Manager (RPM) is proposed by Red Hat and used by many Linux distributions. It is a software management mechanism that installs required software to Linux in database recording mode.
- You are advised to package and name the RPM binary file according to the following rules:

Software name-*Main version number of the software*.*Minor version number of the software*.*Software revision number*-*Number of software compilation times*.*Hardware platform suitable for the software*.**rpm**

For example, **hello-0.17.2-54.x86_64.rpm**. **hello** is the software name, **0** is the major version number of the software, **17** is the minor version number, **2** is the revision number, **54** is the number of times that the software is compiled, and **x86_64** is the hardware platform suitable for the software.

Software Name	Major Version	Minor Version	Revision No.	Compilation Times	Applicable Hardware Platform
hello	0	17	2	54	x86_64

Note: You need to set the following parameters when uploading components.

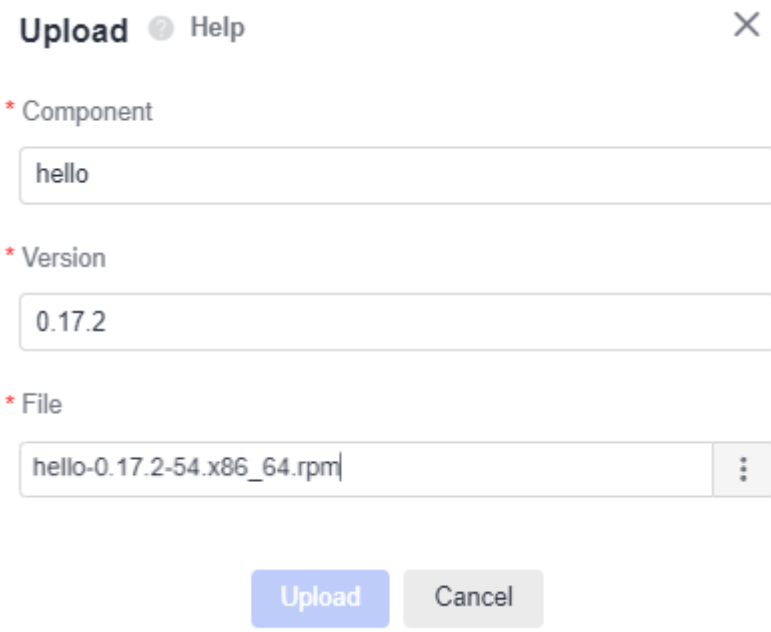
Parameter	Description
Component	Component name

Parameter	Description
Version	Version of the RPM binary package

Step 1 Access the self-hosted repo homepage. In the left pane, choose the repository to which the private component is to be uploaded.

Step 2 Click **Upload**.

Step 3 Set the component parameters, select the file, and click **Upload**.



----End

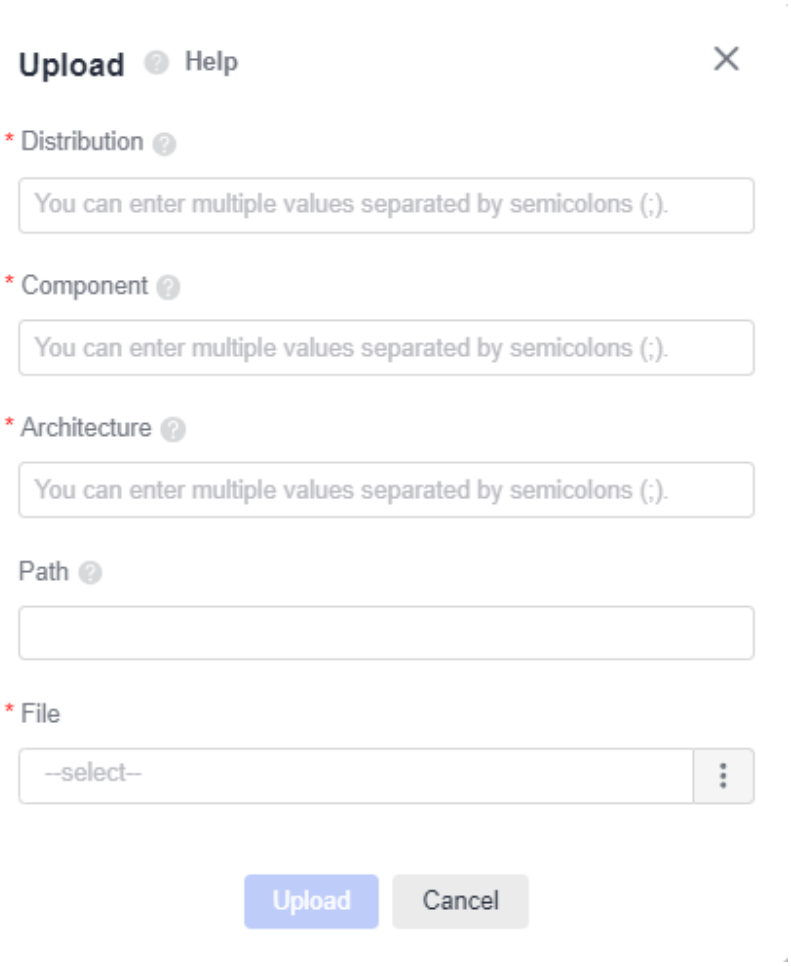
After the upload is successful, you can view the RPM binary package in the repository component list and the corresponding metadata **repodata** directory is generated in the component name directory. You can use Yum to install the component.

Uploading a Debian Component

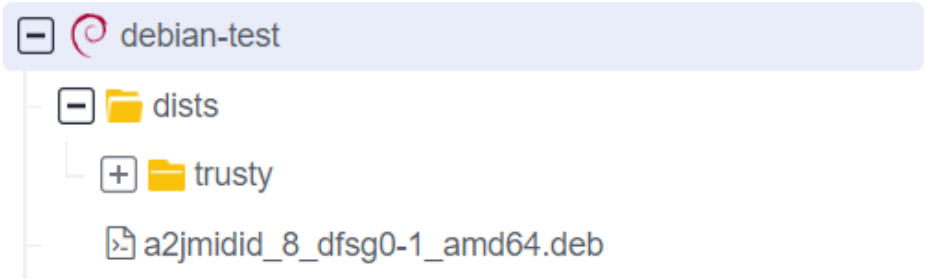
When uploading a Debian component, you need to set the following parameters:

Parameter	Description
Distribution	Release version of the software package
Component	Name of a software package component

Parameter	Description
Architecture	Software package architecture
Path	Path for storing the software package. By default, the software package is uploaded to the root path.
File	Local storage path of the software package



After the upload is successful, you can view the installation package in **.deb** format in the repository component list. In addition, the corresponding metadata is generated in the **dists** directory, which can be used for Debian installation.



Uploading a NuGet Component

The NuGet package is a single ZIP file with the .nupkg extension. As a shareable unit of code, developers can publish it to a dedicated server to share it with other members of the team.

CodeArts Artifact creates a self-hosted NuGet repo to host the NuGet package.

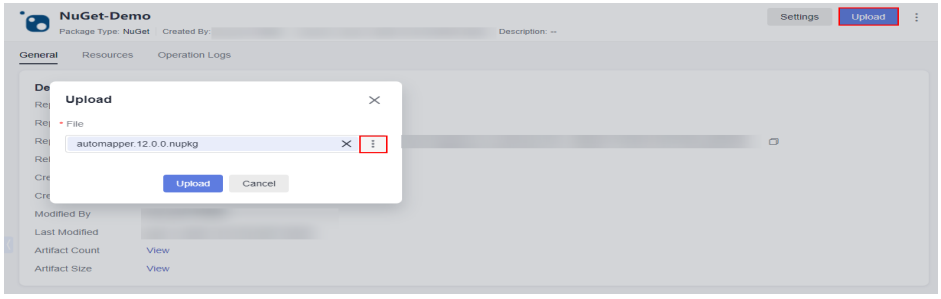
- You are advised to package and name the NuGet file according to the following rules:

  ***Software name*-*Major version number of the software*.nupkg**

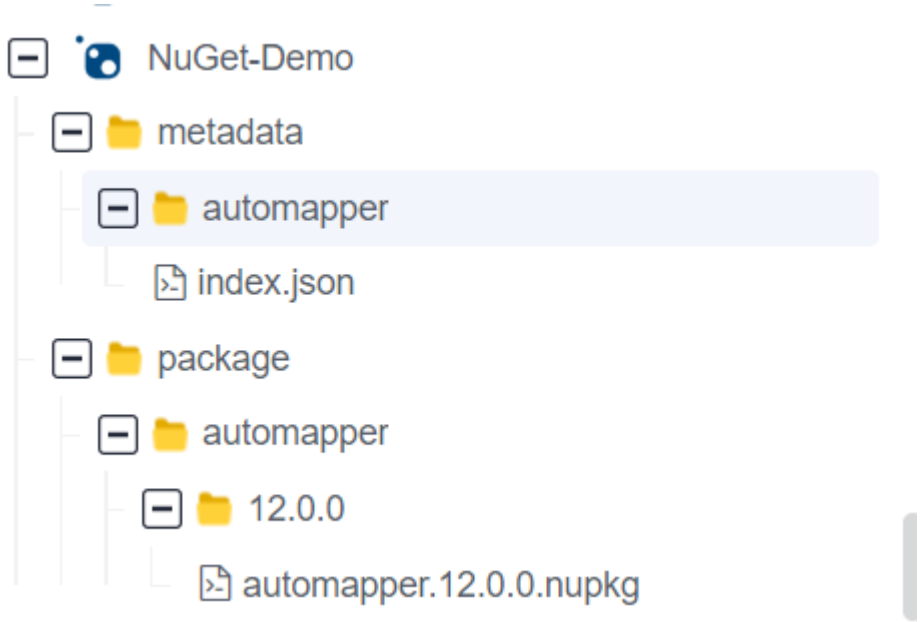
  Example: automapper.12.0.0.nupkg

Step 1 Access the self-hosted repo homepage. In the left pane, choose the NuGet repository to which the private component is to be uploaded.

Step 2 Click **Upload**, select the NuGet file to be uploaded from the local host, and click **Upload**.



Step 3 View components that are successfully uploaded in the repository list.



metadata stores metadata and is named after the component name. **metadata** cannot be deleted. It will be deleted or added when the corresponding component is deleted or restored.

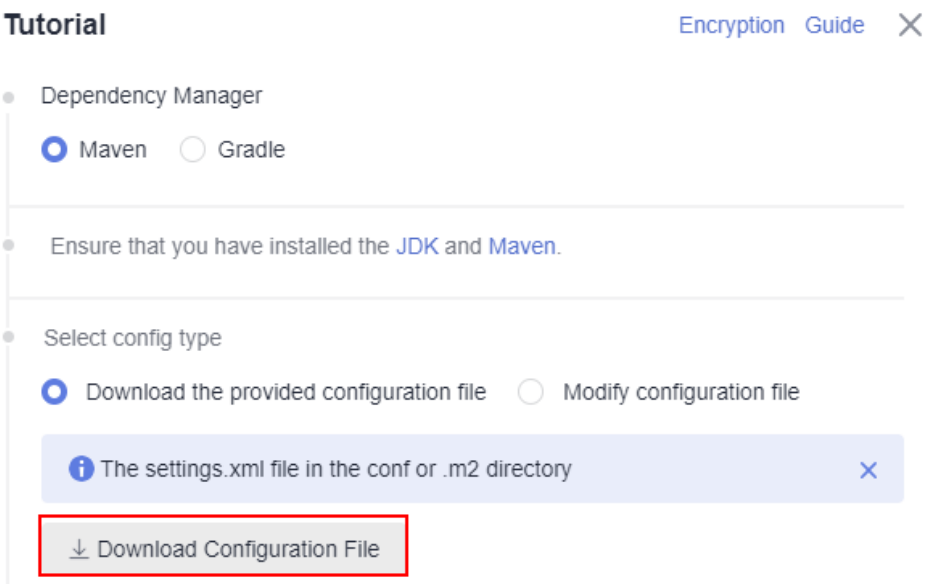
package stores components.

----End

3.9 Uploading and Downloading Private Components

Uploading a Maven Component

- The client tool is Maven. Ensure that the JDK and Maven have been installed.
  a. Download the **settings.xml** file from a self-hosted repo page and replace the downloaded configuration file with the new one or modify the **settings.xml** file of Maven as prompted.



  b. Run the following command to upload a component on the client:

NOTE

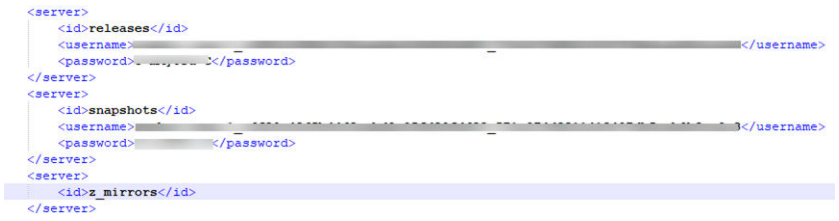
Run the following commands in the directory where the uploaded POM file is located:

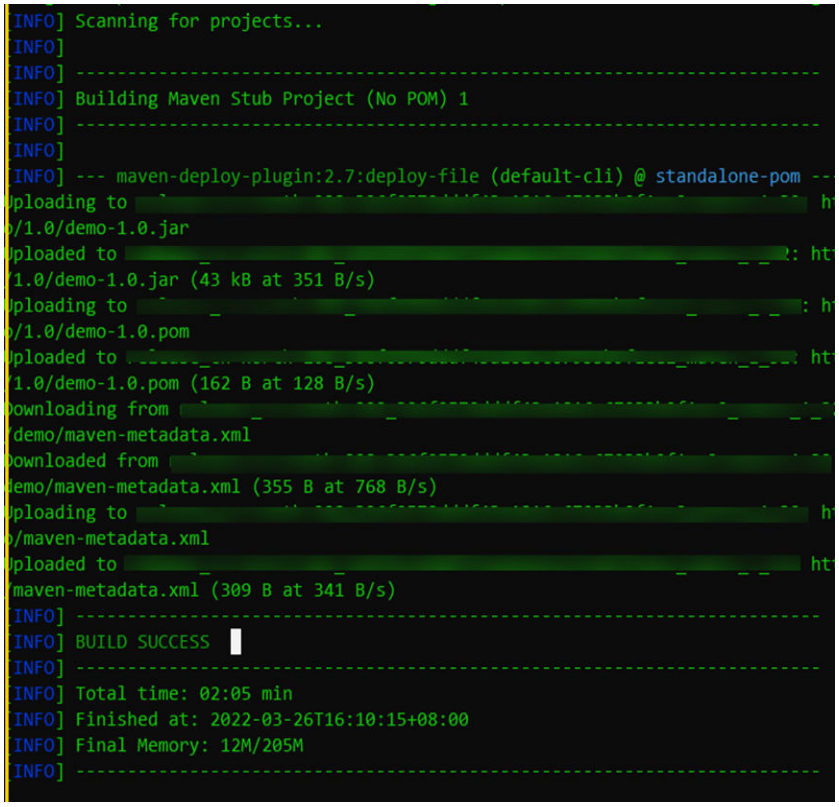
```
mvn deploy:deploy-file -DgroupId={groupId} -DartifactId={artifactId} -Dversion={version} -Dpackaging=jar -Dfile={file_path} -DpomFile={pom_path} -Durl={url} -DrepositoryId={repositoryId} -s {settings_path} -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true -Dmaven.wagon.http.ssl.ignore.validity.dates=true
```

- **Description**
  - **DgroupId**: uploaded group ID
  - **DartifactId**: uploaded artifact ID
  - **Dversion**: version of the uploaded file
  - **Dpackaging**: type of the package to be uploaded, such as JAR, ZIP, and WAR
  - **Dfile**: path of the uploaded entity file
  - **DpomFile**: path of the entity POM file to be uploaded. (For a release version, if this parameter does not exist, the system

automatically generates a POM file. If there are special requirements for the POM file, specify this parameter.)

- The values of **DgroupId**, **DartifactId**, and **Dversion** in the POM file must be the same as those outside the POM file. Otherwise, error 409 is reported.
- Select either **DpomFile** or one of **DgroupId**, **DartifactId**, and **Dversion**.
- **Durl**: path for uploading files to the repository.
- **DrepositoryId**: ID corresponding to the username and password configured in Settings, as shown in the following figure.

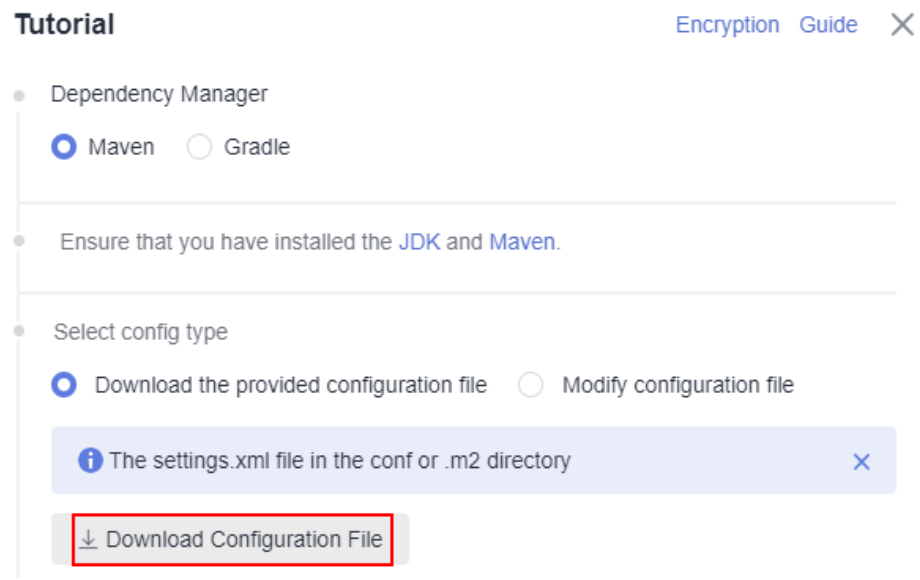




Downloading a Maven Component

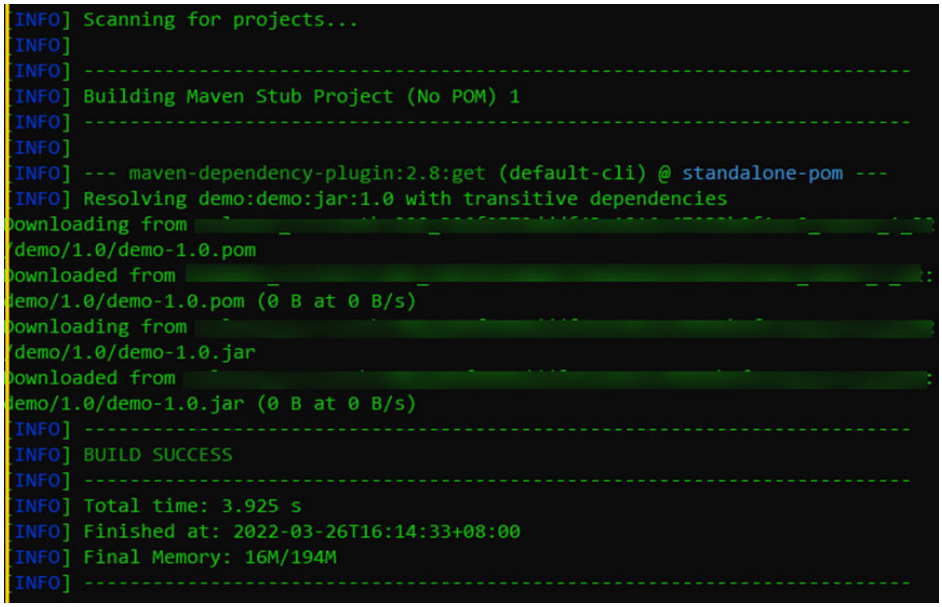
- The client tool is Maven. Ensure that the JDK and Maven have been installed.

  1. Download the **settings.xml** file from a self-hosted repo page and replace the downloaded configuration file with the new one or modify the **settings.xml** file of Maven as prompted.



2. Run the following commands to download the client:

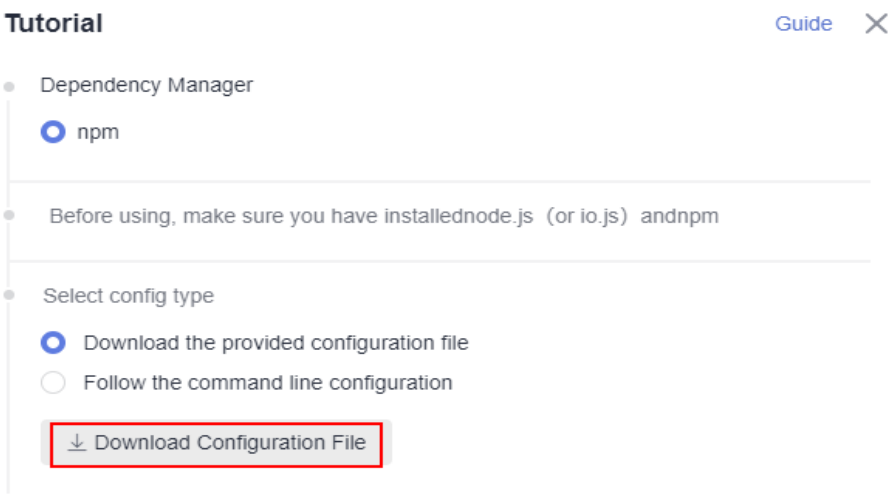
```
mvn dependency:get -DremoteRepositories={repo_url} -DgroupId={groupId} -DartifactId={artifactId} -Dversion={version} -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true -Dmaven.wagon.http.ssl.ignore.validity.dates=true
```



Uploading an npm Component

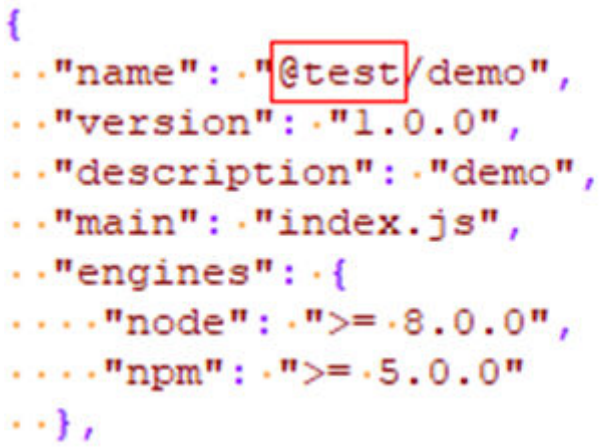
- The client tool is npm. Ensure that **node.js** (or **io.js**) and npm have been installed.

  1. Download the NPMRC file from the self-hosted repo page and save the downloaded NPMRC file as a **.npmrc** file.



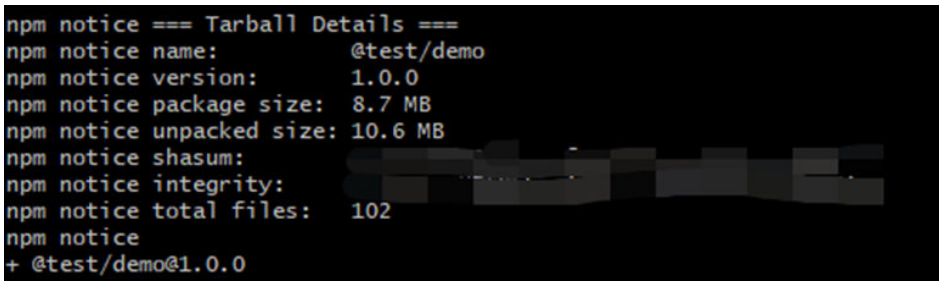
2. Copy the file to the user directory. In Linux, the path is **~/.npmrc** (C:\Users\<*UserName*>\.npmrc).

3. Go to the npm project directory (where the **package.json** file is stored), open the **package.json** file, and add the path information entered during repository creation to the value of the name field.



4. Run the following commands to upload the npm component to the repository:

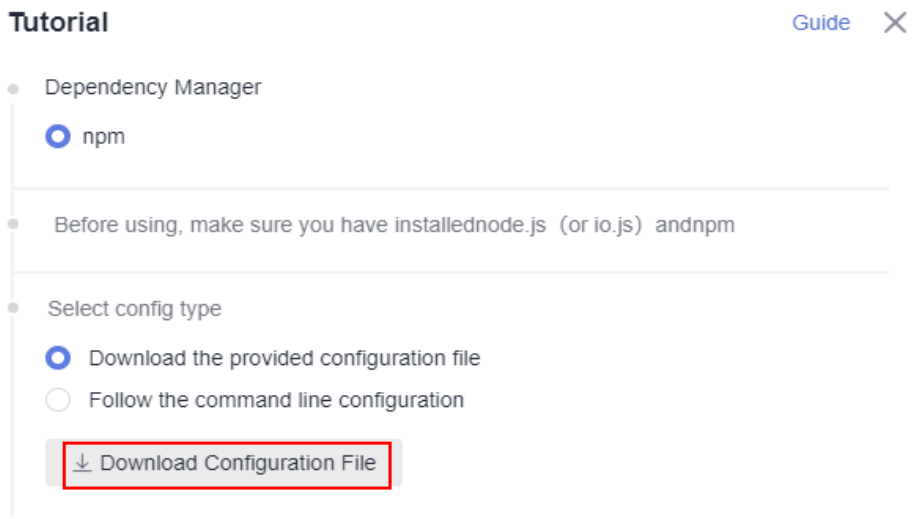
```
npm config set strict-ssl false
npm publish
```



Downloading an npm Component

- The client tool is npm. Ensure that **node.js** (or **io.js**) and npm have been installed.

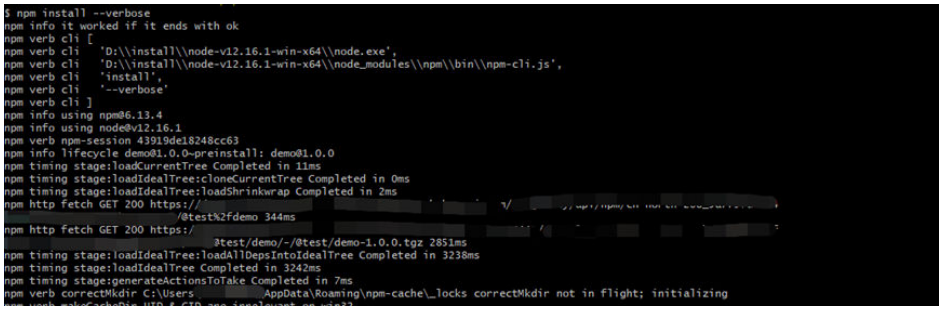
  1. Download the NPMRC file from the self-hosted repo page and save the downloaded NPMRC file as a **.npmrc** file.



  2. Copy the file to the user directory. In Linux, the path is **~/.npmrc**. In Windows, the path is **C:\Users\<*UserName*>\.npmrc**.

  3. Go to the npm project directory (where the **package.json** file is stored) and run the following commands to download the npm dependent component:

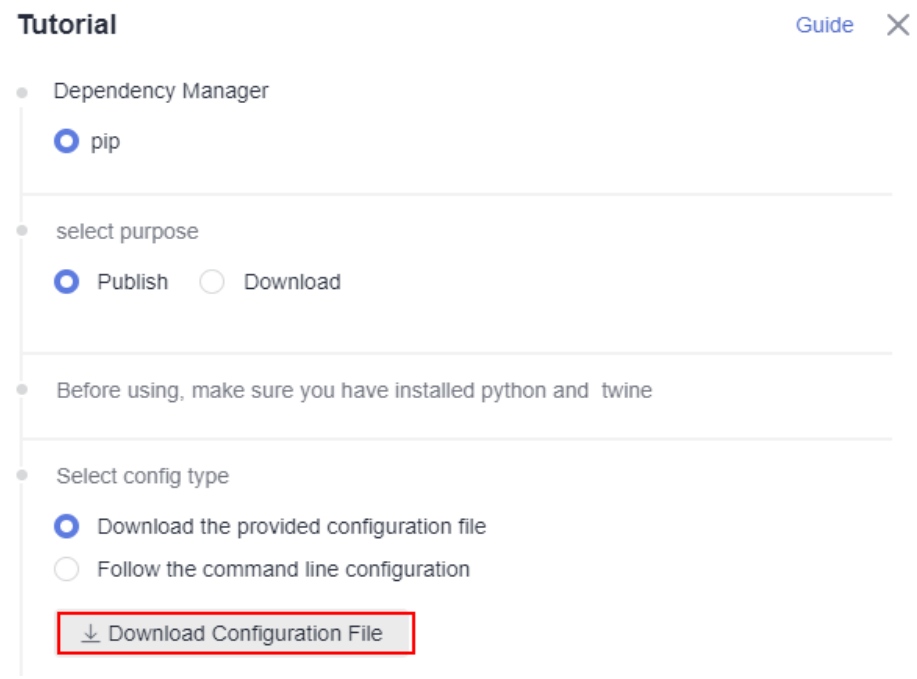
```
npm config set strict-ssl false
npm install –-verbose
```



Uploading a PyPI Component

- The client tools are python and twine. Ensure that python and twine have been installed.

  1. Download the PYPIRC file from the self-hosted repo page and save the downloaded PYPIRC file as a **.pypirc** file.



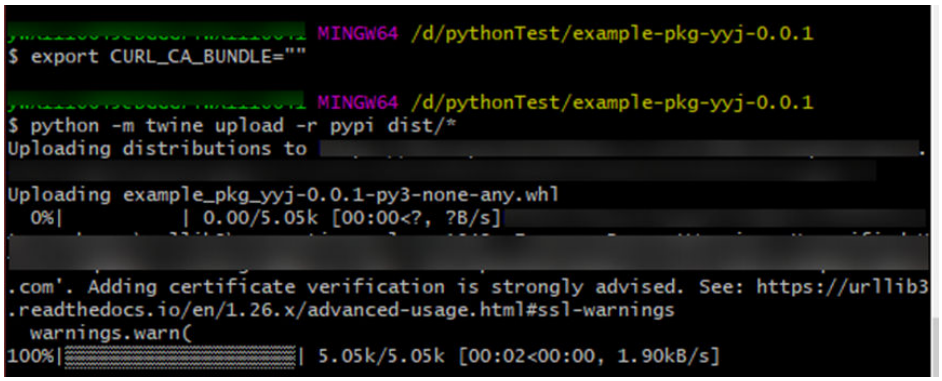
2. Copy the file to the user directory. In Linux, the path is **~/.pypir**c. In Windows, the path is **C:\Users\<***UserName***>\.pypirc**.

3. Go to the Python project directory and run the following command to compress the Python project into a **.whl** package:

```
python setup.py bdist_wheel
```

4. Run the following command to upload the file to the repository:

```
python -m twine upload -r pypi dist/*
```



If a certificate error is reported during the upload, run the following command (use git bash in Windows) to set environment variables to skip certificate verification:

```
export CURL_CA_BUNDLE=""
```

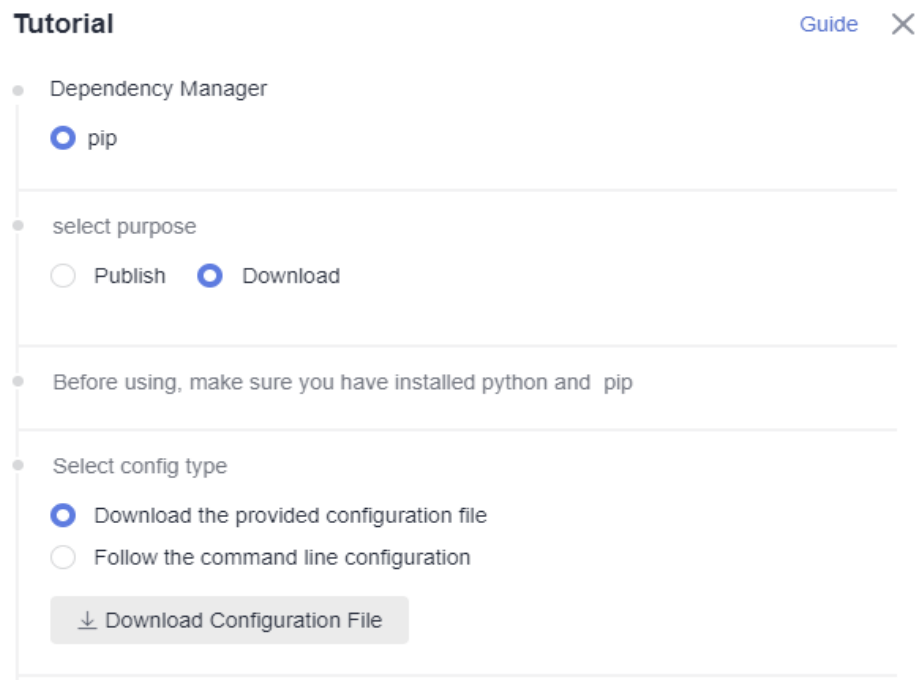
NOTE

The environment variables will be cleared after you log in to the server again, switch to another user, or open the bash window again. Add the environment variables before each upload.

Downloading a PyPI Component

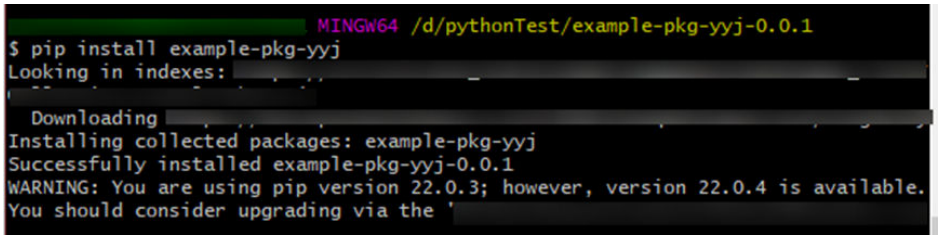
- The client tools are python and pip. Ensure that python and pip have been installed.

1. Download the **pip.ini** file from the self-hosted repo page and copy the file to the user directory. In Linux, the path is **~/.pip/pip.conf** (**C:\Users\<*UserName*>\pip\pip.ini** on Windows)



2. Run the following command to install Python:

```
pip install {package name}
```



Uploading and Downloading a Go Component

The client tool is Go. Ensure that V1.13 or a later version has been installed and the project is a Go module project.

- Go Modules packaging mode and package upload

  This section describes how to build and upload Go components through Go module packaging. The username and password used in the following steps can be obtained from the downloaded configuration file for the Go repository.

  Perform the following steps:

  a. Create a source folder in the working directory.

  ```
  mkdir -p {module}@{version}
  ```

  b. Copy the code source to the source folder.

  ```
  cp -rf . {module}@{version}
  ```

  c. Compress the component into a ZIP package.

  ```
  zip -D -r [package name] [package root directory]
  ```

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

```
curl -k -u {{username}}:{{password}} -X PUT {{repoUrl}}/{filePath} -T {{localFile}}
```

The component 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**.

  Here are a few 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 source to the source 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 component 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 component 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 component ZIP package and the **go.mod** file to the self-hosted repo.

Parameters **username**, **password**, and **repoUrl** can be obtained from the configuration file of the self-hosted repo.

- 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 command is as follows (replace **username**, **password**, and **repoUrl** with the actual values):

```
curl -k -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/v1.0.0.zip -T v1.0.0.zip
curl -k -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 the go.mod file ends 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 source to the source 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 component 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 component 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 component ZIP package and the **go.mod** file to the self-hosted repo.

  Parameters **username**, **password**, and **repoUrl** can be obtained from the configuration file of the self-hosted repo.

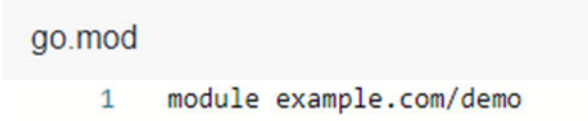
  - For the ZIP package, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.zip** and that of **localFile** is **v2.0.0.zip**.
  - For the **go.mod** file, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.mod** and that of **localFile** is **example.com/demo/v2@v2.0.0/go.mod**.

  The command is 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 the go.mod file does not end with /vX**.

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



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 source to the source 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 component 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 component 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 component ZIP package and the **go.mod** file to the self-hosted repo.

   Parameters **username**, **password**, and **repoUrl** can be obtained from the configuration file of the self-hosted repo.

   - 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 command is as follows (replace **username**, **password**, and **repoUrl** with the actual values):

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

   - **Download the Go component using the Go client.**

     Certificate verification cannot be ignored on the Go client. You need to add the domain name certificate corresponding to the self-hosted repo to the local certificate trustlist and perform the following steps to add the trust certificate list:

     1) Export a certificate.

        ```
        openssl s_client -connect {host}:443 -showcerts </dev/null 2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' |openssl x509 -outform PEM >mycertfile.pem
        openssl x509 -outform der -in mycertfile.pem -out mycertfile.crt
        ```


mycertfile.pem and **mycertfile.crt** are the downloaded certificates.

2) Add the certificate to the root certificate trust list.

3) Run the go commands to download the dependency package.

```
##1. Packages of versions earlier than v2.0
go get -v <moudlename>
##2. v2.0 and later versions
##a. The ZIP package contains go.mod and the path ends with /vN.
go get -v {{moduleName}}/vN@{{version}}
##b. The ZIP package does not contain go.mod or the first line in go.mod does not end with /vN.
go get -v {moduleName}}@{{version}}+incompatible
```

Uploading and Downloading an RPM Component

Use the Linux OS and yum tool. Ensure that the Linux OS is used and yum has been installed.

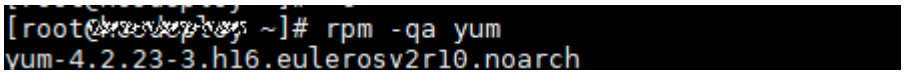
- **Releasing a Component to a Self-Hosted RPM Repo**

Step 1 Check whether the yum tool is installed in Linux.

On the Linux host, run the following command:

```
rpm -qa yum
```

If the following information is displayed, yum has been installed on the server:



Step 2 Log in to the CodeArts homepage and access the self-hosted repo for RPM. 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 commands to upload an RPM component:

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

component, **version**, and **localFile** can be obtained from the RPM component. The **hello-0.17.2-54.x86_64.rpm** component 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 command.

After the command is successfully executed, go to the self-hosted repo and find the uploaded RPM component.

----End

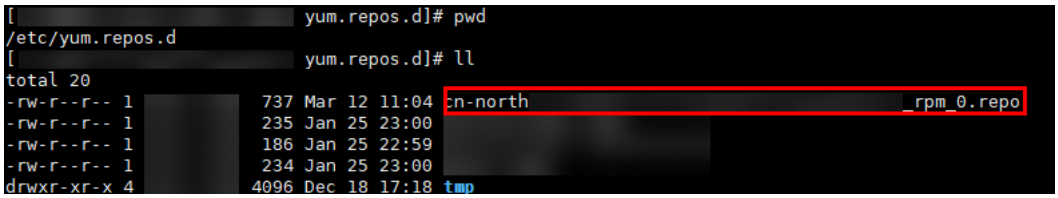
Obtaining a Dependency from a Self-hosted RPM Repo

The following section uses the RPM private component released in **Releasing a Component to a Self-hosted RPM Repo** as an example to describe how to obtain dependency packages from the self-hosted RPM repo.

Step 1 Download the configuration file of the self-hosted RPM repo by referring to **Step 2** and **Step 3** of the released RPM component.

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 component: Replace **hello** with the actual value of **component**.

```
yum install hello
```

----End

Uploading a Conan Component

Conan is a package manager for C and C++ developers. It applies to all operating systems, such as Windows, Linux, OSX, FreeBSD, and Solaris.

Prerequisites

- You have installed the Conan client.
- The Conan repository has been created in the self-hosted repo.

Step 1 Select the corresponding Conan repository from the self-hosted repo page and click **Tutorial** to download the configuration file.

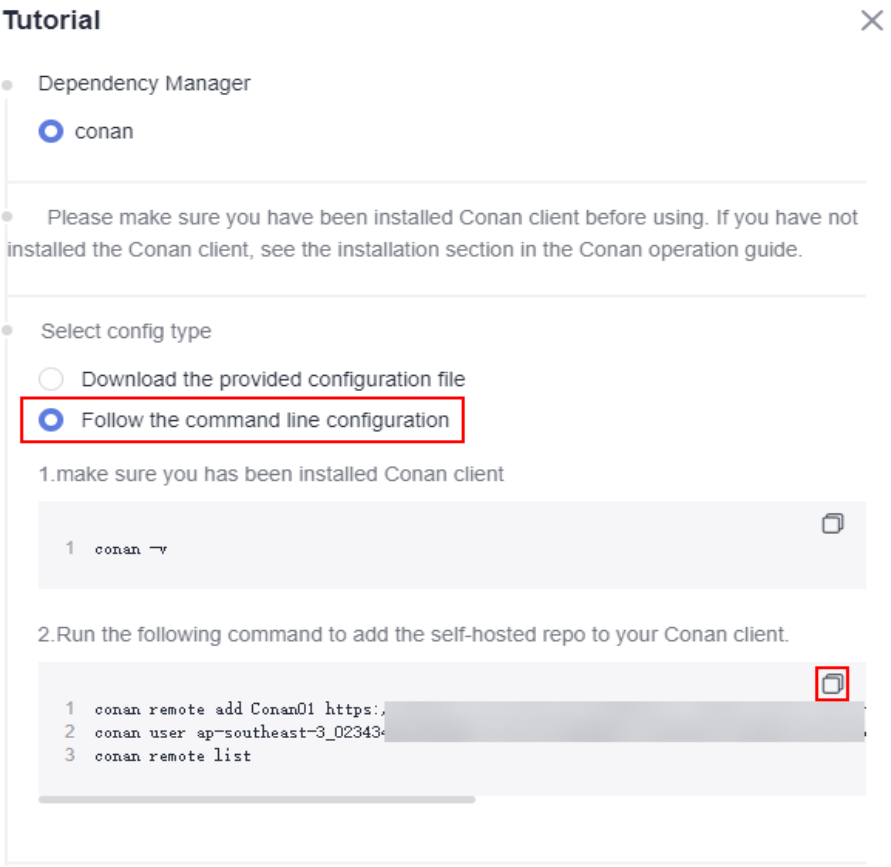
You can replace local Conan configurations with the obtained configuration file (the path is **~/.conan/remotes.json** in Linux or **C:\Users\<UserName>\.conan\remotes.json** in Windows).

Step 2 Copy and run the following commands on the configuration page to add the self-hosted repo to the local Conan client:

```
conan remote add Conan {repository_url}
conan user {user_name} -p={repo_password} -r=Conan
```

Run the following command to check whether the remote repository has been configured on the Conan client:

```
conan remote list
```



Step 3 Upload all software packages to the remote repository. In the example, **my_local_server** is an example remote repository. You can replace it with your own repository.

```
$ conan upload hello/0.1@demo/testing --all -r=my_local_server
```

Step 4 View the software packages that have been uploaded to the remote repository.

```
$ conan search hello/0.1@demo/testing -r=my_local_server
```

----End

Downloading a Conan Component

Step 1 Select the corresponding Conan repository from the self-hosted repo page and click **Tutorial** to download the configuration file.

You can replace local Conan configurations with the obtained configuration file (the path is **~/.conan/remotes.json** in Linux or **C:\Users\<UserName>\.conan\remotes.json** in Windows).

Step 2 Run the following commands to download the Conan dependency package from the remote repository.

```
$ conan install ${package_name}/${package_version}@${package_username}/${channel} -r=cloud_artifact
```

Step 3 Run the following command to view the downloaded Conan software package.

```
$ conan search "*"
```

Step 4 Run the following command to remove the software package from the local cache.

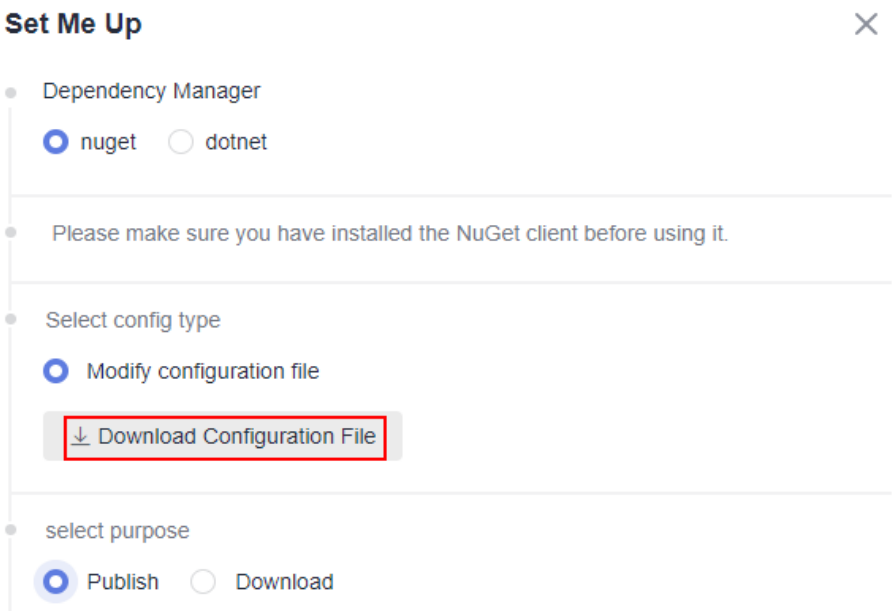
```
$ conan remove ${package_name}/${package_version}@${package_username}/${channel}
```

----End

Uploading a NuGet Component

Ensure that you have installed the NuGet.

Step 1 Select the corresponding NuGet repository from the self-hosted repo page and click **Tutorial** to download the configuration file **NuGet.txt**.



Step 2 Open the downloaded configuration file and run the following commands to add the source:

```
##----------------------NuGet add source----------------------##
nuget sources add -name {repo_name} -source {repo_url} -username {user_name} -password
{repo_password}
```

Step 3 Run the following commands to upload the package. Replace **<PATH_TO_FILE>** with the path of the file to be uploaded and run the upload statement. (If a configuration source exists, use the configured source name as the parameter following **-source**.)

```
##----------------------NuGet Download----------------------##
nuget push <PATH_TO_FILE> -source <SOURCE_NAME>
```

----End

Uploading a .NET Component

Ensure that you have installed .NET.

NOTE

You can use the .NET client only after you have added the trusted server certificate.

- To obtain the Windows trust certificate, perform the following steps:

  1. Export the server certificate.

  ```
  openssl s_client -connect {host}:443 -showcerts </dev/null 2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' |openssl x509 -outform PEM >mycertfile.pem
  openssl x509 -outform der -in mycertfile.pem -out mycertfile.crt
  ```

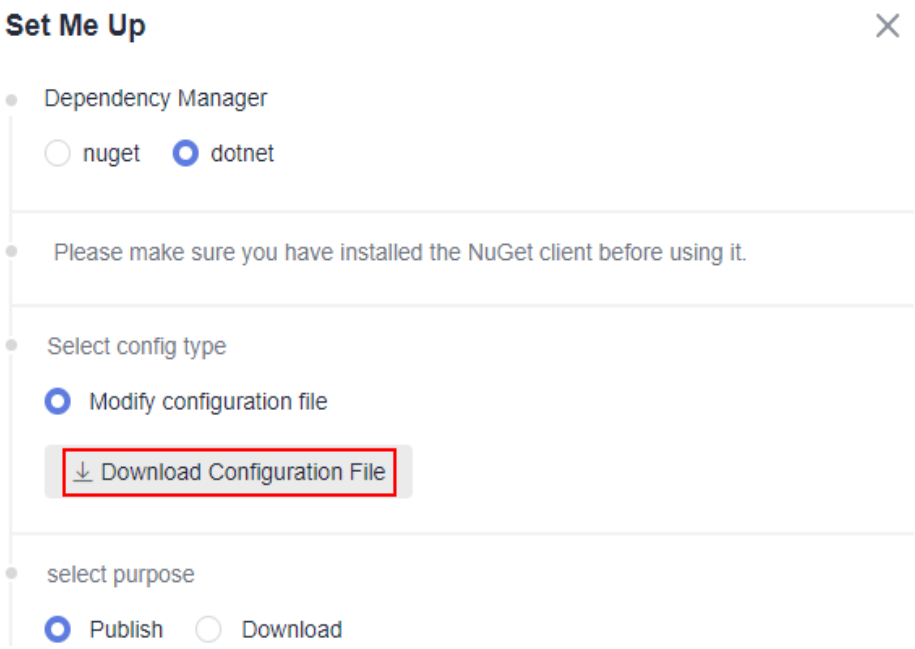
  mycertfile.pem and **mycertfile.crt** are the downloaded certificates.

  2. In Windows, you need to use PowerShell to add the certificate trust.

  Add a certificate

  ```
  Import-Certificate -FilePath "mycertfile.crt" -CertStoreLocation cert:\CurrentUser\Root
  ```

Step 1 Select the corresponding NuGet repository from the self-hosted repo page and click **Tutorial** to download the configuration file **dotnet.txt**.



Step 2 Open the configuration file, find the command under **dotnet add source**, and add the source.

```
##----------------------dotnet add source----------------------##
dotnet nuget add source {repo_url} add -n {repo_name} -u {user_name} -p {repo_password}
```

Step 3 Find the statement under **dotnet upload**, replace <PATH_TO_FILE> with the path of the file to be uploaded, and run the upload statement.

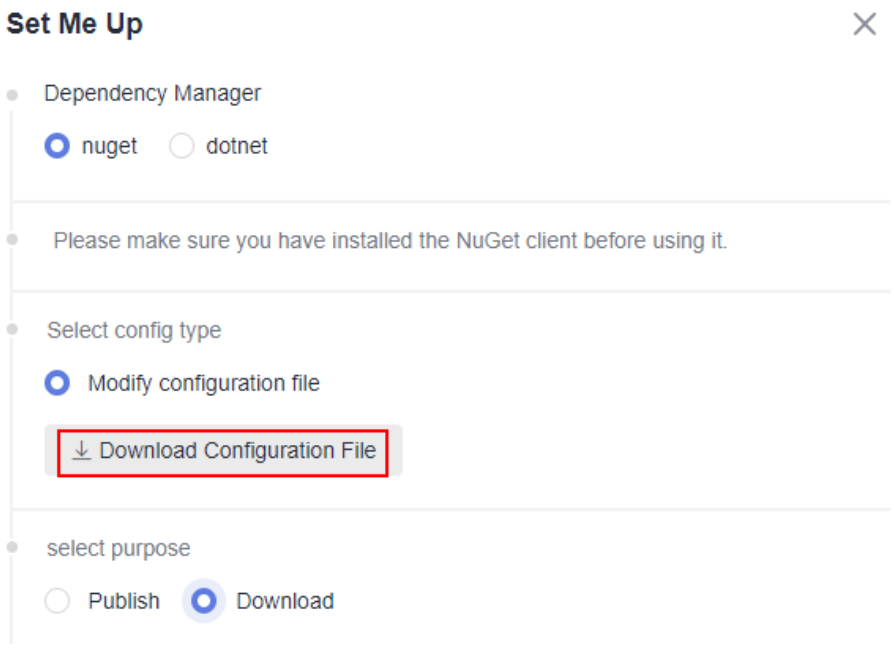
```
##----------------------dotnet upload----------------------##
dotnet nuget push <PATH_TO_FILE> -s {repo_name}
```

----End

Downloading a NuGet Component

Ensure that you have installed the NuGet.

Step 1 Select the corresponding NuGet repository from the self-hosted repo page and click **Tutorial** to download the configuration file **NuGet.txt**.



Step 2 Open the configuration file, find the command under **NuGet add source**, and add the source.

```
##----------------------NuGet add source----------------------##
nuget sources add -name {repo_name} -source{repo_url} -username {user_name} -password
{repo_password}
```

Step 3 Open the configuration file, find the statement under **NuGet Download**, replace <PACKAGE> with the name of the component to be downloaded, and run the download statement. (If a configuration source exists, use the configured source name as the parameter following **-source**.)

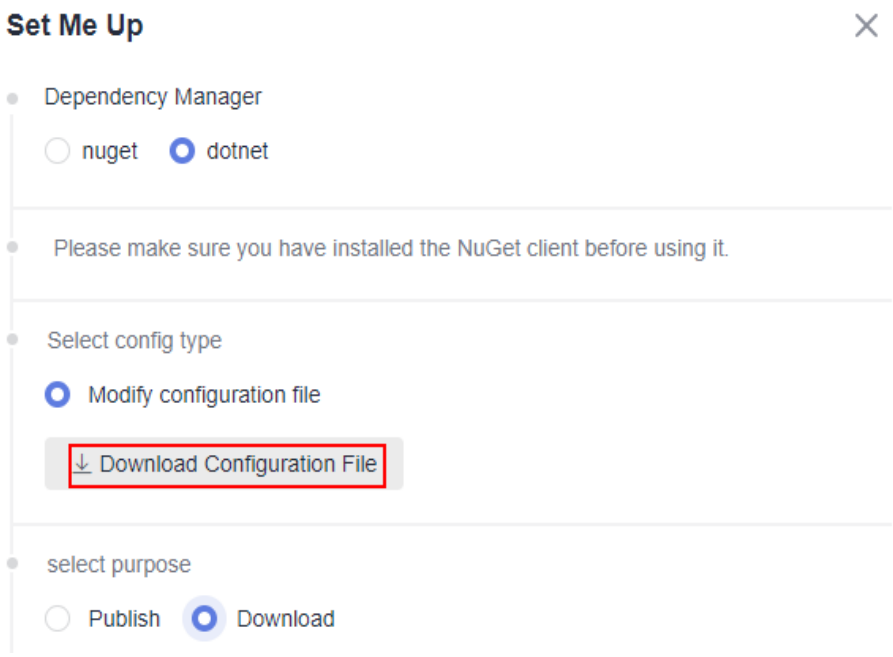
```
##----------------------NuGet Download----------------------##
nuget install <PACKAGE>
```

----End

Downloading a .NET Component

Ensure that you have installed .NET.

Step 1 Select the corresponding NuGet repository from the self-hosted repo page and click **Tutorial** to download the configuration file **dotnet.txt**.



Step 2 Open the configuration file, find the command under **dotnet add source**, and add the source.

```
##----------------------dotnet add source----------------------##
dotnet nuget add source {repo_url} add -n {repo_name} -u {user_name} -p {repo_password}
```

Step 3 Find the statement under **dotnet download**, replace < PACKAGE > with the name of the component to be downloaded, and run the download statement.

```
##----------------------dotnet download----------------------##
dotnet add package <PACKAGE>
```

----End

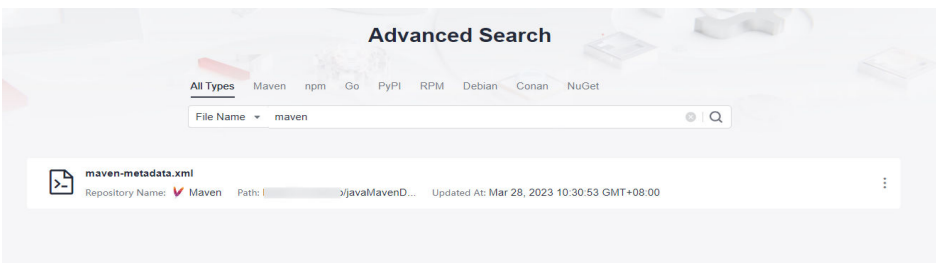
3.10 Managing Private Components

Searching for Private Components

Step 1 Go to the **Self-hosted Repos** page and click **Advanced Search** in the upper left corner of the page.

Step 2 In the upper part of the page, you can select the repository where the component to be queried is located. By default, all repositories are selected.

Step 3 Enter the keyword of the file name in the search box, and click to search for the component.

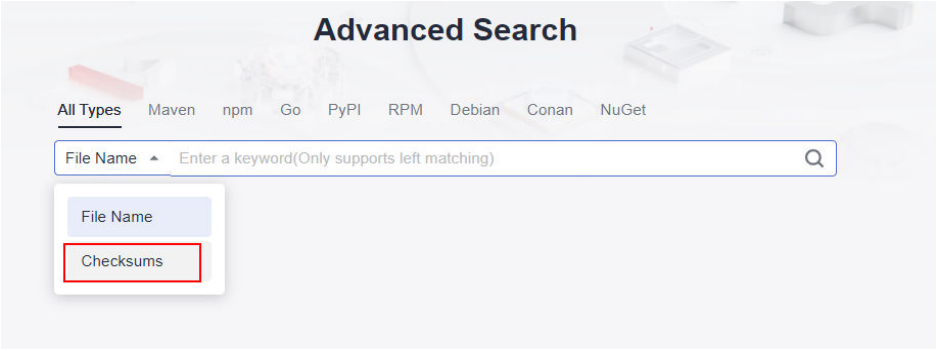


Step 4 Click the **File Name** to view its details.

----End

- Searching for Artifacts by Checksums

1. Click the drop-down list on the left of the search box and select **Checksums** (The default value is the file name).



2. You can also enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click to find the corresponding component.

Downloading a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, locate the private component to be downloaded, and click the component name.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Download**.

----End

Deleting a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, locate the private component to be downloaded, and click the component name.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Delete**.

Step 3 In the displayed dialog box, click **Yes**.

----End

Following or Unfollowing a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, locate the private component to be downloaded, and click the component name.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Unfollowed** on the right of the page.

When the icon changes to  , click **Following** in the lower left corner of the page to view the list of followed components. Click the **path** value in the list to go to the component details page.

----End

3.11 Managing the Recycle Bin

Repositories and components deleted from a self-hosted repo are moved to the recycle bin, where you can manage them.

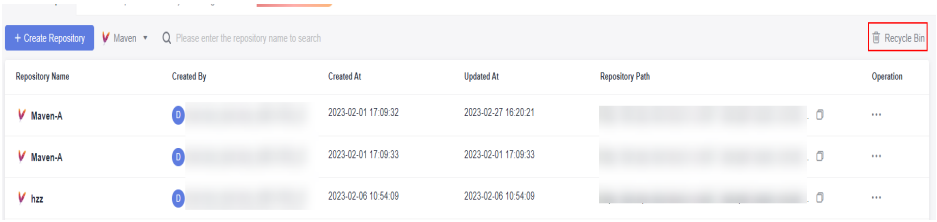
The self-hosted repo provides the recycle bin entry both from the homepage and project pages.

Homepage Recycle Bin

You can process all deleted components in the recycle bin on the CodeArts Artifact homepage.

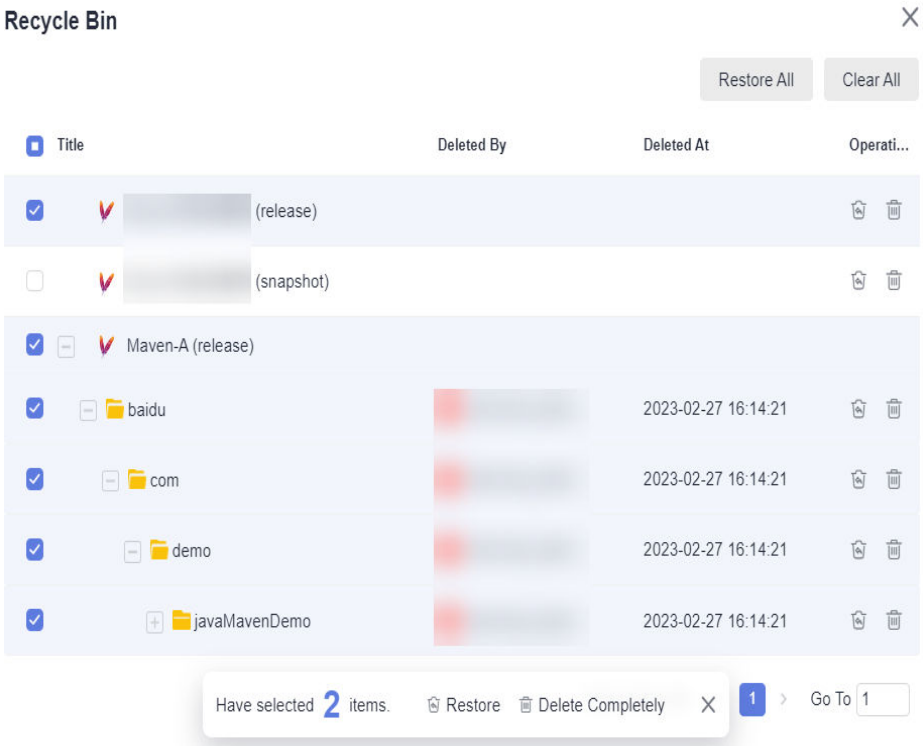
Step 1 Access the self-hosted repo by following instructions in **Accessing Through the Homepage**.

Step 2 Click **Recycle Bin** in the upper right corner of the page. The **Recycle Bin** page is displayed on the right.



Step 3 Delete or restore the repositories and components in the list as required.

If the icons  and  are both displayed in the **Operation** column, the repository shown in the corresponding row has been deleted. Otherwise, the repository shown in the corresponding row has not been deleted but the components in it have been deleted. You can click the repository name to view the deleted components in the repository.



Available operations are as follows.

Operation Type	Operation	Description
Restore	Restore a repository	Click in the **Operation** column to restore the repository.
	Restore a component	Go to the repository where the component to restore is located, and click in the **Operation** column to restore the component.
	Batch restore components	Go to the repository where the components to restore are located, select the components, and click **Restore** below the list to restore the components.
	Restore all	Click **Restore All** in the upper right corner of the page to restore all repositories and components in the recycle bin.
Delete	Delete a repository	Click in the **Operation** column to delete a repository.
	Delete a component	Go to the repository where the component to delete is located, and click in the **Operation** column to delete the component.

Operation Type	Operation	Description
	Delete components in batches	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to delete the components.
	Clear recycle bin	Click **Clear All** to delete all repositories and components in the recycle bin.

----End

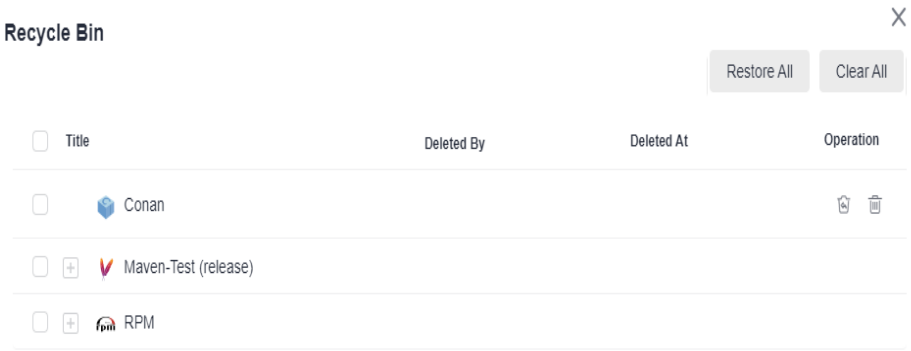
Recycle Bin in a Project

Step 1 Access the self-hosted repo by following instructions in **Accessing Through a Specific Project**.

Step 2 In the lower left corner of the page, click **Recycle Bin**. The **Recycle Bin** page is displayed on the right.

Step 3 Delete or restore the repositories and components in the list as required.

If the icons and are both displayed in the **Operation** column, the repository shown in the corresponding row has been deleted. Otherwise, the repository shown in the corresponding row has not been deleted but the components in it have been deleted. You can click the repository name to view the deleted components in the repository.



Available operations are as follows.

Operation Type	Operation	Description
Restore	Restore a repository	Click in the **Operation** column to restore the repository.

Operation Type	Operation	Description
	Restore a component	Go to the repository where the component to restore is located, and click  in the **Operation** column to restore the component.
	Batch restore components	Go to the repository where the components to restore are located, select the components, and click **Restore** below the list to restore the components.
	Restore all	Click **Restore All** in the upper right corner of the page to restore all repositories and components in the recycle bin.
Delete	Delete a repository	Click  in the **Operation** column to delete a repository.
	Delete a component	Go to the repository where the component to delete is located, and click  in the **Operation** column to delete the component.
	Delete components in batches	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to delete the components.
	Clear recycle bin	Click **Clear All** to delete all repositories and components in the recycle bin.

----End

4 Self-hosted Repo (Legacy Version)

4.1 Introduction

A self-hosted repo manages private components, supporting Maven, npm, Go, PyPI, RPM, and Debian.

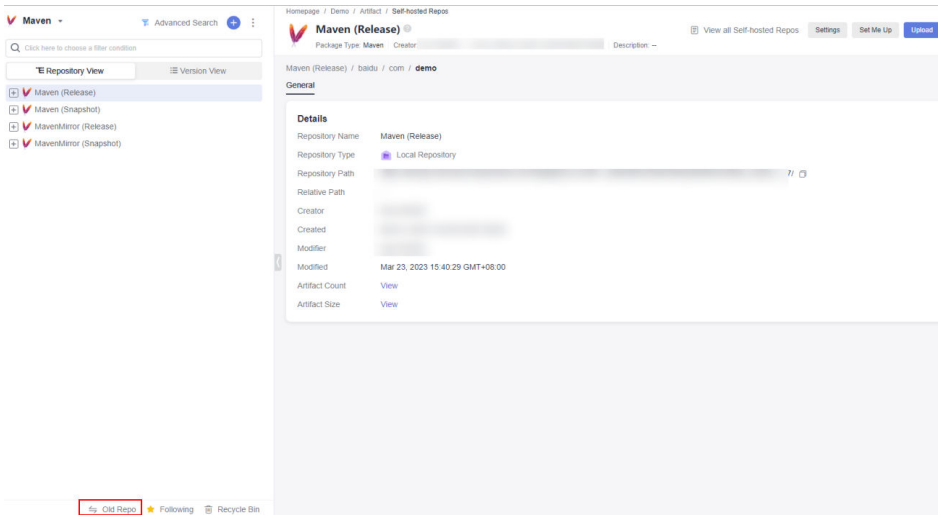
A self-hosted repo provides the following functions:

- Repository management: includes creating a repository, editing basic repository information, managing repository permissions, and connecting the repository to the local development environment.
- Private component management: includes uploading, downloading, searching for, and deleting private components, and managing the recycle bin.

4.2 Accessing a Self-hosted Repo (Legacy Version)

Step 1 Access the self-hosted repo by following instructions in **Accessing Through a Specific Project**.

Step 2 Click **Old Repo** in the lower left corner.



----End

4.3 Managing Self-hosted Repos

You can edit repository descriptions, add paths, delete repositories, and manage user permissions.

Editing Repository Descriptions and Paths

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be edited.

Step 2 Click **Settings** on the right of the page to display the basic information about a repository.

Step 3 Edit the repository description as required and click **Submit**.

NOTE

On the **Basic Information** tab page, the repository name and repository format cannot be modified.

On the **Basic Information** page of the repository, enter the path and click + to add paths for the npm, Go, PyPI, and RPM repositories.

Click 🗑 to delete a path.

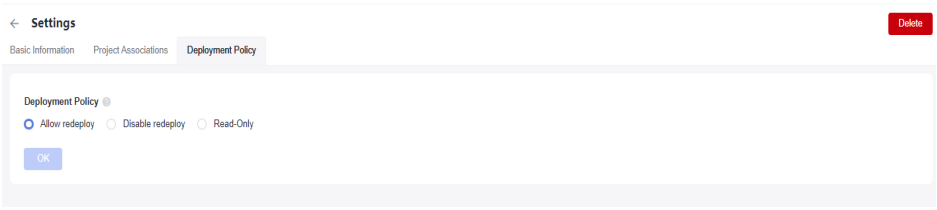
----End

Configuring Deployment Policies

The self-hosted repo supports three version policies: **Allow redeploy**, **Disable redeploy**, and **Read-only**. You can set whether to allow artifacts in the same path to be uploaded and overwrite the original package.

Step 1 Access the self-hosted repo homepage. In the left pane, click the repository name.

Step 2 Click **Settings** on the right of the page. The basic information about the repository is displayed. Click the **Deployment Policies** tab.



- **Allow redeploy** (selected by default): Artifacts in the same path can be uploaded. After being uploaded, the original package will be overwritten.
- **Disable redeploy**: Artifacts in the same path cannot be uploaded.
- **Read-only**: Artifacts cannot be uploaded, updated, or deleted. You can download an uploaded artifact.

Step 3 Click **OK**.

----End

Deleting a Repository

You can delete a self-hosted repo. Deleted repositories are moved to the recycle bin.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be deleted.

Step 2 Click **Settings** on the right of the page to display the basic information about a repository.

Step 3 Click **Delete**. Check that the deleted repository is no longer displayed in the repository list in the left pane.

----End

Managing User Permissions

The tenant account can add members to or delete members from a self-hosted repository. The administrator of each repository can manage roles of the members in the repository.

NOTE

This section describes how to set user permissions only for the existing self-hosted repo of the old version.

Step 1 Click the username in the upper right corner, and select **All Account Settings** from the drop-down list.

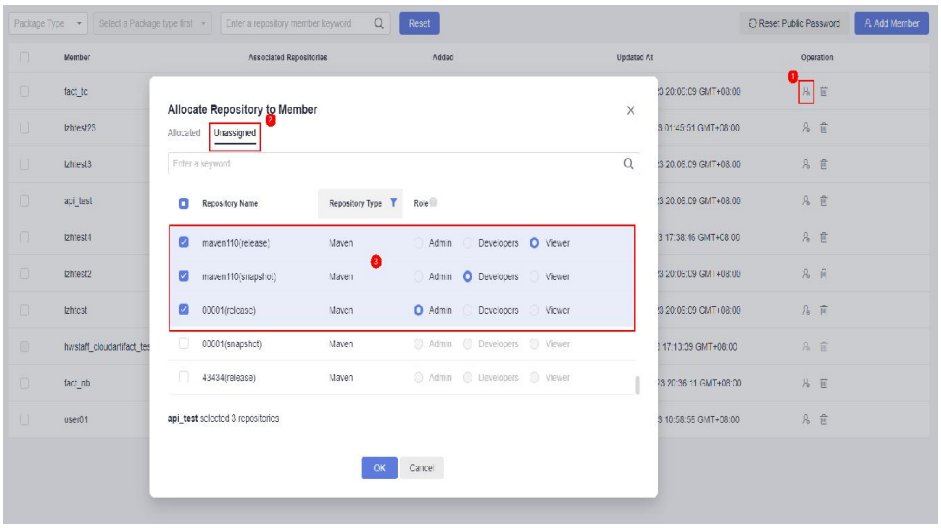
Step 2 In the left navigation pane, choose **Artifact** > **User Permissions**.

Step 3 Click **Add Member**, select a member, and click **OK**.

Step 4 Assign roles to the member.

1. In the **Operation** column of the target member, click .
2. Click the **Assigned** tab.
3. Select the desired repositories and roles, and click **OK**.

Figure 4-1 Assigning repository roles



----End

A tenant administrator can add members to or delete members from a self-hosted repo and manage the roles of members in the repository.

The following table lists the operation permissions of each repository role.

Operation/Role	Tenant Administrator			Non-Tenant Administrator		
	Repository Administrator	Developer	Viewer	Repository Administrator	Developer	Viewer
Create a self-hosted repo	√	√	√	×	×	×
Edit a self-hosted repo	√	×	×	×	×	×
Manage the association between repositories and projects	√	√	√	×	×	×

Upload a private component	√	√	×	√	√	×
Download a component	√	√	√	√	√	√
Delete a component	√	√	×	√	√	×
Restore a component	√	√	×	√	√	×
Permanently delete a component	√	√	×	√	√	×
Delete a repository	√	×	×	×	×	×
Restore a repository	√	√	×	×	×	×
Permanently delete a repository	√	×	×	×	×	×
Clear recycle bin	√	√	√	×	×	×
Restore all	√	√	√	×	×	×
Manage user permissions	√	√	√	√	×	×
Reset the password of a public account	√	√	√	×	×	×

You can also perform the following operations on the **Set User Permissions** page.

Operation	Description
Delete members	To delete a member, click in the corresponding row. To delete multiple members, select them and click **Delete**.
Modify member permissions	Click . In the displayed dialog box, select the repository name, and click **OK**.
View members	Select the package type and repository name in the upper left corner of the page. The member list of the repository is displayed. Click **Reset** in the upper part of the page to view the list of all members.
Remove a member	Select a repository and click in the member list to remove the member from the repository. **NOTE** Removing a member from a repository does not affect the role and permissions of the member in other repositories. Deleting a member is to remove it from all the related repositories. The corresponding permissions of the member for those repositories are also deleted.
Search for a member	Enter a member name or keyword in the search box in the upper part of the page and click to search for a repository member.
Reset the password of a public account (This function is available only to tenant administrators.)	The password of a public account is used by the CodeArts Build to upload and download components to the self-hosted repo and is invisible on the page. Click **Reset Public Password** in the upper right corner of the page to reset the password.

Set-Up

You can connect the self-hosted repo to a local development environment so that private components in the self-hosted repo can be used during local development.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of the repository to be connected to the local development environment.

Step 2 Click **Set Me Up** on the right of the page.

Step 3 In the displayed dialog box, click **Download Configuration File** to download the configuration file to your local directory.

Step 4 Copy the downloaded file to the corresponding directory based on the instructions in the **Information** dialog box.

----End

Resetting the Repository Password

You can reset the password in the self-hosted repo configuration file. After the password is reset, download the configuration file again to replace the original file.

Step 1 Access the self-hosted repo homepage. Click ⋮ above the repository list on the left and choose **Reset Repository Password**.

Step 2 In the displayed dialog box, click **OK**. Check that a message is displayed indicating the password has been reset.

----End

Obtaining the Self-hosted Repo Path

The path of the self-hosted repo will be used when you connect the repository to the local development environment. You can perform the following operations to obtain the path:

Step 1 Access the self-hosted repo homepage. In the left pane, click the repository name.

Step 2 The path of the self-hosted repo is displayed in the repository details on the page. You can click  to obtain the path.

----End

Managing the Association Between the Self-hosted Maven Repo and Project

When uploading a Maven component to the self-hosted repo through a build task, specify the repository path in the **Build with Maven** step.

- **Do not configure POM**: The dependency package is not released to the self-hosted repo.
- **Configure all POMs**: If you run the **mvn deploy** command, the dependency package is released to the specified release repository and snapshot repository.

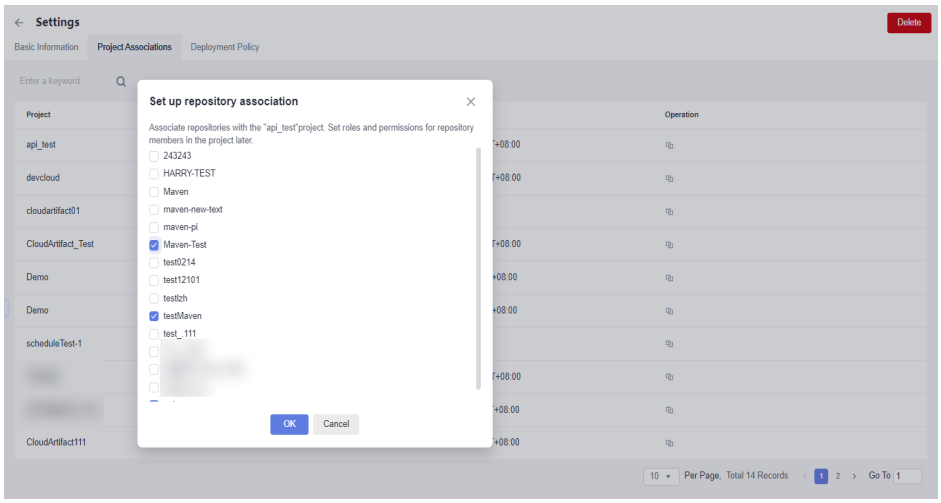
After the self-hosted Maven repo is associated with a project, you can select the repository in the build step of the build task in the project.

Step 1 Access the self-hosted repo homepage. In the left pane, click the name of a self-hosted Maven repo.

Step 2 Click **Settings** on the right of the page, and choose **Project Associations**.

Step 3 In the **Operation** column of the target project in the self-hosted Maven repo, click  .

Step 4 In the displayed dialog box, select the repository name, and click **OK**.



After an "Operation successful" message is displayed, the value of **Associated Repositories** for the project will be updated according to the number of selected repositories.

----End

4.4 Uploading a Private Component

Only repository administrators and developers can upload private components. You can set repository roles on the **User Permissions** page.

Procedure

To upload a component, perform the following steps:

Step 1 Access the self-hosted repo homepage. In the left pane, choose the repository to which the private component is to be uploaded.

Step 2 Click **Upload**.

Step 3 Set the component parameters, select the file, and click **Upload**.

The detailed configuration for each type of component is described below.

NOTE

You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to the self-hosted repo.

----End

Introduction to Maven Components

- POM: Project Object Model (POM) is the basic working unit of a Maven project. It is an XML file that contains basic project information to describe how to build a project and declare project dependencies. When a build task is executed, Maven searches for POM in the current directory, reads the POM, obtains the required configuration information, and constructs the target component.

- Maven coordinates: X, Y, and Z are used to uniquely identify a point in the three-dimensional space. In Maven, GAV is used to identify a unique Maven component package. GAV is short for **groupId**, **artifactId**, and **version**. **groupId** indicates a company or organization. For example, Maven core components are in the **org.apache.maven** organization. **artifactId** indicates the name of a component package. **version** indicates the version of the component package.
- Maven dependency: The dependency list is the cornerstone of POM. The building and running of most projects depend on the dependency on other components. Add the dependency list to the POM file. If the App component depends on the **App-Core** and **App-Data** components, the configuration is as follows:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.companyname.groupname</groupId>
    <artifactId>App</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Core</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
    <dependencies>
      <dependency>
        <groupId>com.companyname.groupname</groupId>
        <artifactId>App-Data</artifactId>
        <version>1.0</version>
      </dependency>
    </dependencies>
</project>
```

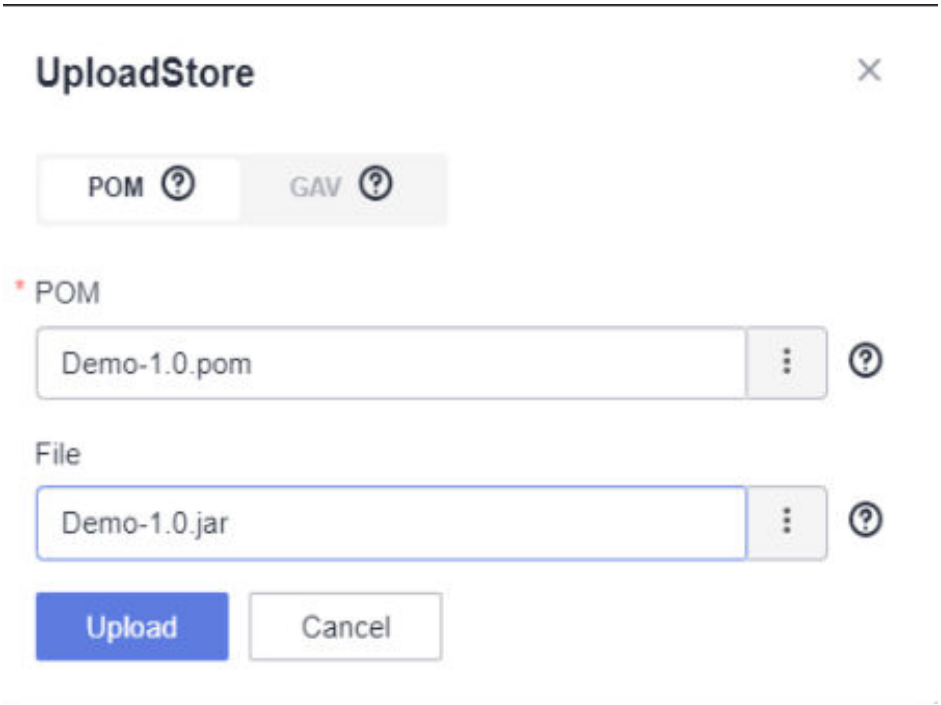
Uploading a Maven Component

A self-hosted repo supports two upload modes: **POM** and **GAV**.

Upload Mode	Description
POM	GAV parameters are obtained from the **POM** file. The system retains transitive dependencies of components.
GAV	GAV, short for **Group ID**, **Artifact ID**, and **Version**, is the unique identifier of a JAR package. In this mode, GAV parameters are manually specified. The system automatically generates a **POM** file without any transitive dependency.

- POM

  In POM mode, you can upload only the **POM** file or upload the **POM** file and related components. The name of the uploaded file must be the same as the **artifactId** value and **version** value in the **POM** file. As shown in the following figure, the **artifactId** value is **demo** and the **version** value is **1.0** in the POM. The uploaded file must be **demo-1.0.jar**.



The **POM** file structure is as follows:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>demo</groupId>
  <artifactId>demo</artifactId>
  <version>1.0</version>
</project>
```

NOTE

The **modelVersion** tag must exist and the value must be **4.0.0**, indicating that **Maven2** is used.

If you upload files in both the **POM** and **File** area, the **artifactId** and **version** parameter values in the uploaded **POM** file must match the name of the file uploaded in the **File** area. For example, if **artifactId** is **demo** and **version** is **1.0** in the **POM** file, the name of the file to be uploaded in the **File** area must be **demo-1.0**. Otherwise, the upload will fail.

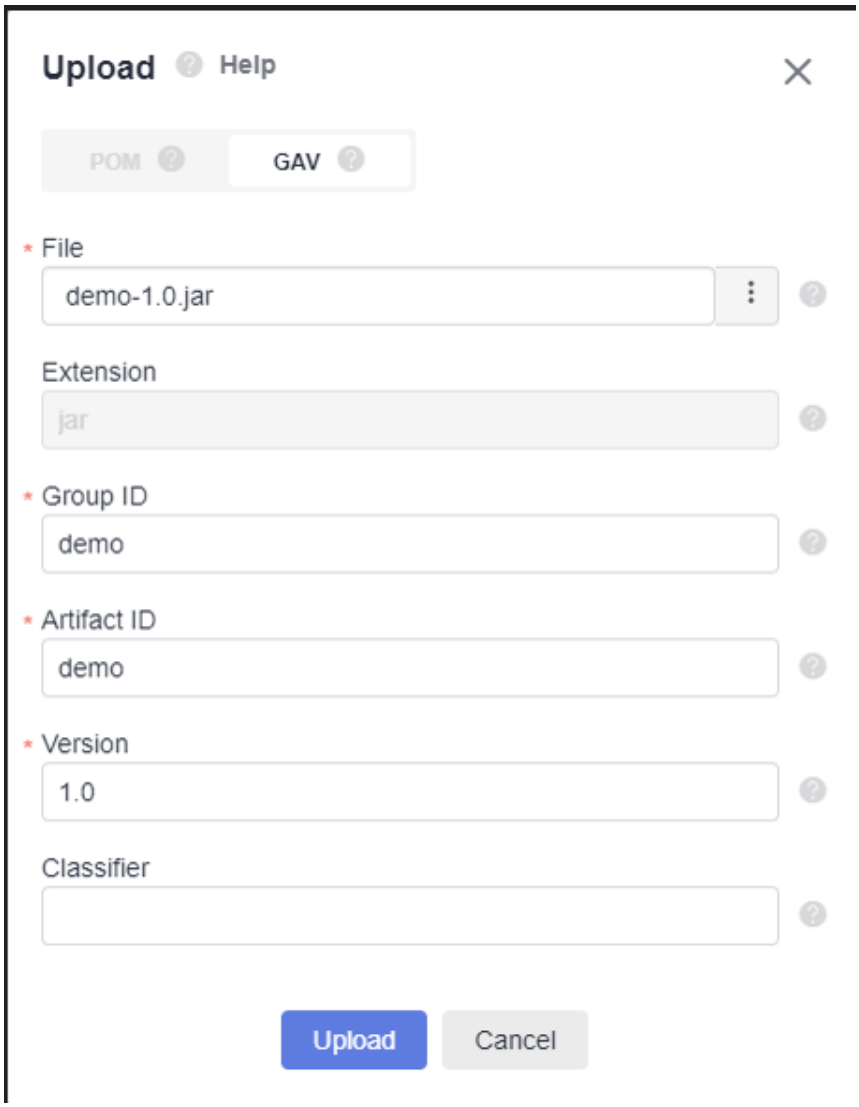
- GAV

  In the GAV mode, the **Group ID**, **Artifact ID**, and **Version** parameters must be manually specified and they determine the name of the file to be uploaded. **Extension** indicates the packaging type, which determines the type of the file to be uploaded.

  Classifiers are used to distinguish artifacts that are constructed from the same POM and have different contents. This field is optional. It can contain letters, digits, underscores (_), hyphens (-), and dots (.). If you enter a value, it will be appended to the file name.

  Common Usage Scenario

  – Differentiate versions by names, such as **demo-1.0-jdk13.jar** and **demo-1.0-jdk15.jar**.
  – Differentiate usage by names, such as **demo-1.0-javadoc.jar** and **demo-1.0-sources.jar**.



Introduction to npm Components

Node Package Manager (npm) is a JavaScript package management tool. The npm component package is the object managed by the npm, and the self-hosted npm repo is for managing and storing the npm component package.

The npm component package consists of the structure and file description.

- Package structure: organizes various files in a package, such as source code files and resource files.
- Description file: describes package information. Example: **package.json**, **bin**, and **lib** files

The **package.json** file in the package is a description file of a project or module package. It contains information such as the name, description, version, and author. The **npm install** command downloads all dependent modules based on this file.

An example of the **package.json** file is as follows:

```
{
  "name": "third_use",          //Package name
```



```
  "version": "0.0.1",           //Version number
"description": "this is a test project", // Description
  "main": "index.js",           //Entry file
  "scripts": {                  //Script commands
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [                 //Keyword
    "show"
  ],
  "author": "f",                //Developer name
  "license": "ISC",             //License agreement
  "dependencies": {             //Project production dependencies
    "jquery": "^3.6.0",
    "mysql": "^2.18.1"
  },
  "devDependencies": {          //Project development dependencies
    "less": "^4.1.2",
    "sass": "^1.45.0"
  }
}
```

The **name** and **version** are the most important fields and must exist. Otherwise, the current package cannot be installed. The two attributes together form the unique identifier of an npm package.

name indicates the name of a package. The first part of the **name** value, such as **@scope/**, is mandatory in the self-hosted repo and is used as the namespace. Generally, you can search for name to install and use the required package.

```
{
  "name": "@scope/name"
}
```

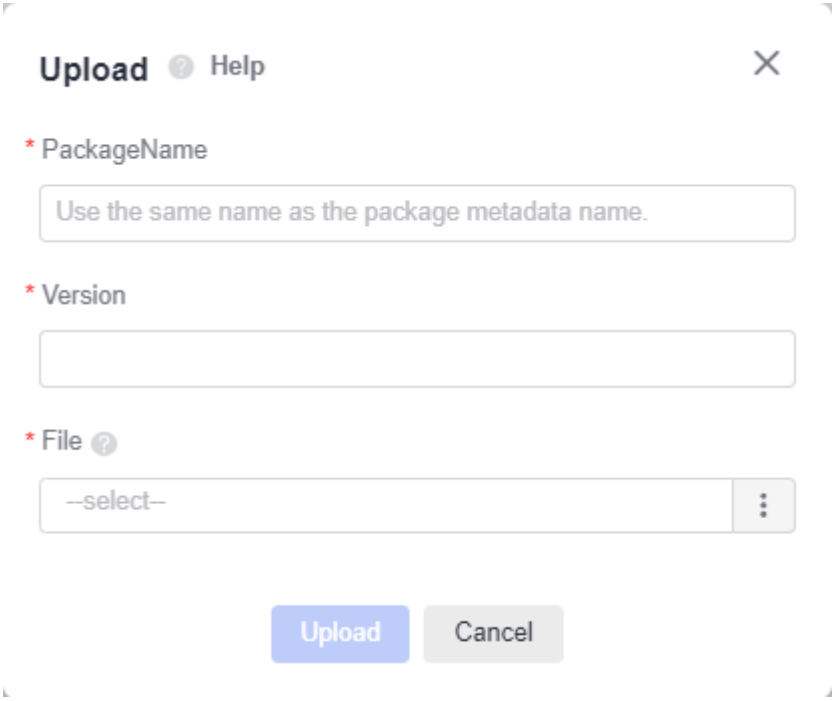
version indicates the version of a package, which is in the x.y.z format.

```
{
  "version": "1.0.0"
}
```

Uploading an npm Component

A self-hosted repo allows you to upload npm component packages in .tgz format. When uploading a package, you need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as the value of **name** in the **package.json** file.
Version	The value must be the same as that of **version** in the **package.json** file.



NOTE

The **PackageName** value of the component to upload must start with the path in the path list added during repository creation. For details, see **Repository Configuration Items**.

Example:

The path **@test** is added during the creation of an npm repository.

When uploading an npm component to the repository, make sure that the value of **PackageName** starts with **@test**. If a path outside the path list is used, for example, **@npm**, the upload will fail.

After the upload is successful, you can view the component package in .tgz format in the repository component list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Component

Go (also called Golang) is a programming language developed by Google. Golang 1.11 and later versions support modular package management tools. A module is a unit for source code exchange and versioning of Go. A **MOD** file is used to identify and manage a module. A **ZIP** file is a source code package. There are two types of Go modules: v2.0 and later versions and v2.0 and earlier versions. The management of the Go module is different between the two versions.

To upload a Go component, upload a **ZIP** file and a **MOD** file. You need to set the following parameters.

Parameter	Description
Zip Path	Complete path of the ZIP file. Valid path formats are: • Versions earlier than v2.0: {moduleName}/@v/{version}.zip • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the file path format is: {moduleName}/vX/@v/vX.X.X.zip – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the file path format is: {moduleName}/@v/vX.X.X+incompatible.zip
Zip File	Directory structure of the ZIP file. Valid directory structure formats are: • Versions earlier than v2.0: {moduleName}@{version} • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the directory structure format is: {moduleName}/vX@{version} – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the directory structure format is: {moduleName}@{version}+incompatible
Mod Path	Complete path of the MOD file. Valid path formats are: • Versions earlier than v2.0: {moduleName}/@v/{version}.mod • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the file path format is: {moduleName}/vX/@v/vX.X.X.mod – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the file path format is: {moduleName}/@v/vX.X.X+incompatible.mod
Mod File	MOD file content. Valid content formats are: • Versions earlier than v2.0: module {moduleName} • Versions later than v2.0: – If the ZIP file contains **go.mod** and the path ends with **/vN**, the content format is: module {moduleName}/vX – If the ZIP file does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**, the content format is: module {moduleName}

Uploading a PyPI Component

You are advised to go to the project directory (which must contain the **setup.py** configuration file) and run the following command to compress the components

to be uploaded into a wheel (.whl) installation package. By default, the installation package is generated in the **dist** directory of the project directory. The Python software package management tool pip supports only wheel installation packages.

```
python setup.py sdist bdist_wheel
```

You need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as the value of **name** in the **setup.py** file.
Version	The value must be the same as the value of **version** in the **setup.py** file.

After the upload is successful, you can view the installation package in **.whl** format in the repository component list. In addition, the corresponding metadata is generated in the **.pypi** directory, which can be used for pip installation.

Uploading an RPM Component

Introduction to RPM

- Red Hat Package Manager (RPM) is proposed by Red Hat and used by many Linux distributions. It is a software management mechanism that installs required software to Linux in database recording mode.
- You are advised to package and name the RPM binary file according to the following rules:

Software name-*Main version number of the software*.*Minor version number of the software*.*Software revision number*-*Number of software compilation times*.*Hardware platform suitable for the software*.**rpm**

For example: **hello-0.17.2-54.x86_64.rpm**.

- **hello**: the software name.
- **0**: the major version number of the software.
- **17**: the minor version number.
- **2**: the revision number.
- **54**: the number of times that the software is compiled.
- **x86_64**: the hardware platform suitable for the software.

Software Name	Major Version	Minor Version	Revision No.	Compilation Times	Applicable Hardware Platform
hello	0	17	2	54	x86_64

Note: You need to set the following parameters when uploading components.

Parameter	Description
Component	Component name
Version	Version of the RPM binary package

Step 1 Access the self-hosted repo homepage. In the left pane, choose the repository to which the private component is to be uploaded.

Step 2 Click **Upload**.

Step 3 Set the component parameters, select the file, and click **Upload**.

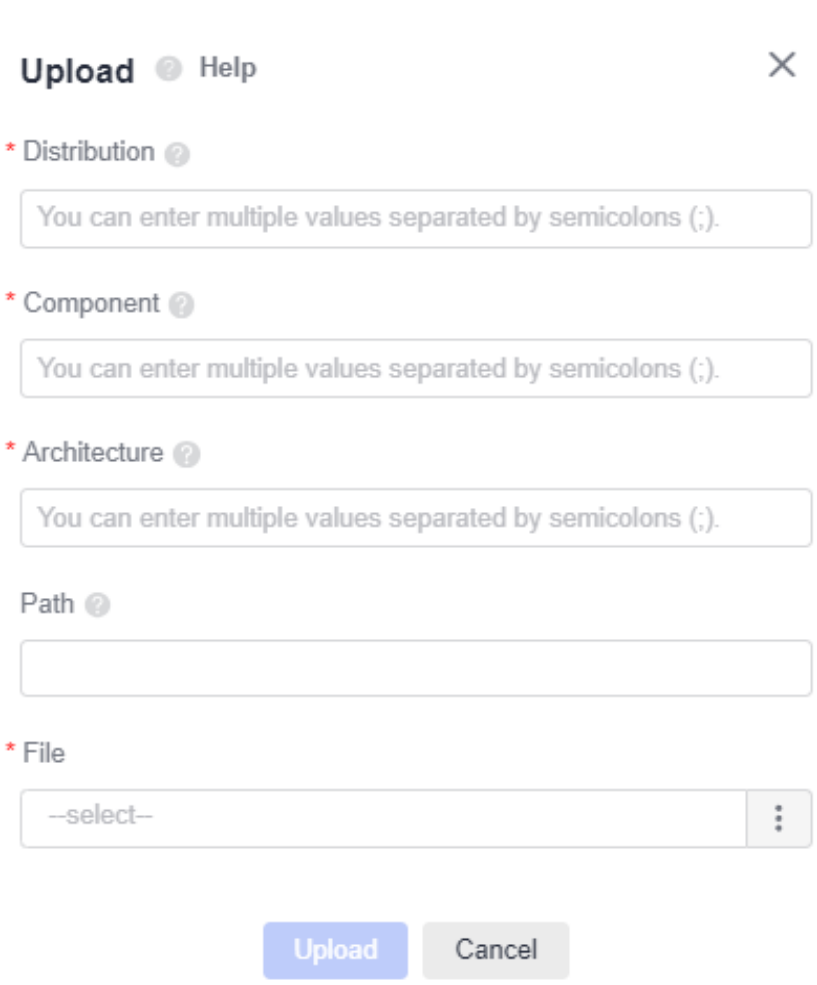
----End

After the upload is successful, you can view the RPM binary package in the repository component list and the corresponding metadata **repodata** directory is generated in the component name directory. You can use Yum to install the component.

Uploading a Debian Component

When uploading a Debian component, you need to set the following parameters:

Parameter	Description
Distribution	Release version of the software package
Component	Name of a software package component
Architecture	Software package architecture
Path	Path for storing the software package. By default, the software package is uploaded to the root path.
File	Local storage path of the software package



After the upload is successful, you can view the installation package in **.deb** format in the repository component list. In addition, the corresponding metadata is generated in the **dists** directory, which can be used for Debian installation.

4.5 Managing Private Components

You can search for, download, delete, follow, and unfollow private components.

Searching for Private Components

Step 1 Go to the **Self-hosted Repos** page and click **Advanced Search** in the upper left corner of the page. The **Advanced Search** page is displayed.

Step 2 You can select the type of component to be searched for in the upper part of the page. By default, all types are selected.

Step 3 Enter the keyword of the file name in the search box, and click to search for the component.

Step 4 Click the **File Name** to view its details.

----End

- Searching for Artifacts by Checksums

1. Click the drop-down list on the left of the search box and select **Checksums** (The default value is the file name).
2. You can also enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click to find the corresponding component.

Downloading a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, locate the private component to be downloaded, and click the component name.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Download**.

----End

Deleting a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, locate the private component to be deleted, and click the component name.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Delete**.

Step 3 In the displayed dialog box, click **Yes**.

----End

Following or Unfollowing a Private Component

Step 1 Access the self-hosted repo homepage. In the left pane, choose the repository to which the private component is to be followed.

If there are too many repositories or components, you can search for the desired one by following instructions in **Searching for Private Components**.

Step 2 Click **Unfollowed** on the right of the page.

When the icon changes to , click **Following** in the lower left corner of the page to view the list of followed components. Click the **path** value in the list to go to the component details page.

----End

4.6 Managing the Recycle Bin

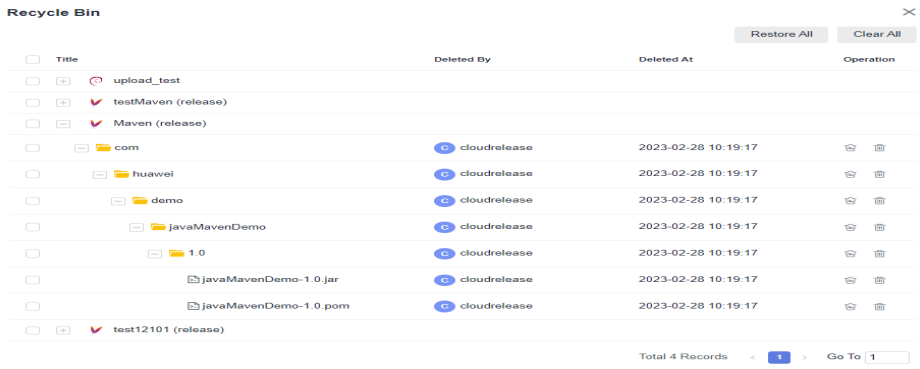
Repositories and components deleted from a self-hosted repo are moved to the recycle bin, where you can manage them.

Step 1 Access the self-hosted repo homepage.

Step 2 Click **Recycle Bin**.

Step 3 Delete or restore the repositories and components in the list as required.

If the icons and are both displayed in the **Operation** column, the repository shown in the corresponding row has been deleted. Otherwise, the repository shown in the corresponding row has not been deleted but the components in it have been deleted. You can click the repository name to view the deleted components in the repository.



Available operations are as follows:

Operation Type	Operation	Description
Restore	Restore a repository	Click in the **Operation** column to restore the repository.
	Restore a component	Go to the repository where the component to restore is located, and click in the **Operation** column to restore the component.
	Batch restore components	Go to the repository where the components to restore are located, select the components, and click **Restore** below the list to restore the components.
	Restore all	Click **Restore All** in the upper right corner of the page to restore all repositories and components in the recycle bin.
Delete	Delete a repository	Click in the **Operation** column to delete a repository.

Operation Type	Operation	Description
	Delete a component	Go to the repository where the component to delete is located, and click  in the **Operation** column to delete the component.
	Delete components in batches	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to delete the components.
	Clear recycle bin	Click **Clear All** to delete all repositories and components in the recycle bin.

NOTICE

If you choose to delete a repository or component in the recycle bin, it cannot be retrieved. Exercise caution when performing this operation.

----End