CodeArts Artifact

User Guide

Issue 01

Date 2024-10-15

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property 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 Cloud 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.

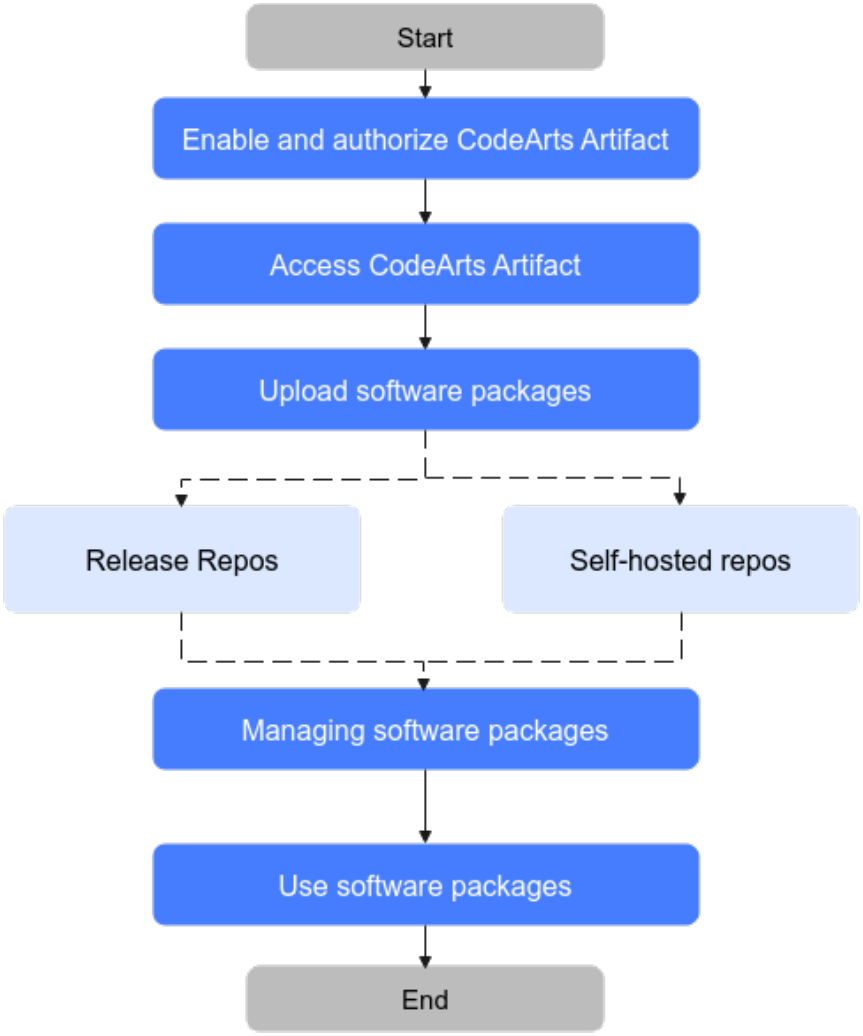
Contents

1 CodeArts Artifact User Guide

CodeArts Artifact assists developers in managing dependencies across different programming languages during development and builds. It stores binary artifacts and centralizes key delivery components. It supports popular package types like Maven and npm. It can seamlessly interconnect with local build tools and on-cloud CI/CD so that you can manage software package lifecycle to improve release quality and efficiency. It also provides features such as artifact package version control, granular permission management, security scanning, and more.

CodeArts Artifact is a service provided within the **CodeArts** solution. For details about its role in the solution, see **CodeArts Architecture**.

Basic operation process of CodeArts Artifact



2 Release Repos 2.0

2.1 Overview 2.0

CodeArts Artifact is a general 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.

NOTE

CodeArts Artifact was upgraded in March 2023. Inventory Release Repos and resources belonging to users who registered before this update are stored in the old Release Repos.

You do not need to create Release Repos. After you create a project, Release Repos with the same project name is automatically generated in the new Release Repos.

Accessing Release Repos 2.0

Step 1 Subscribe to CodeArts Artifact by referring to **Purchasing a CodeArts Package**.

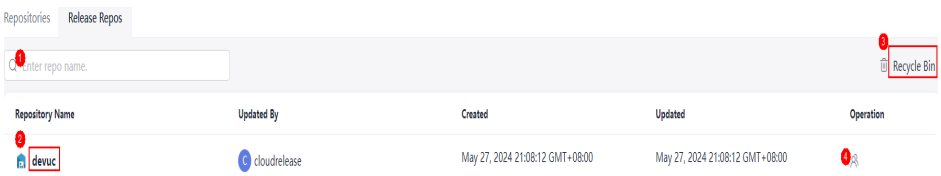
Step 2 Add members and assign roles to them. For details, see **Configuring Permissions 2.0**.

Step 3 **Log in to the Huawei Cloud console**.

Step 4 Click ☰ in the upper left corner of the page and choose **Developer Services** > **CodeArts Artifact** from the service list.

Step 5 Click **Access Service**. The homepage of CodeArts Artifact is displayed.

Step 6 Click the **Release Repos** tab and view the project name list of the current tenant. You can perform the following operations as required:



No.	Operation	Description
1	Search for repositories	Enter the project name in the search box to find the Release Repos of the project.
2	View folder details	Click any folder to view the list of archived software packages in folders. You can upload, download, and edit software packages or folders.
3	Manage recycle bin	Click **Recycle Bin**. You can delete or restore software packages or folders as required.
4	Set project permissions	Click ··· to go to the **Set Project Permissions** page and edit member permissions. For details, see **Configuring Permissions 2.0**.

NOTE

On the **Release Repos** homepage, 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 it first.

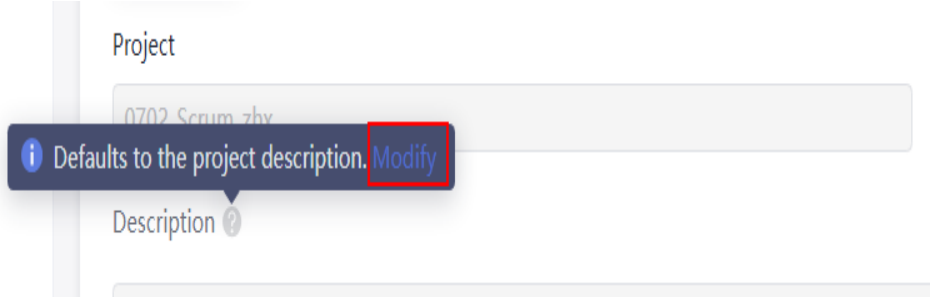
----End

Checking Basic Information

Step 1 On the Release Repos page, click **Settings** in the upper right corner of the page.

Step 2 The repository name, package type, and description are displayed.

- The repository name is the same as that of the project and cannot be modified.
- The description is synchronized with that of the project, as shown in the following figure. Click **Modify** to go to the basic project information page and modify the description.



----End

2.2 Configuring Permissions 2.0

A new member must be assigned a specified role to use CodeArts Artifact. In Release Repos, different roles have different permissions. Members with permission to set permissions can edit the permissions.

Step 1 Add members and assign roles to them by referring to **Adding Members** and **Assigning Roles**.

Step 2 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

Step 3 Click ··· in the upper left corner and choose **Set Project Permissions** from the drop-down list.

Step 4 Click the role for which you want to set permissions, choose **CodeArts Artifact**, click **Edit** to select permissions as required, and click **Save**.

The table below lists the default permission matrix provided by Release Repos.

Table 2-1 Project-level permissions

Description									
Role/ Operation	Change package status	Upload	Delete/ Restore (test package)	Delete/ Restore (production package)	Edit (test package)	Create folder	Download	Restore all	Clear all
Project manager	√	√	√	×	√	√	√	√	√
Product manager	×	×	√	×	√	√	√	×	×
Test manager	×	√	√	×	√	√	√	√	√
System engineer	×	√	√	×	√	√	√	×	×
Committer	×	√	√	×	√	√	√	×	×
Developer	×	√	√	×	√	√	√	×	×
Tester	×	√	×	×	×	√	√	×	×

Description									
Participant	×	×	×	×	×	×	√	×	×
Viewer	×	×	×	×	×	×	×	×	×
project administrator	√	√	√	√	√	√	√	√	√

NOTE

- By default, project administrators have all permissions and their permission scope cannot be modified.
- Custom roles do not have preset permissions. You can contact the administrator to grant permissions for the resources needed for your role.
- By default, the project administrator, project manager, and test manager can assign permissions. If other roles can assign permissions, they can continue to manage permissions for other roles in Release Repos.

----End

2.3 Uploading Software Packages 2.0

Software packages are intermediate products generated during compilation and build in software development. They are an indispensable part of continuous integration and continuous delivery. By uploading packages to Release Repos for storage and management, you can secure file storage, effectively support software development activities, provide reliable software package for deployment, and provide dependencies for build tasks.

Uploading Packages

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

After selecting a folder, click **Upload** to manually upload local software packages to the target folder.

Step 2 In the displayed dialog box, enter the required information and click **Upload**.

- **Target Repository**: current Release Repos.
- **Version**: set the version number for software packages.
- **Upload Mode**: select **Single file** or **Multiple files**. **Single file** is selected by default here.
- **Path**: After you set the path name, a folder with that name is created in the **Repo View**, where the uploaded software packages will be stored.
- **File**: select software packages from your local PC to upload.

Step 3 In the **Repo View**, click the name of the uploaded software package to view its details.

Step 4 Click  next to the package to change its name.

Click  next to the package to permanently delete it 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 Repos.

----End

Related Tasks

CodeArts Artifact allows you to upload software packages either from the page or through CodeArts Build to Release Repos. For details, see **Uploading Software Packages to Release Repos**.

2.4 Managing Software Packages 2.0

Viewing Packages in Repo View

On the **Repo View** page, you can view or edit software package details, including general information, build metadata, and build packages.

Access Release Repos, select **Repo View**, and click a software package name to see its details page. The software package details are displayed on three tab pages: **General**, **Build Metadata**, , and **Build Packages**.

- **General**: displays the repository name, relative path, repository path, version, creator, creation time, modifier, modified time, size, and checksum.

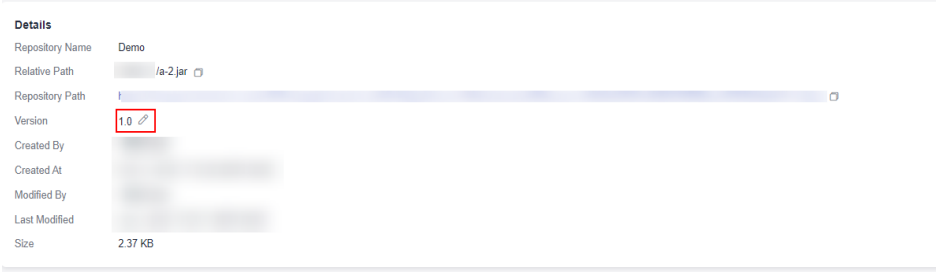
  Click  to change the release version of the software package. (The release version archived by CodeArts Build is the build sequence number by default.)
- **Build Metadata**: displays the build task, size, build number, builder, code repository, and code branch of the software package. Click **Build Task** to link to the task in CodeArts Build.
- **Build Packages**: displays records of software packages archived from build tasks. Click  to download the package.

Viewing Packages in Version View

CodeArts Artifact displays software packages by version. In **Version View** tab page, you can filter and display artifact packages by name and version number, and sort files by their update time.

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

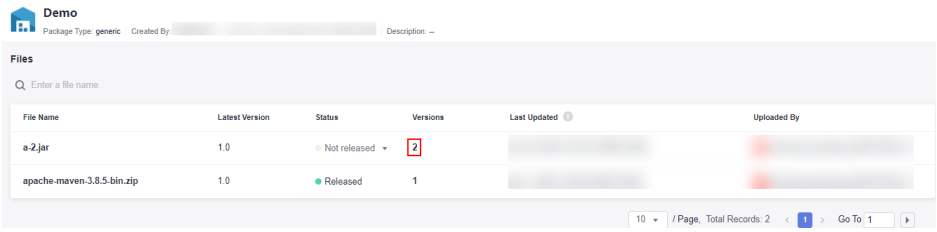
Step 2 You need to set the version of the uploaded package. By default, CodeArts Build archives the release version as the version number set during the build task.



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

Step 4 Release Repos stores software packages of different versions with the same name in the same file. Click a file name in the **File Name** column. The latest version of the software package is displayed.

Step 5 Click a number in the **Versions** column. The version list of the software package is displayed.



Click a number in the **Version No.** column. The **Overview** and **Files** pages of the software package are displayed. In the **Files** tab, click a name in the **File Name** column. The path of the software package is displayed.

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

- In **Files**, set the status to **Released**. This will update the software package to the **Released** status for the latest version.
- Click a version in the **Versions** column to go to the 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 (including file names or version numbers), but can only be downloaded or deleted.

----End

Setting Status of a Folder

Step 1 After entering a level-1 folder, 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.

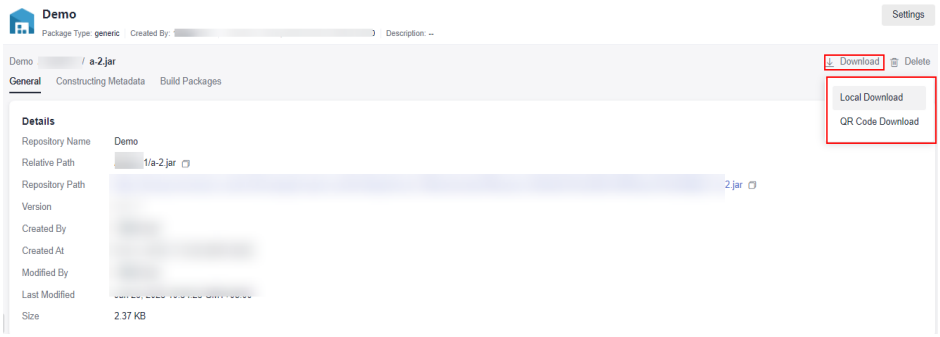
----End

Downloading Software Packages

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

Step 2 In the **Repo View** tab, select the software package to be downloaded. You can download it in either of the following ways:

- Method 1: Click **Download** on the right of the page and select a download mode from the drop-down list.



  - **Local Download**: Download the package to your local PC.
  - **QR Code Download**: Use your mobile phone to scan the QR code to download the package.
- Method 2: Hover your cursor over the software package to be downloaded and click  on the right of the package.

----End

Searching for Software Packages

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

Step 2 Enter a keyword (a folder or file name) in the search box above the list to search for the software package whose name contains the keyword.

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

----End

2.5 Clearing Policies 2.0

If files in Release Repos become redundant or invalid, you can use the scheduled auto-clearing policy to either move expired files to the recycle bin or delete outdated files from it. This helps you manage your Release Repos more efficiently and in an organized manner.

Setting Clearing Policies

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

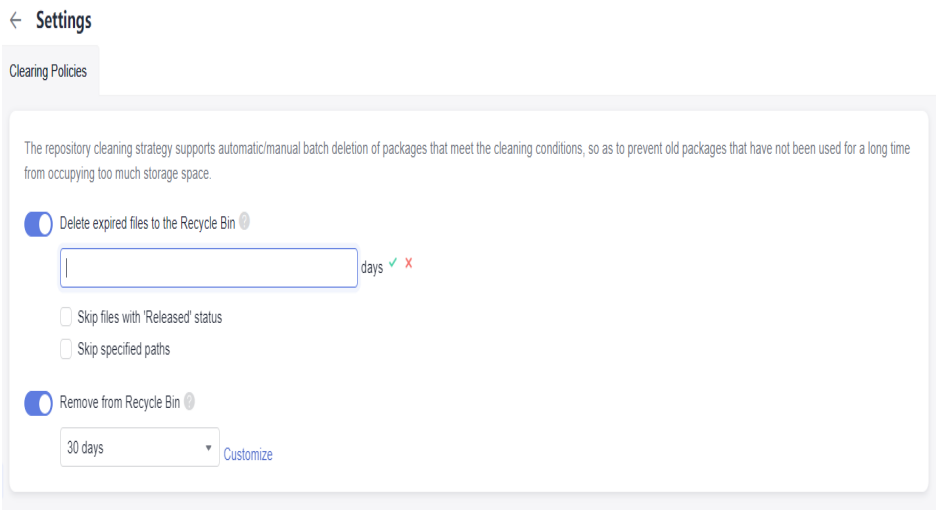
Step 2 Click **Settings** in the upper right corner of the page and select **Clearing Policies**.

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.

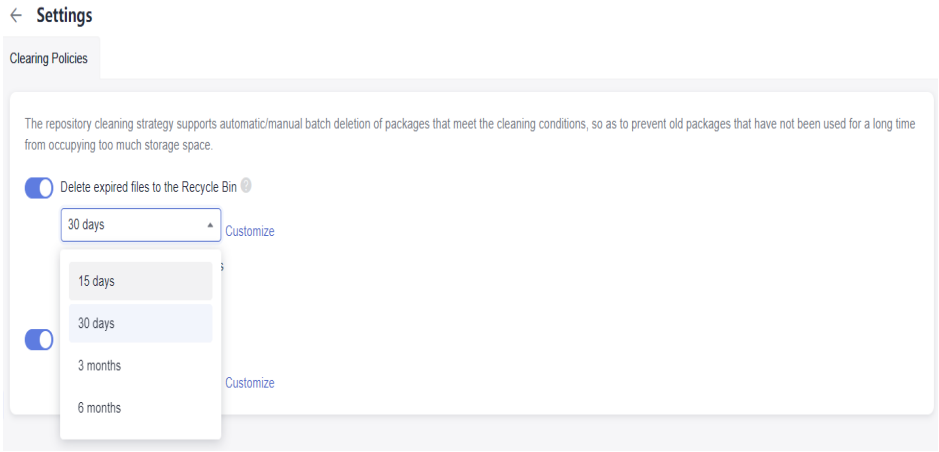
Default retention periods:

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

Figure 2-1 Setting clearing policies



- You can also set a period. Click **Customize**, enter a number, and click **√** to save the setting.



NOTE

Parameters below are optional.

- **Skip released files**: The system retains files in the production package state when clearing files. For details, see **Setting Production Package Status**.
- **Skip specified paths**: The system retains software packages that match the file paths set by users when clearing files. You can set multiple file paths, each starting with a slash (/) and separated by semicolons (;).

----End

2.6 Recycle Bin 2.0

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

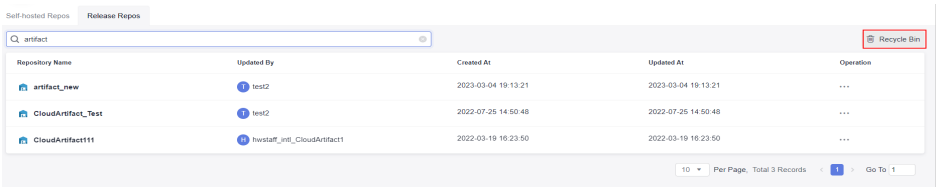
CodeArts Artifact provides both a **global recycle bin** and **project recycle bins**.

Global Recycle Bin

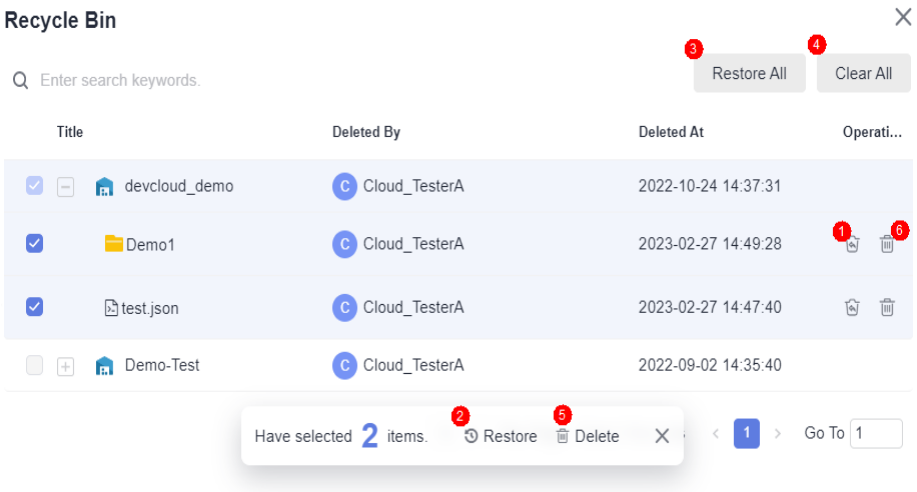
In the global recycle bin, you can manage software packages and folders deleted from any project.

Step 1 **Log in to CodeArts Artifact homepage**.

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



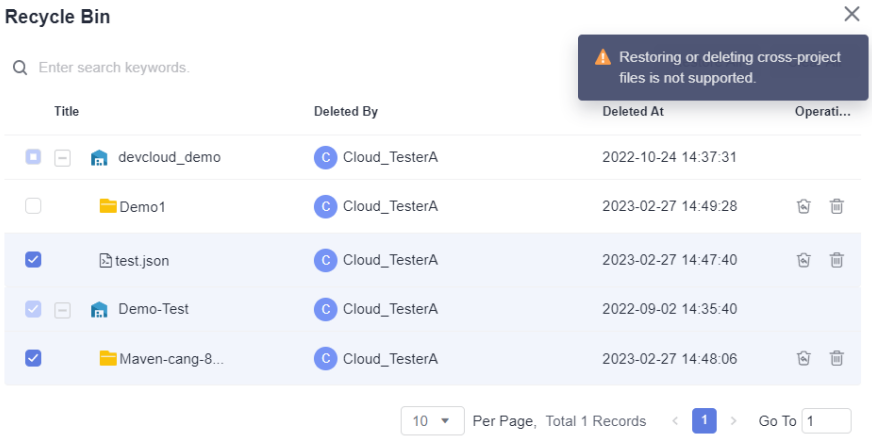
Step 3 The deleted files from different projects are displayed on this page. Perform the following operations on the software package or folder as required:



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

NOTICE

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



----End

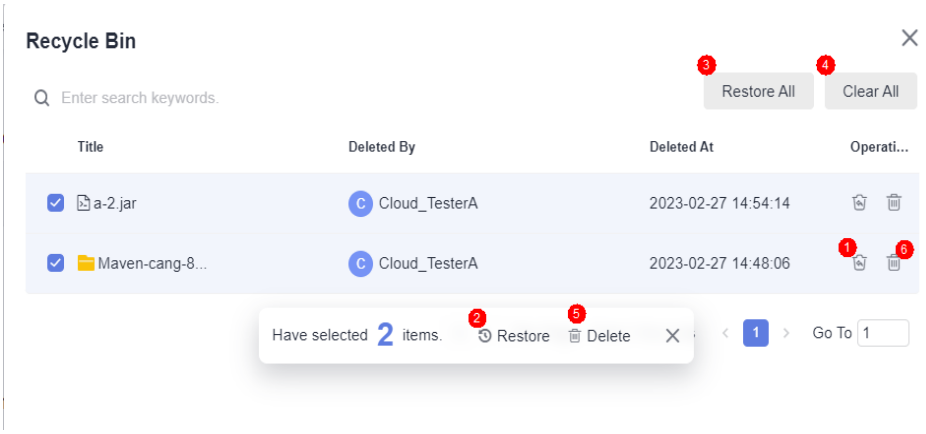
Project Recycle Bin

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

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

Step 2 Click **Recycle Bin** in the lower left corner of the page.

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



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

NOTICE

Once you delete a software package or folder from the recycle bin, it cannot be recovered. Exercise caution when performing this operation.

----End

3 Self-Hosted Repos 2.0

3.1 Overview 2.0

Developers often need to share components with their team during development. Self-hosted repos serve as a central place where these components can be stored and accessed by others, making it easy for team members to obtain components from repositories.

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

NOTE

In March 2023, CodeArts Artifact was upgraded. Before this upgrade, existing self-hosted repos were not associated to projects. Consequently, repos and their resources from before the upgrade remain in the old repos.

Accessing Self-hosted Repos

Step 1 Subscribe to CodeArts Artifact by referring to **Purchasing a CodeArts Package**.

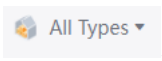
Step 2 Add members and assign roles to them. For details, see **Configuring Permissions 2.0**.

Step 3 **Log in to the Huawei Cloud console**.

Step 4 Click  in the upper left corner of the page and choose **Developer Services** > **CodeArts Artifact** from the service list.

Step 5 Click **Access Service**. The homepage of CodeArts Artifact is displayed.

Step 6 Click the **Repositories** tab. All self-hosted repos created are displayed.

Click 

. In the drop-down list box, you can view self-hosted repos by type.

Step 7 Click a repository name to go to the self-hosted repo page for the project.

----End

3.2 Creating a Self-Hosted Repo

If you use this service for the first time, you need to create a repository. Self-hosted repos are divided into local repositories and virtual repositories.

Local Repository: An actual physical repository hosted on the server, where you can upload different types of artifacts.

Virtual Repository: Users can configure agent sources in the virtual repository to connect with local third-party repositories. It also offers local repository functions and provides a single entry to make setup easier for customers.

Step 1 Click a project card to access the project. (If no project is available, **create one**.)

Step 2 Choose **Artifact Self-hosted Repos** from the menu bar. Click + Create in the upper left corner of the page.

Step 3 The **Create Repository** page is displayed.

Step 4 Configure basic information and click **Submit** by referring to **Table 3-1**.

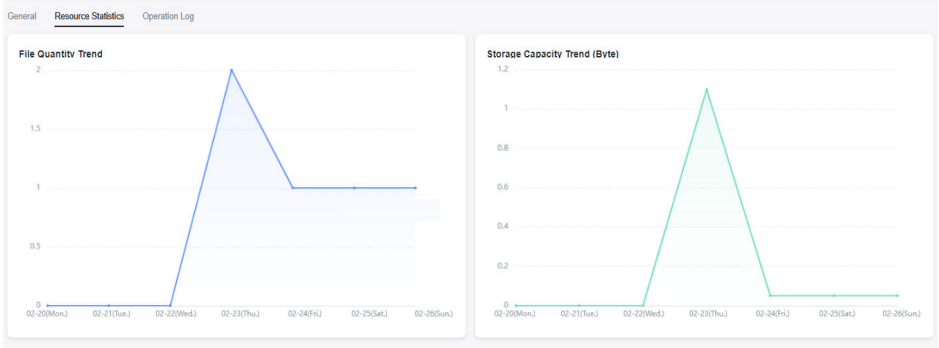
Table 3-1 Parameters for configuring repositories

Item	Mandatory	Description
Repository Type	Yes	You can choose **Local Repository** or **Virtual Repository**. • A local repository is hosted on the server and is a physical repository that stores artifact data. • A virtual repository combines local and proxy repositories, offering a single, unified entry for accessing artifacts.
Repository Name	Yes	Enter up to 20 characters: letters, numbers, underscores (_), hyphens (-), and periods (.). **NOTE** After a repository is created, its name cannot be changed.
Package Type	Yes	Local repository supports Maven, npm, Go, PyPI, RPM, Debian, Conan, Docker, CocoaPods, and NuGet artifacts. Virtual repository supports Maven, npm, Docker, and PyPI artifacts. Complete the configuration for the selected format by referring to **Configuring a Repository**. **NOTE** Docker and CocoaPods repositories currently are only available in the AP-Singapore region.

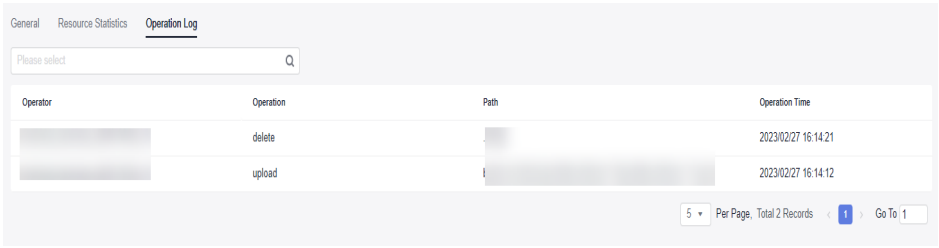
Item	Mandatory	Description
Project	Yes	Choose a project to associate with the newly created repository. After the settings are complete, the project to which the user belongs cannot be changed.
Description	No	Enter up to 200 characters.

Step 5 View the list on the left side of the page to find the new repository. Click the repository name to view more details. The repository details are displayed on the **General**, **Resources**, and **Operation Logs** tabs.

- **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 a Repository

The following table describes the configuration items specific to each type of repository. For details, see **Table 3-2**.

Table 3-2 Configuration items

Package Type	Item	Mandatory	Description
Maven	Version Policy	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 path to be added and click **+**. During builds, only Maven files whose paths start with the preceding paths can be uploaded to the self-hosted repo.
npm	Include patterns	No	Enter the path to be added and click **+**. During builds, only npm files whose paths start with the preceding paths can be uploaded to the self-hosted repo.
Go	Include patterns	No	Enter the path to be added and click **+**. During builds, only Go files whose paths start with the preceding paths can be uploaded to the self-hosted repo.
PyPI	Include patterns	No	Enter the path to be added and click **+**. During builds, ensure that PyPI dependency packages in which the **name** value in the **setup.py** file matches the preceding paths.
RPM	Include patterns	No	Enter the path to be added and click **+**. During builds, only RPM binary files whose paths start with the preceding paths can be uploaded to the self-hosted repo.
Conan	Include patterns	No	Enter the path to be added and click **+**. Only Conan files whose paths start with the preceding paths can be uploaded from a local client to the self-hosted repo.
Docker	Include patterns	No	Enter the path to be added and click **+**. Only image files whose paths start with the preceding paths can be pushed to the self-hosted repo.
CocoaPods	Include patterns	No	Enter the path to be added and click **+**. During builds, only CocoaPods files whose paths start with the preceding paths can be uploaded to the self-hosted repo.

Setting Proxy for a Virtual Repository

The new support for custom proxy repositories enables users to create proxies for both open-source and third-party dependency repositories. After files are downloaded from proxy repos, they can be cached to CodeArts Artifact, allowing users to download files from third-party dependencies as fast as from the local repos.

NOTE

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

In the self-hosted repo, you can add custom mirror sources to the Maven, npm, Docker, 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, choose **Mirror** > **Mirror**.

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

Step 4 In the displayed dialog box, choose the package type, and enter the proxy name (required), proxy address (required), PyPI index proxy address (required only if the package type is PyPI), proxy username, and proxy password.

NOTE

- The proxy address must start with **https://** or **http://**. Otherwise, an error message is displayed, indicating that the URL is invalid.
- 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.

Operation	Description
Edit	Click in the **Operation** column to change the proxy 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 from the **Proxy Settings** page of that repository and return to this page to delete the proxy source.

----End

Adding Proxies for a Virtual Repository

Step 1 A virtual repo has been created. For details, see **Creating Repositories**.

Step 2 Go to the self-hosted repo page. In the left pane, click the name of 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 **Proxy Name** drop-down list and select a custom proxy source. For details about how to create a custom proxy source, see **Custom Proxy 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 from the **Proxy Name** drop-down list.

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

- Click  in the **Operation** column to change the proxy 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 the proxy.

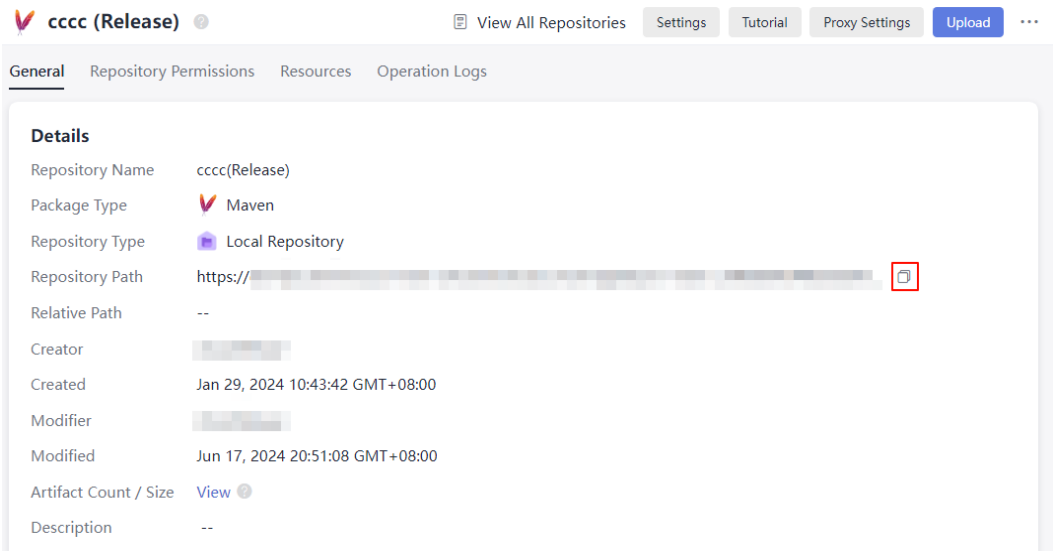
----End

Obtaining Repository Paths

Each time you create a repository, a repository path is generated for it. You will use this path to connect the repository to your local development environment. Perform the following operations to obtain the path.

Step 1 Go to the self-hosted repo page. In the left pane, click the name of the target repository.

Step 2 The repository path is displayed on the **General** page. Click  to obtain the path.



----End

Deleting a Repository

You can delete repos, and they will be moved to the recycle bin.

Step 1 Go to the self-hosted repo page. In the left pane, click the name of the target repository.

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

Step 3 Click **Delete** on the right of the page. The deleted repo no longer displays in the repo list on the left pane.

----End

3.3 Configuring Repository Permissions 2.0

Managing Repository Permissions

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

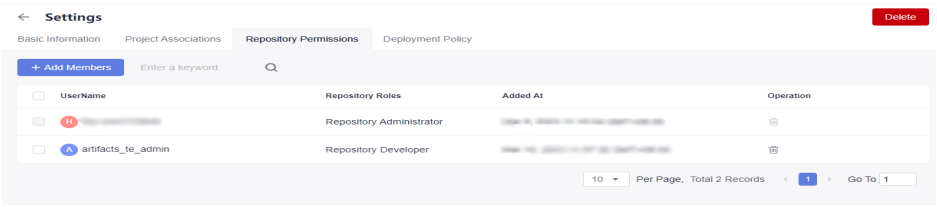
- The project creator and project manager are repository administrators.
- The developer, test manager, tester, and operation manager are repository developers.
- The participant, viewer, and custom roles are repository viewers.

To add or remove 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.

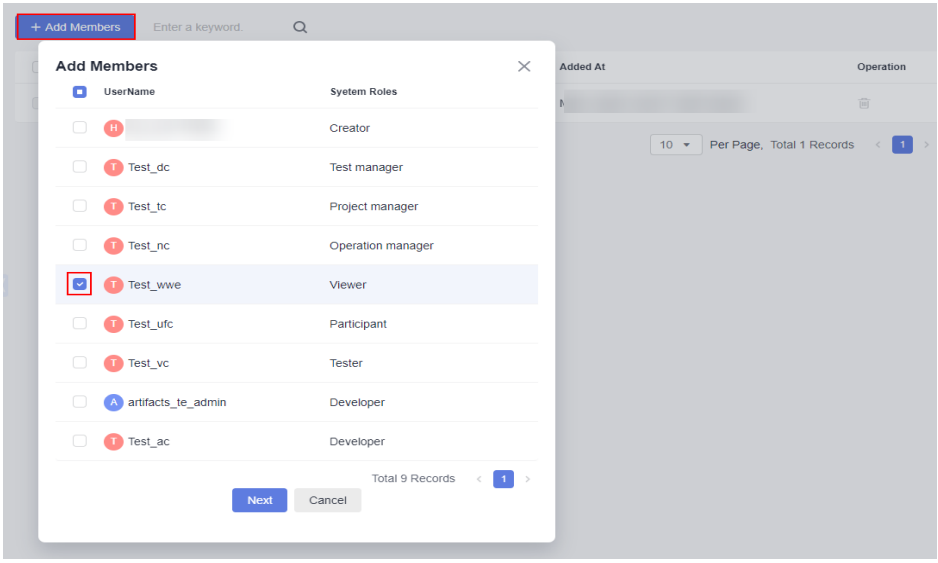
Step 2 Click **Settings** on the right of the page.

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



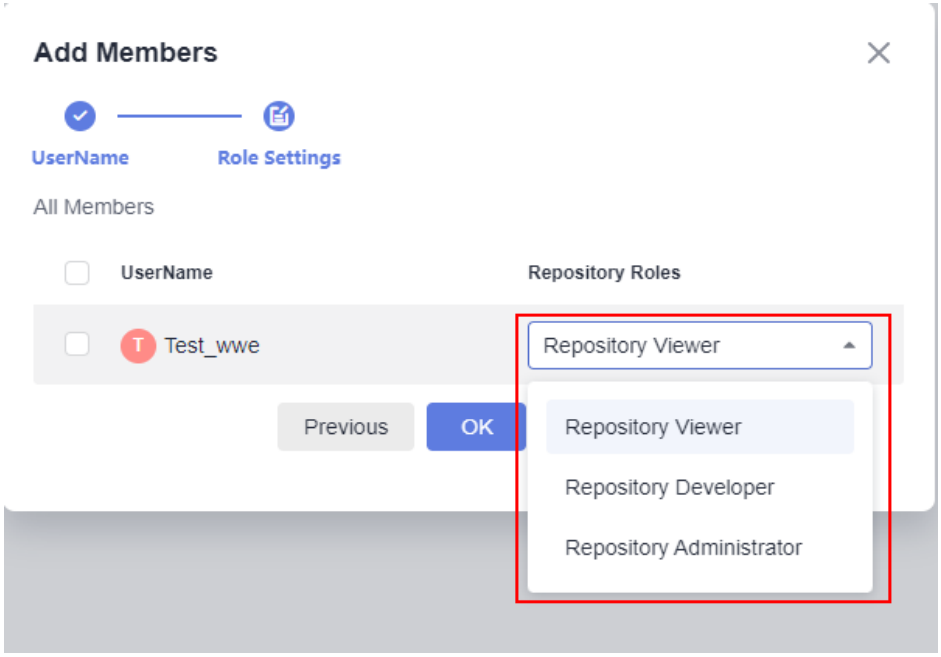
Step 4 Add members.

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



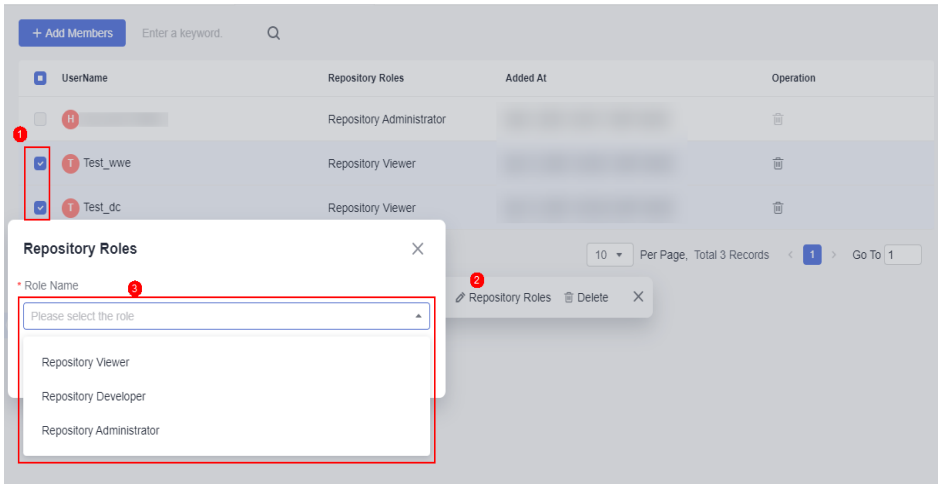
Step 5 Assign roles to members.

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



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

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



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 repository	√	√	√	×	×	×

Edit a repository	√	√	√	×	×	×
Manage the association between repositories and projects	√	√	√	×	×	×
Upload a component	√	√	×	√	√	×
Download a component	√	√	√	√	√	√
Delete components	√	√	×	√	√	×
Restore components	√	√	×	√	√	×
Permanently delete a component	√	√	×	√	√	×
Delete a repository	√	×	×	×	×	×
Restore a repository	√	√	×	√	√	×
Permanently delete a repository	√	×	×	×	×	×
Clear all	√	√	√	×	×	×
Restore all	√	√	√	×	×	×

Manage user permissions	√	√	√	√	×	×

----End

3.4 Managing a Self-Hosted Repo 2.0

3.4.1 Checking Basic Information and Adding Path Patterns

Step 1 Click a project card to access the project. (If no project is available, **create one**.)

Step 2 Choose **Artifact** > **Self-hosted Repos** from the navigation pane.

Step 3 In the left pane, click the name of the target repository to be edited.

Step 4 Click **Settings** on the right of the page. The **Basic Information** tab page is displayed by default.

Step 5 The repository name, package type, project, and version policy cannot be edited. But you can edit the description and include and exclude patterns as required.

On the **Basic Information** page, enter the path and click + to add it for Maven, npm, Go, PyPI, RPM, Docker, CocoaPods, and Conan repositories.

Click 🗑 to delete paths.

Step 6 Click **Submit** to save the settings.

----End

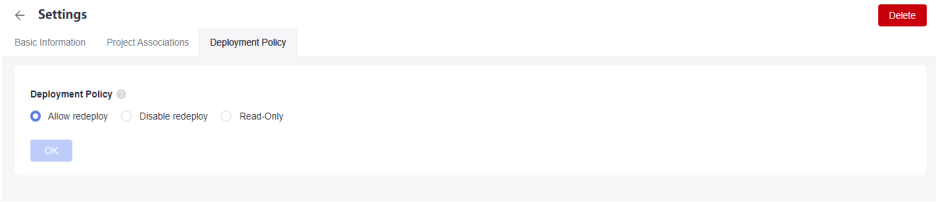
3.4.2 Configuring Deployment Policies

You can set policies to control how artifacts are uploaded to your repository, such as whether new artifacts can redeploy existing ones in the same path.

Step 1 Go to the self-hosted repo page. In the left pane, click the name of the target repository.

Step 2 Click **Settings** on the right of the page and click the **Deployment Policies** tab.

Figure 3-1 Configuring deployment policies



- **Allow redeploy** (enabled by default): Artifacts in the same path can be uploaded, replacing the original package.

- **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 Once you complete the settings, they are automatically saved by the system.

----End

3.4.3 Configuring Clearing Policy for Maven Repository

The clearing policy allows you to automatically or manually clear artifacts in batches. This helps optimizes storage, keeps artifacts organized, and ensures they are transferred correctly during development, testing, deployment, and release.

Background

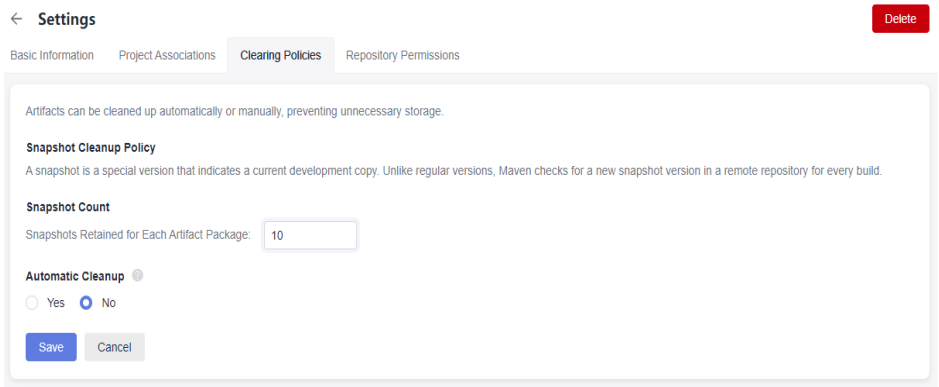
A Maven snapshot is a special version that reflects the current state of ongoing development and differs from regular versions. Maven checks for new snapshots in the remote repository with each build and allows you to set limits on how many snapshot versions to retain, as well as automatically clear out expired snapshots.

Snapshot Cleanup Policy

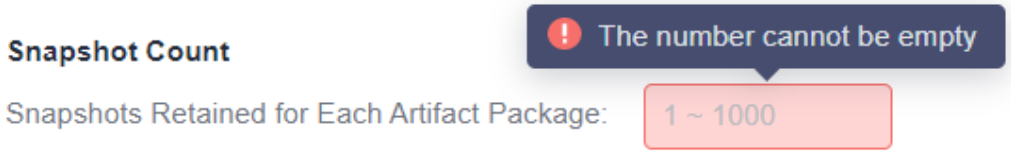
Step 1 Click a project card to access the project and choose **Artifact** > **Self-hosted Repos** from the navigation pane.

Step 2 Select a Maven repository (**Snapshot**) from the 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 1,000.



When the number of retained snapshots exceeds the set limit, the oldest snapshot is replaced by the latest version.

Step 5 Enable automatic cleanup (**No** by default). Click **Yes** and enter the number of days. Snapshots older than the specified number of days will be automatically cleaned up.

The automatic cleanup time must be set between 1 and 100 days.

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

----End

3.4.4 Associating Maven Repository with Projects

After the Maven repository is associated with multiple projects, you can select this Maven repository in the build step of the build task in the project to store the build product to the repository.

Procedure

Step 1 Go to the self-hosted repo page. In the left pane, click the name of a Maven repository.

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

Step 3 Find the target project to be associated in the list and click in the **Operation** column of the corresponding row.

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

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

----End

Related Tasks

In CodeArts Build, you need to upload build products to self-hosted repos. For details, see **Using the File From the Self-hosted Repo to Build with Maven and Uploading the Resulting Software Package (Built-in Executors, GUI)**.

3.5 Uploading/Downloading Components on the Self-Hosted Repo Page

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

Uploading Components

Step 1 Click a project card to access the project. (If no project is available, **create one**.)

Step 2 Choose **Artifact** > **Self-hosted Repos** from the navigation pane.

Step 3 In the left pane, click the target repository to which the private component is to be uploaded.

Step 4 Click **Upload** on the right of the page.

Step 5 In the **Upload** dialog box, specify parameters, select the file, and click **Upload**. Detailed configuration for each component type is described below.

NOTE

- The maximum file size for uploads is 100 MB for Maven, npm, PyPI, RPM, and Debian types, and 20 MB for NuGet.
- You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to self-hosted repos.

----End

Maven Components

- 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 run, Maven searches for the POM in the current directory, reads its content, obtains the required configuration information, and builds 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 component package. It stands for **Group ID**, **Artifact ID**, and **Version**. **Group ID** indicates a company or organization. For example, Maven core components are in the **org.apache.maven** organization. **Artifact ID** indicates the name of the component package. **Version** indicates the version of the component package.
- Maven dependencies are crucial for POM files. The building and running of most projects depend on the dependency on other components. Add the dependency list to your 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

POM and GAV upload modes are supported.

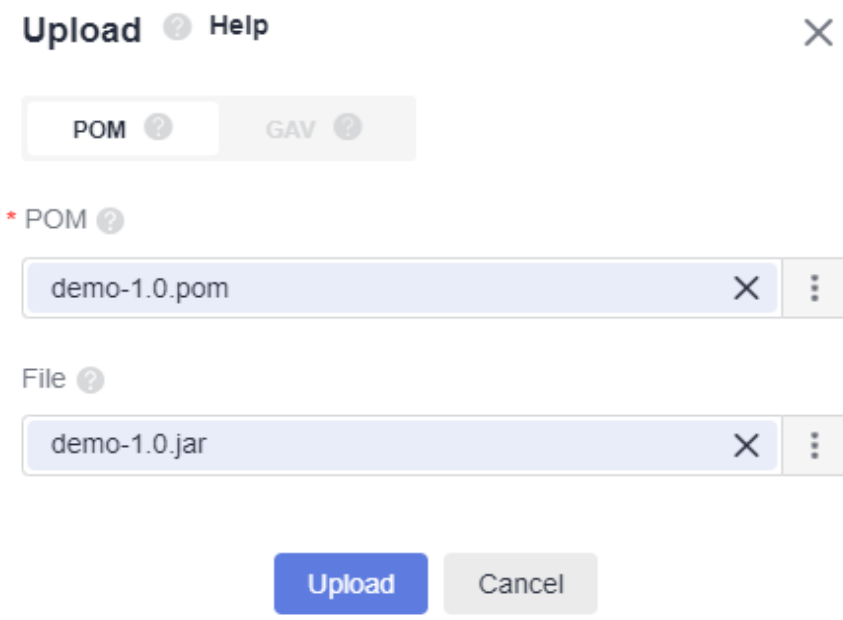
Table 3-3 Upload modes

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 either upload just the POM file or both the POM file and its related components. The uploaded file must match the **artifactId** and **version** values specified in the POM. As shown in the following figure, if the **artifactId** value is **demo** and the **version** value is **1.0** in the POM, then the uploaded file must be named **demo-1.0.jar**.

  Figure 3-2 Uploading a component in POM mode



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** values in **POM** must match the file name in **File**. For example, if the **artifactId** value is **demo** and the **version** value is **1.0** in **POM**, then the uploaded file must be named **demo-1.0** in **File**.

- GAV

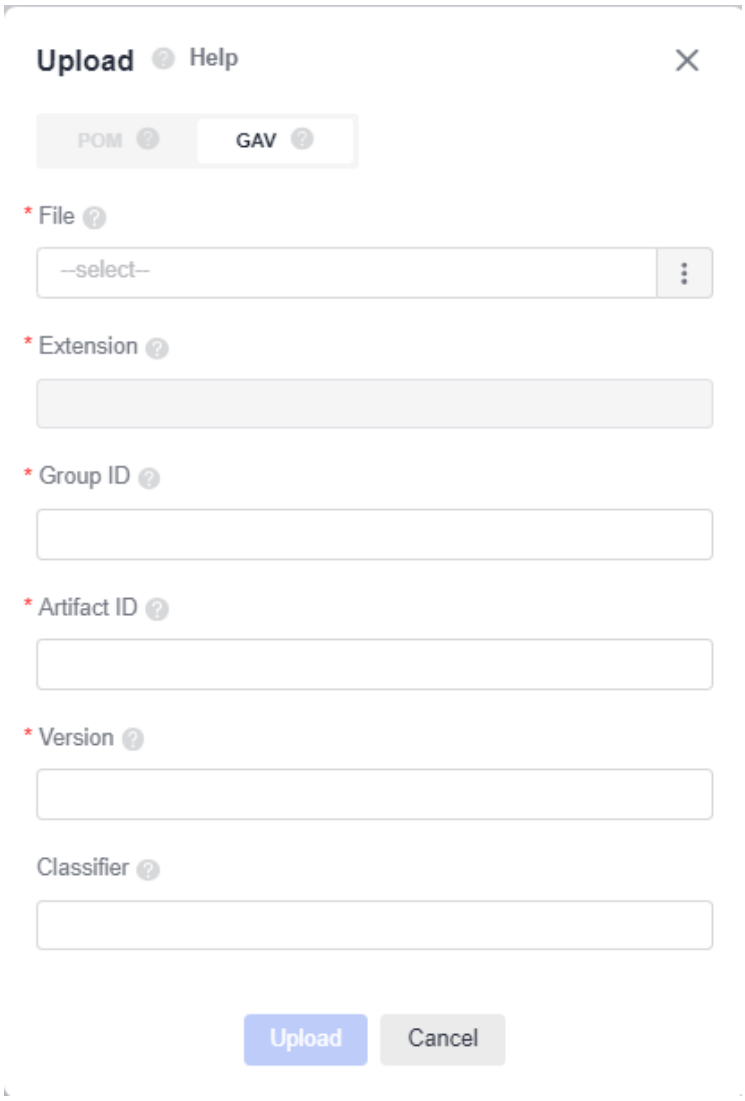
  In the GAV mode, the **Group ID**, **Artifact ID**, and **Version** parameters must be manually specified and they determine the name of the file to upload. **Extension** indicates the packaging type and determines the file type to be uploaded.

  Classifier is used to differentiate artifacts that are built from the same POM but contain different contents. This field is optional. It can contain letters, digits, underscores (_), hyphens (-), and dots (.). If you specify this field, it will be appended to the file name.

  Common Usage Scenario

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

Figure 3-3 Uploading a component in GAV mode



npm Components

Node Package Manager (npm) is a JavaScript package management tool. An npm component package is the item managed by npm, and the npm repository is used to store and manage these packages.

The npm component package consists of a 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 command
    "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 the package. The first part of the name, such as **@scope**, is used as the namespace. The other part is **name**. Generally, you can search with the **name** field to install and use the required package.

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

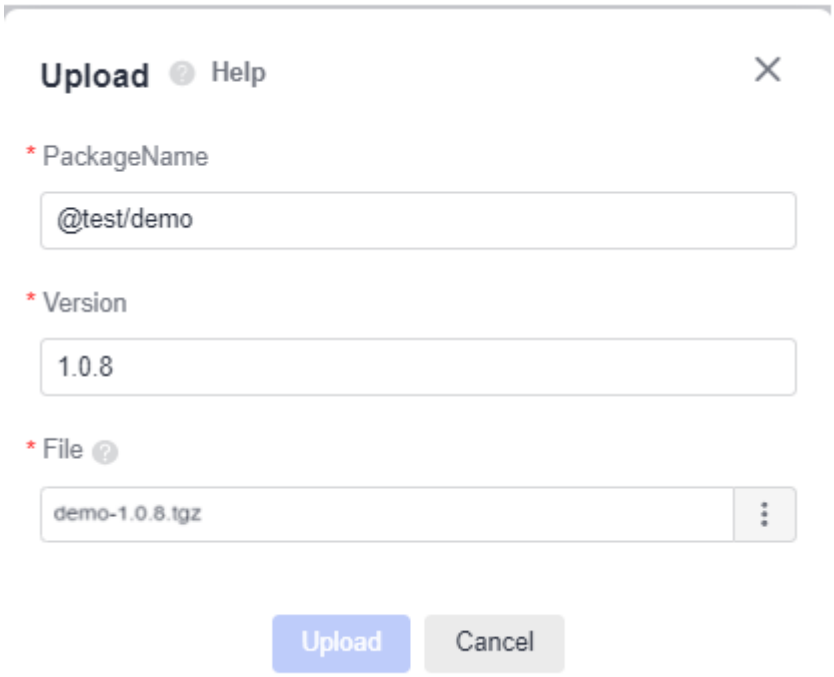
version indicates the version of the package, which is in the x.y.z format.

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

Uploading an npm Component

npm component packages in .tgz format can be uploaded to self-hosted repos. When uploading packages, you need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as that 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 **PackageName** starts with a path in the path list added during repository creation. For details, see **Configuring a Repository** in the help guide.

Example:

The path **@test** is added when you create an npm repository.

When uploading components to the repository, make sure that the **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 find the component in .tgz format in the repository list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Component

Go, also known as Golang, is a programming language developed by Google. Golang 1.11 and later support modular package management. A module is a unit for source code exchange and versioning in Go. A MOD file identifies and manages a module. A ZIP file is a source code package. There are two types of Go modules: v2.0 and later and v2.0 and earlier. The management of the Go module is different between these two versions.

To upload a Go component, you need to upload both a ZIP file and a MOD file and 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, this package is generated in the **dist** directory of your project directory. Note that the Python 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 that of **name** in the **setup.py** file.
Version	The value must be the same as that of **version** in the **setup.py** file.

After the upload is successful, you can find the installation package in **.whl** format in the repository 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), developed by Red Hat, is used by many Linux distributions. It is a software management system that installs software on 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

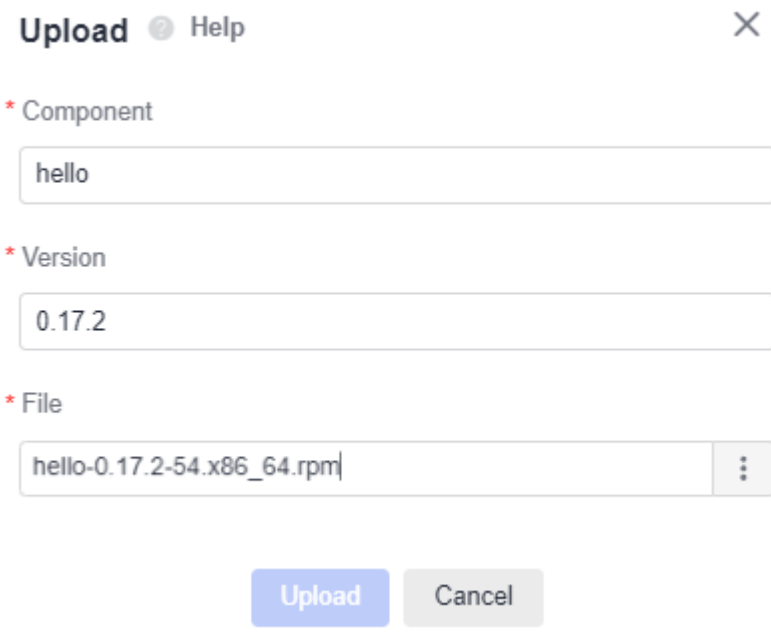
Parameter	Description
Version	Version of the RPM binary package

Step 1 Go to the self-hosted repo page. In the left pane, click the target repository to which the private component is to be uploaded.

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

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

Figure 3-4 Uploading a component



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

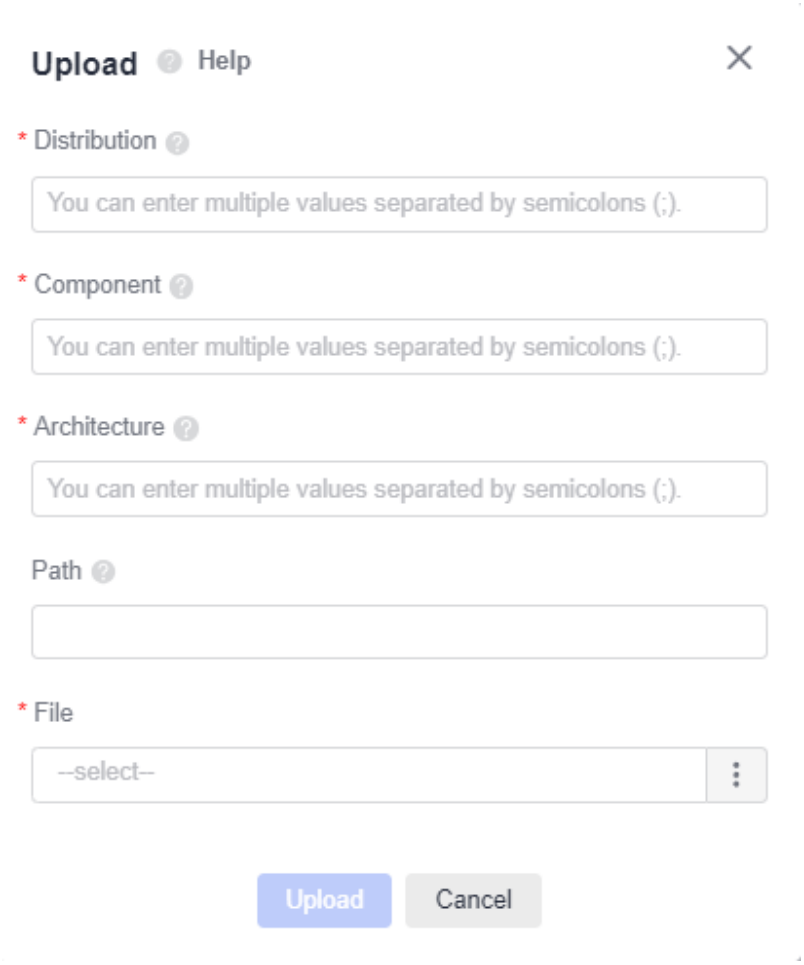
----End

Uploading a Debian Component

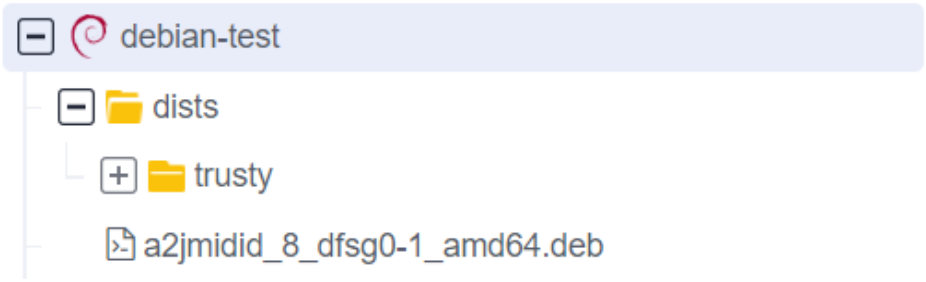
When uploading a Debian component, you need to set the following parameters:

Parameter	Description
Distribution	Release version of the software package

Parameter	Description
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 to be uploaded



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



Uploading a NuGet Component

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

CodeArts Artifact creates a NuGet repository to host and manage NuGet packages.

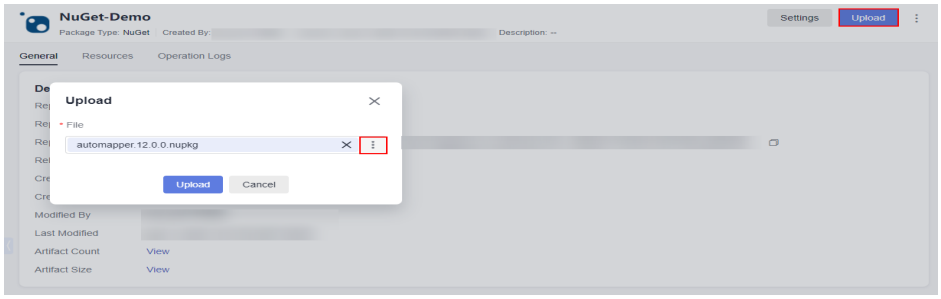
- You are advised to package and name the NuGet file according to the following rules:

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

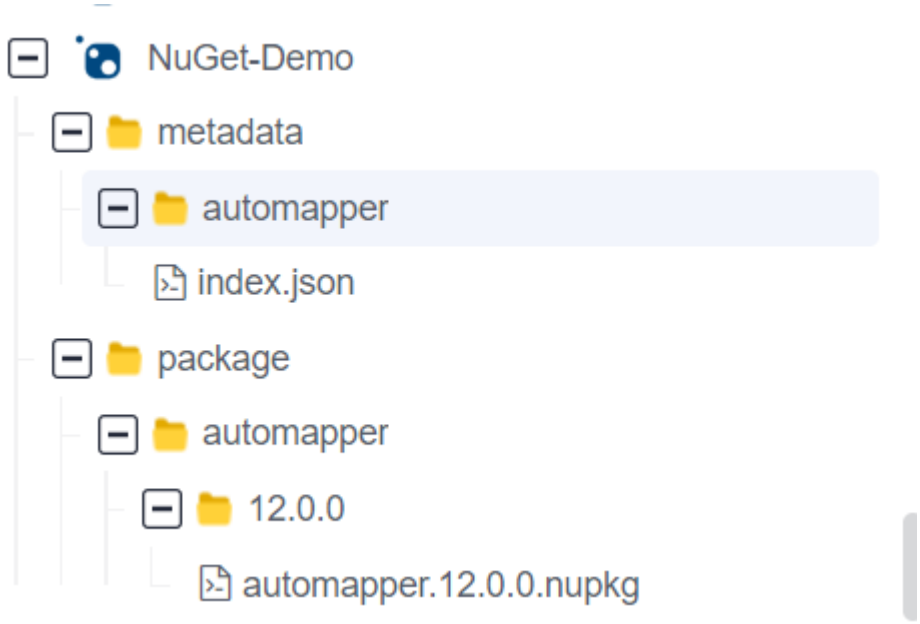
  Example: **automapper.12.0.0.nupkg**

Step 1 Go to the self-hosted repo page. In the left pane, click the target 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 manually. It will be deleted or added automatically when the corresponding component is deleted or restored.

package stores components.

----End

Uploading a Docker Component

Step 1 Go to the self-hosted repo page. In the left pane, click the target Docker repository to which the private component is to be uploaded.

Step 2 Click **Upload** and configure page information (for details, see the following table).

Parameter	Description
Upload Mode	The default upload mode is **Single file**.
PackageName	Enter the artifact package name.
File	Select the Docker file to be uploaded from the local PC.

Step 3 Click **Upload**.

When the progress bar in the lower right corner of the page shows 100%, it indicates the upload is complete.

----End

Uploading a CocoaPods Component

Step 1 Go to the self-hosted repo page. In the left pane, click the target CocoaPods repository to which the private component is to be uploaded.

Step 2 Click **Upload** and configure page information (for details, see the following table).

Parameter	Description
Upload Mode	The default upload mode is **Single file**.
PackageName	Enter the artifact package name.
File	Select the CocoaPods file to be uploaded from the local PC.

Step 3 Click **Upload**.

When the progress bar in the lower right corner of the page shows 100%, it indicates the upload is complete.

----End

Downloading Components

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be downloaded, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

----End

3.6 Uploading/Downloading Components on the Client

3.6.1 Connecting Self-Hosted Repos 2.0 to Your Local Development Environment

By default, the Maven tool connects to the public repository, and is implemented through configurations in the **setting.xml** file. Self-hosted repos can be connected to your local development environment. You can modify the configuration in the **setting.xml** file to connect a self-hosted repo to your local development environment.

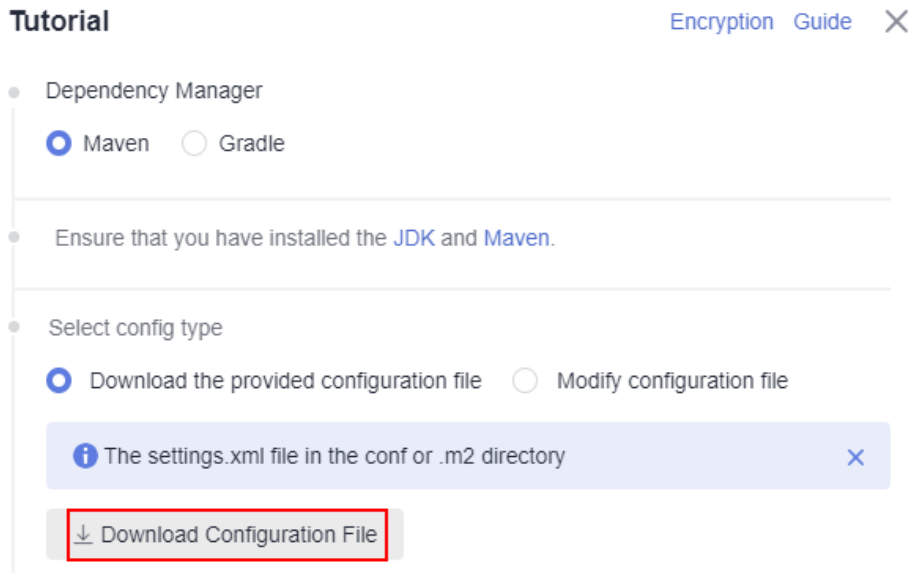
Configuring setting.xml Files

Step 1 Go to the self-hosted repo page. In the left pane, click the name of the repository to be connected to your 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.

Figure 3-5 Downloading the configuration file



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

----End

Resetting Repository Password

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

Step 1 Go to the self-hosted repo page. Click ... on the right of the page and select **Reset Repository Password**.

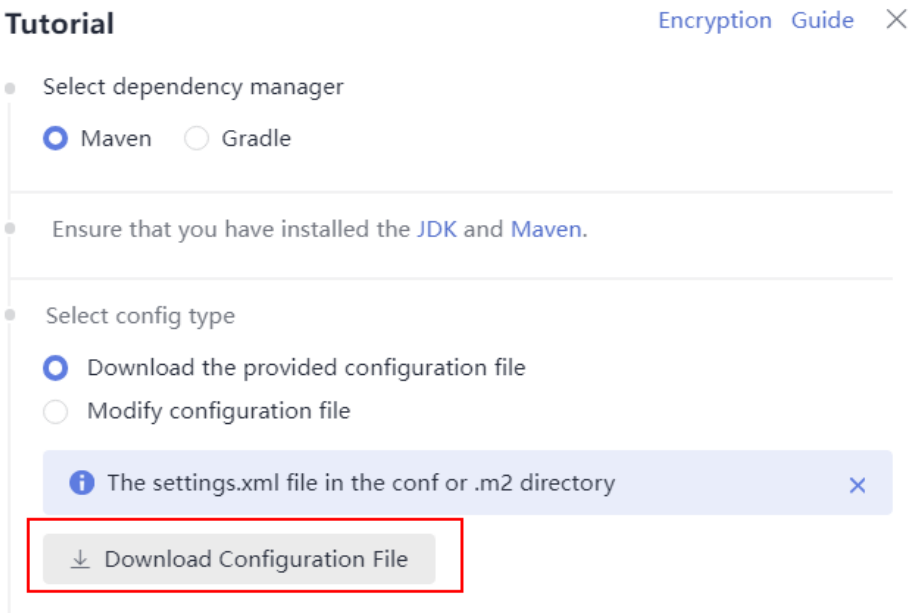
Step 2 In the displayed dialog box, click **OK**. Check that a message is displayed indicating the password has been reset.

----End

3.6.2 Uploading Components to Self-Hosted Repos on the Client

Uploading a Maven Component on the Client

- The client tool is Maven. Ensure that the JDK and Maven have been installed.
  a. Download the **settings.xml** file from the 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 commands to upload a component on the client:

NOTE

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

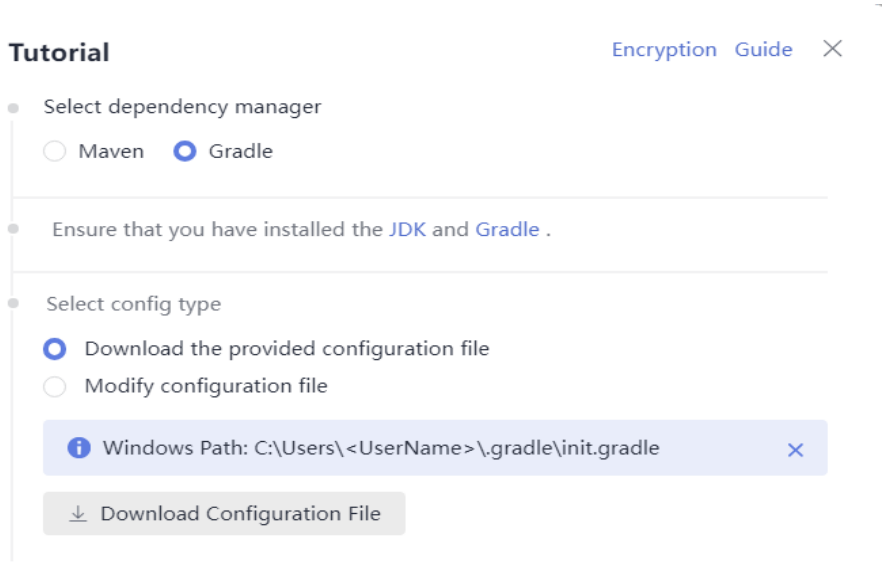
```
mvn deploy:deploy-file -DgroupId={groupId} -DartifactId={artifactId} -Dversion={version} -Dpackaging=jar -Dfile={file_path} -DpomFile={pom_path} -Durl={url} -DrepositoryId={repositoryId} -s {settings_path} -Dmaven.wagon.http.ssl.insecure=true -Dmaven.wagon.http.ssl.allowall=true -Dmaven.wagon.http.ssl.ignore.validity.dates=true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

- **Parameter 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.


```
<server>
    <id>releases</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
</server>
<server>
    <id>snapshots</id>
    <username>[redacted]</username>
    <password>[redacted]</password>
</server>
<server>
    <id>z_mirrors</id>
</server>
```

```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-deploy-plugin:2.7:deploy-file (default-cli) @ standalone-pom ---
Uploading to [redacted]
/1.0/demo-1.0.jar
Uploaded to [redacted]
1.0/demo-1.0.jar (43 kB at 351 B/s)
Uploading to [redacted]
/1.0/demo-1.0.pom
Uploaded to [redacted]
1.0/demo-1.0.pom (162 B at 128 B/s)
Downloading from [redacted]
demo/maven-metadata.xml
Downloaded from [redacted]
emo/maven-metadata.xml (355 B at 768 B/s)
Uploading to [redacted]
/maven-metadata.xml
Uploaded to [redacted]
maven-metadata.xml (309 B at 341 B/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 02:05 min
[INFO] Finished at: 2022-03-26T16:10:15+08:00
[INFO] Final Memory: 12M/205M
[INFO] ------------------------------------------------------------------------
```

- The client tool is Gradle. Ensure that JDK and Gradle have been installed.
  a. Download the **inti.gradle** file from the self-hosted repo page.



  b. Find the **build.gradle** file in the local project and add the following commands to the gradle file:


```
uploadArchives {
    repositories {
        mavenDeployer {repository(url:"***") {
                authentication(userName: "{repo_name}", password: "{repo_password}")
            }
            // Pom file of the build project
            pom.project {
                name = project.name
                packaging = 'jar'
                description = 'description'
            }
        }
    }
}
```

- **url**: path for uploading files to the repository. You can click on the Maven repository page to obtain the path.
- *{repo_name}*: username obtained from the **inti.gradle** file downloaded from the Maven repository page.
- *{repo_password}*: password obtained from the **inti.gradle** file downloaded from the Maven repository page.

c. Run the following command in the directory where the local project is located:

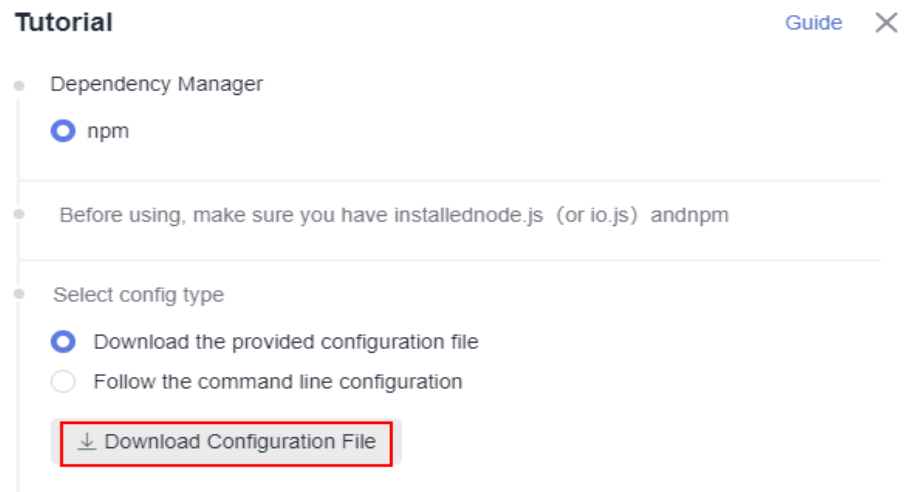
```
gradle uploadArchives
```

d. Return to the target Maven repository to view the uploaded component.

Uploading an npm Component on the Client

- 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 an **.npmrc** file.



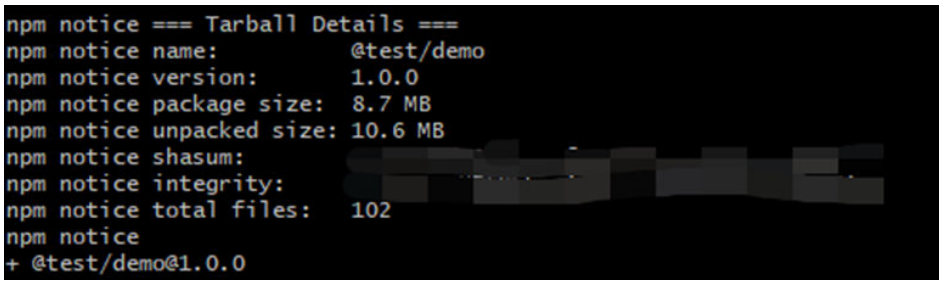
  2. Copy the file to the user directory. In Linux, the path is **~/.npmrc** (C:\Users\<*UserName*>\.npmrc).

  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 **name** field.


```
{
  "name": "@test/demo",
  "version": "1.0.0",
  "description": "demo",
  "main": "index.js",
  "engines": {
    "node": ">= 8.0.0",
    "npm": ">= 5.0.0"
  },
```

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

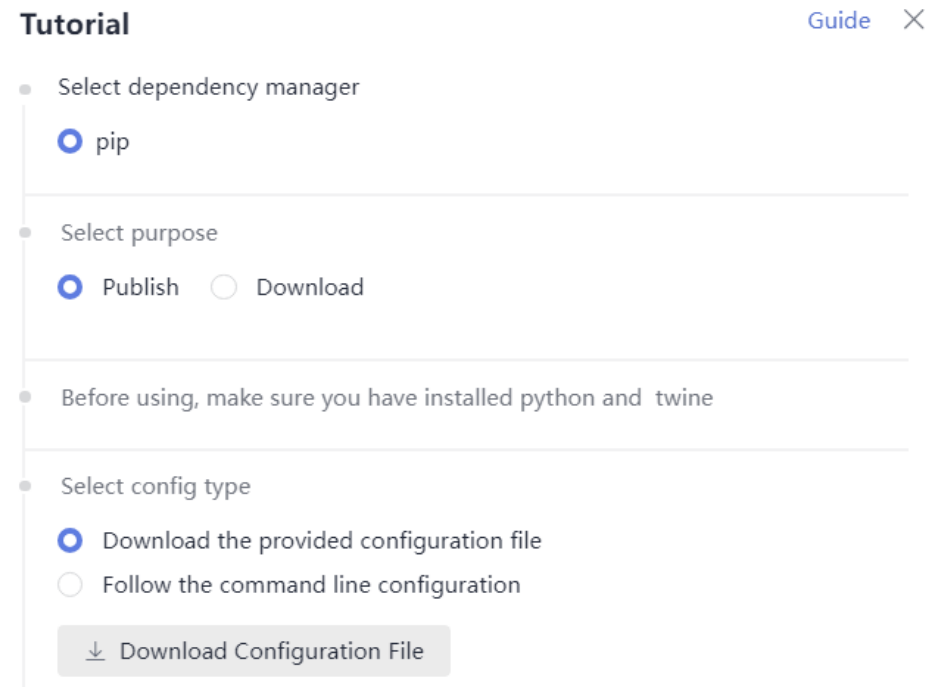
```
npm config set strict-ssl false
npm publish
```



Uploading a PyPI Component on the Client

- 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.



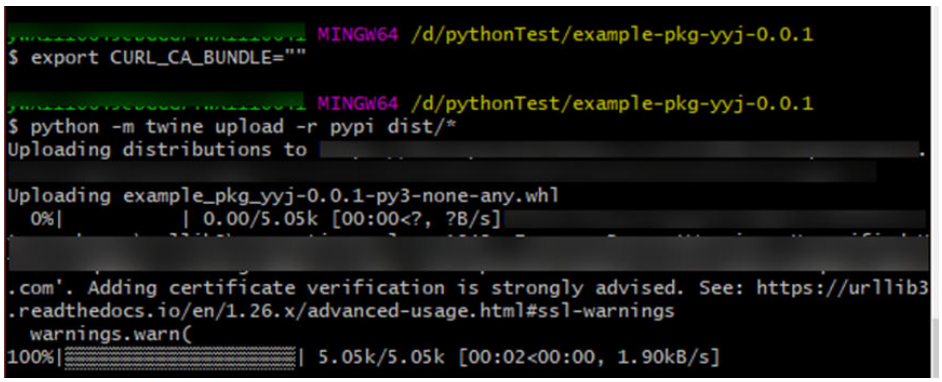
2. Copy the file to the user directory. In Linux, the path is **~/.pypirc**. In Windows, the path is **C:\Users**<*UserName*>**\.pypirc**.

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 reset if you log in to the server again, switch users, or open a new bash window. Be sure to set the environment variables before each upload.

Uploading a Go Component on the Client

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 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 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.

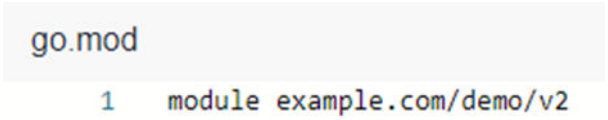
- For the ZIP package, the value of **filePath** is **example.com/demo/@v/v1.0.0.zip** and that of **localFile** is **v1.0.0.zip**.
- For the **go.mod** file, the value of **filePath** is **example.com/demo/@v/v1.0.0.mod** and that of **localFile** is **example.com/demo@v1.0.0/go.mod**.

The commands are 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.



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**.

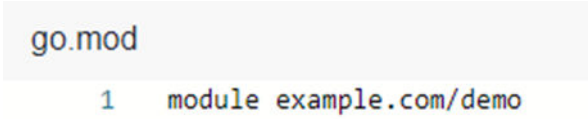
- For the **go.mod** file, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.mod** and that of **localFile** is **example.com/demo/v2@v2.0.0/go.mod**.

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

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

- **v2.0 and later, with the first line in 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 commands are 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
```


Upload an RPM Component on the Client

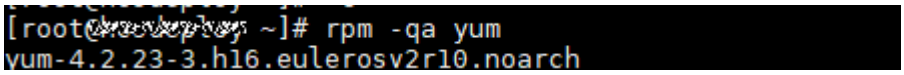
The client tool is Linux OS with Yum. Ensure that the Linux OS is used and Yum has been installed.

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 CodeArts Artifact and access the RPM repository. Click **Tutorial** on the right of the page.

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

Step 4 On the Linux host, run the following 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 to be uploaded. 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

Uploading a Conan Component on the Client

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.
- You have created a Conan repository in the self-hosted repo.

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

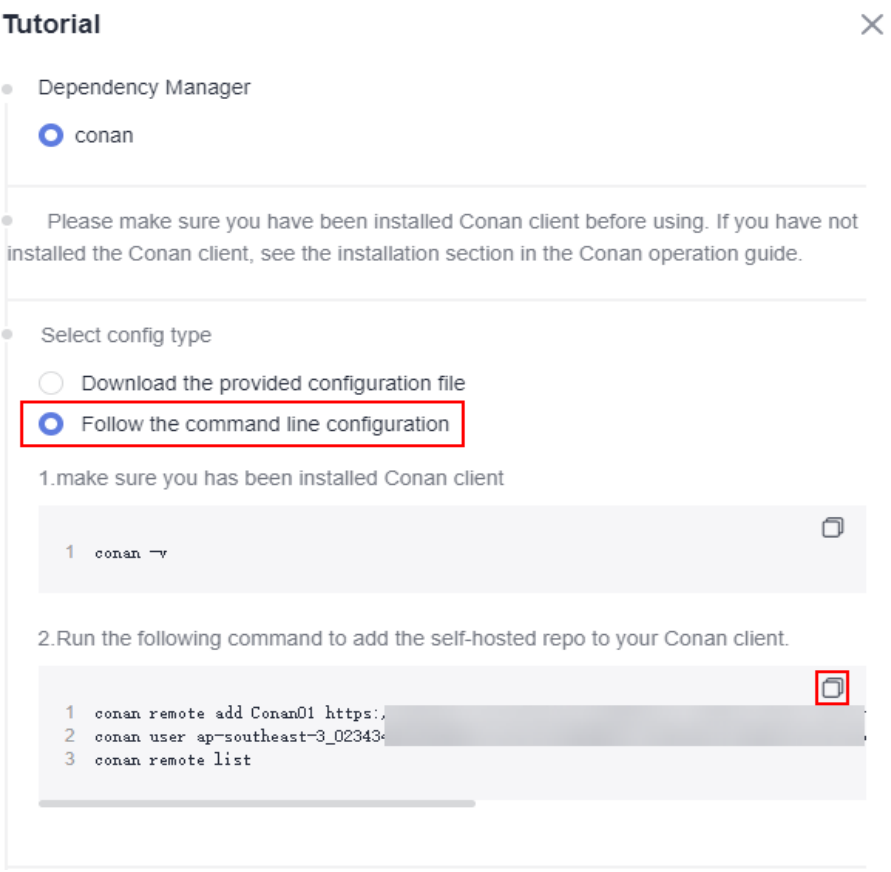
You can replace local Conan configurations with the obtained configuration file (the path is **~/.conan/remotes.json** in Linux or **C:\Users\<UserName>\.conan\remotes.json** in Windows).

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 the remote repository. You can replace it with your own repository.

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

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

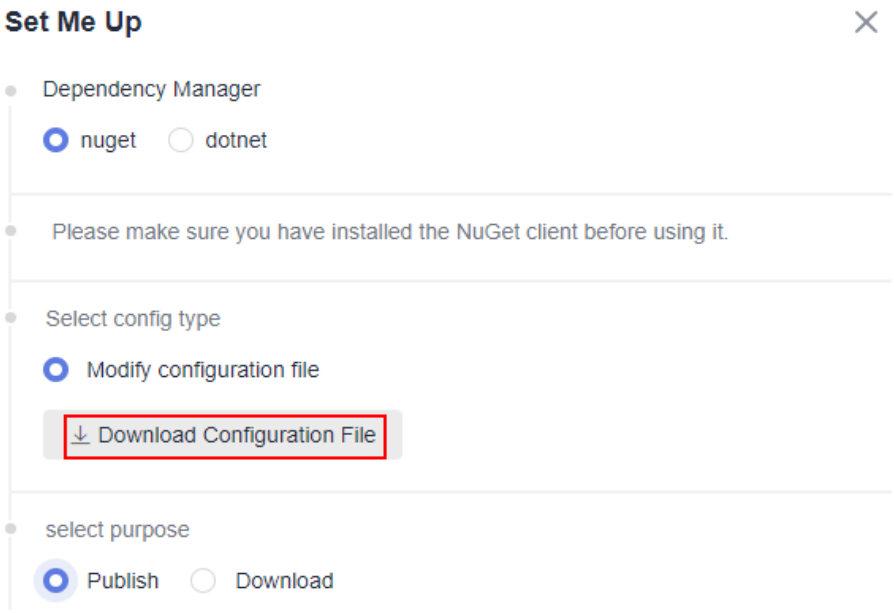
```
$ conan search hello/0.1@demo/testing -r=my_local_server
```

----End

Uploading a NuGet Component on the Client

Ensure that you have installed the NuGet.

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



Step 2 Open the downloaded 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 on the Client

Ensure that you have installed the .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
  ```

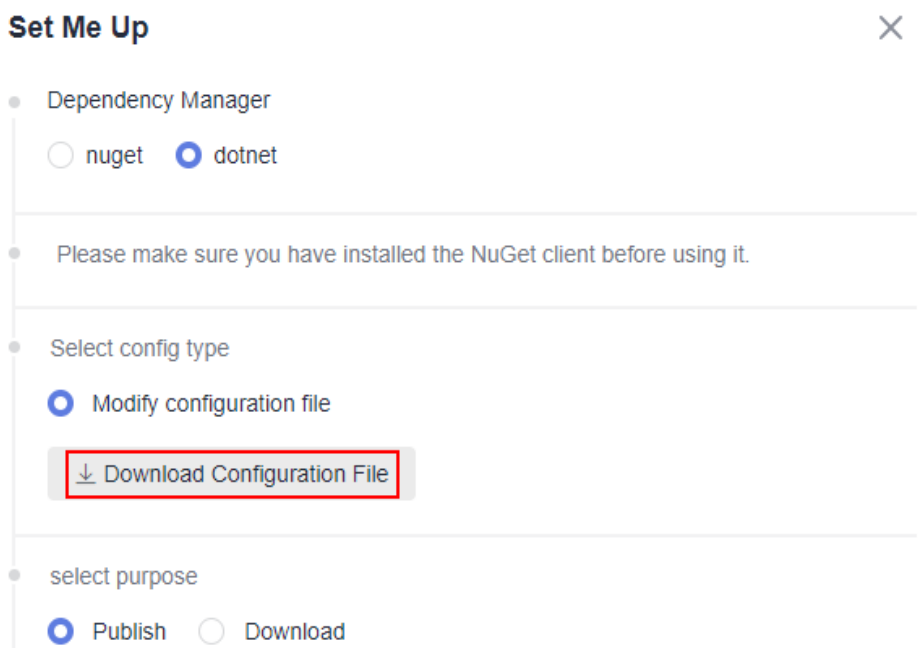
  mycertfile.pem and **mycertfile.crt** are the downloaded certificates.

  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 target NuGet repository from the self-hosted repo page and click **Tutorial** on the right 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

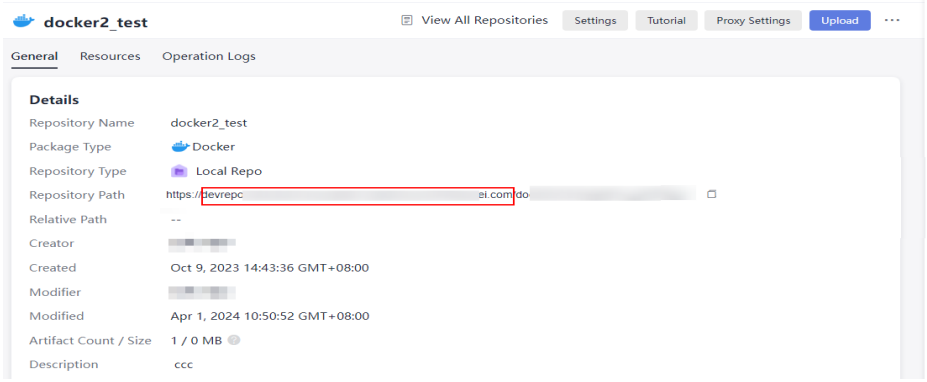
Uploading a Docker Component on the Client

Prerequisites

- You have installed the Docker client.
- You have created a Docker repository in the self-hosted repo.
- Run the following commands on the Docker client to modify the configuration and ignore the certificate:

  ```
  vi /etc/docker/daemon.json
  {
    "insecure-registries": ["url"]
  }
  ```

  {url}: repository path, as shown in the following figure.



Procedure

Step 1 Select the target Docker repository from the self-hosted repo page and click **Tutorial** on the right of the page.

Step 2 Click **Download Configuration File** to download the **config.json** configuration file.

Step 3 Obtain *{username}* and *{password}* from the downloaded file.

Step 4 Run the following command on the local client to log in to the Docker repository:

```
docker login {url} -u ${username} -p ${password}
```

url: repository path.

username: username obtained in **Step 3**.

password: password obtained in **Step 3**.

Step 5 Run the following commands on the local client to package the image:

```
sudo docker tag ${image_name1}:${image_version1} {url}/${image_name2}:${image_version2}
```

image_name1: local image name.

image_version1: local image version.

url: repository path.

image_name2: You can name the uploaded image. The component name is displayed in the component list of the Docker repository.

image_version2: You can set the version of the uploaded image.

Step 6 Run the following command on the local client to upload the Docker component to the self-hosted repo:

```
sudo docker push {url}/${image_name}:${image_version}
```

url: repository path.

image_name: Enter **image_name2** in **Step 5**.

image_version: Enter **image_version2** in **Step 5**.

Step 7 Check the uploaded component in the Docker repository.

----End

Uploading a CocoaPods Component on the Client

Prerequisites

- You have installed the Ruby client and cocoapods-art plug-in.
- You have created a CocoaPods repository in the self-hosted repo.

Downloading configuration files and uploading CocoaPods components to self-hosted repos

Step 1 Select the target CocoaPods repository from the self-hosted repo page and click **Tutorial** on the right of the page.

Step 2 Select **Download the provided configuration file** and click **Download Configuration File** to download the **cocoapods.txt** file.

Step 3 Obtain *{username}* and *{password}* from the downloaded file.

Step 4 Run the following commands on the local client to upload the local CocoaPods software package to the target self-hosted repo:

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME: *{username}* obtained in **Step 3**.

PASSWORD: *{password}* obtained in **Step 3**.

url: repository path of the CocoaPods repository.

TARGET_FILE_PATH: name of the self-hosted repo folder to be stored.

PATH_TO_FILE: local path of the component to be uploaded.

FILE_NAME: name of the self-hosted repo to which the component is uploaded.

Step 5 Check the uploaded component in the CocoaPods repository.

----End

Following the command line configuration to upload CocoaPods components to self-hosted repos

Step 1 Select the target CocoaPods repository from the self-hosted repo page and click **Tutorial** on the right of the page.

Step 2 Select **Follow the command line configuration**.

Step 3 Run the following command to check whether the Rudy client has been installed:

```
rudy -v
```

Step 4 Run the following command to install the cocoapods-art plug-in:

```
sudo gem install cocoapods-art
```

Step 5 Run the following command to add the self-hosted repo to your CocoaPods client.

```
pod repo-art add <repo_name> "{url}"
```

repo_name: name of the folder for storing components of self-hosted repos on the local client.

url: repository path of the CocoaPods repository.

Step 6 In the **Select purpose** area, click **Publish**.

Step 7 Run the following commands to upload the local component to the target CocoaPods repository:

```
curl -k -u "<USERNAME>:<PASSWORD>" -XPUT "{url}/<TARGET_FILE_PATH>/" -T <PATH_TO_FILE>/<FILE_NAME>
```

USERNAME: *{username}* obtained in **Step 3**.

PASSWORD: *{password}* obtained in **Step 3**.

url: repository path of the CocoaPods repository.

TARGET_FILE_PATH: name of the self-hosted repo folder to be stored.

PATH_TO_FILE: local path of the component to be uploaded.

FILE_NAME: name of the self-hosted repo to which the component is uploaded.

Step 8 Check the uploaded component in the CocoaPods repository.

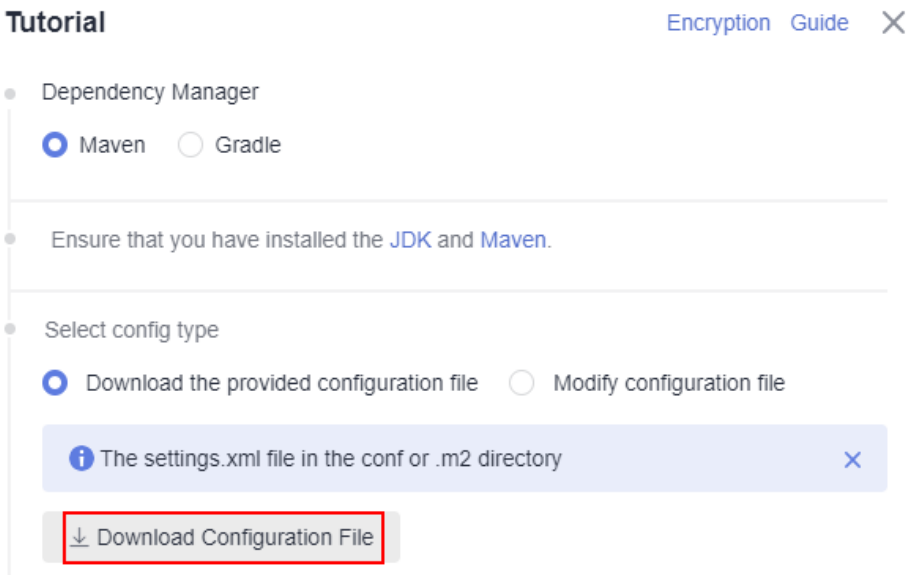
----End

3.6.3 Downloading Components from Self-Hosted Repos on the Client

Downloading a Maven Component on the Client

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

  1. Download the **settings.xml** file from the 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
```

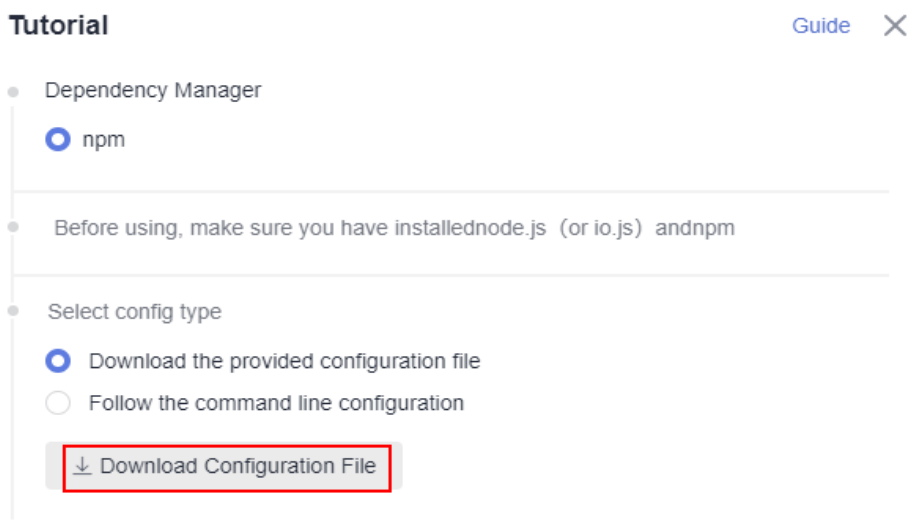


```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.8:get (default-cli) @ standalone-pom ---
[INFO] Resolving demo:demo:jar:1.0 with transitive dependencies
Downloading from
/demo/1.0/demo-1.0.pom
Downloaded from
demo/1.0/demo-1.0.pom (0 B at 0 B/s)
Downloading from
/demo/1.0/demo-1.0.jar
Downloaded from
demo/1.0/demo-1.0.jar (0 B at 0 B/s)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.925 s
[INFO] Finished at: 2022-03-26T16:14:33+08:00
[INFO] Final Memory: 16M/194M
[INFO] ------------------------------------------------------------------------
```

Downloading an npm Component on the Client

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

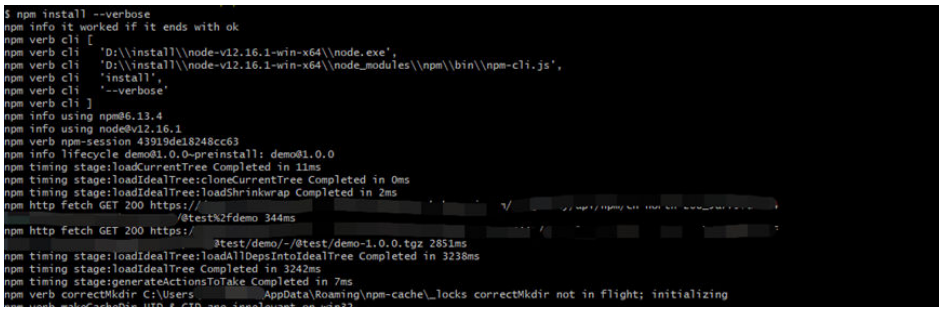
  1. Download the NPMRC file from the self-hosted repo page and save the downloaded NPMRC file as an **.npmrc** file.

  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 dependency component:

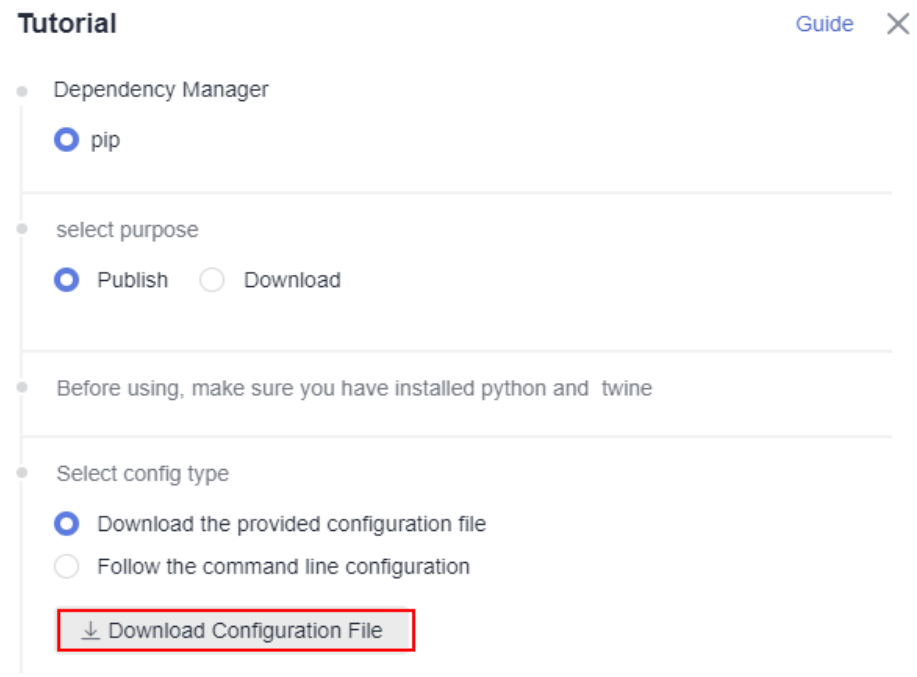
  ```
  npm config set strict-ssl false
  npm install --verbose
  ```

Downloading a PyPI Component on the Client

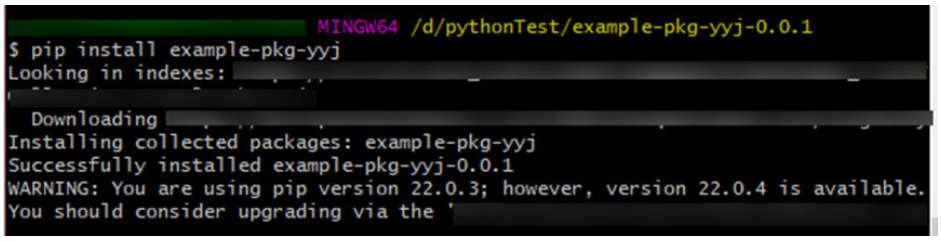
- The client tools are python and pip. Ensure that python and pip have been installed.

  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}
```



Downloading a Go Component on the Client

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

1. Export the certificates.

```
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 certificates 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
```

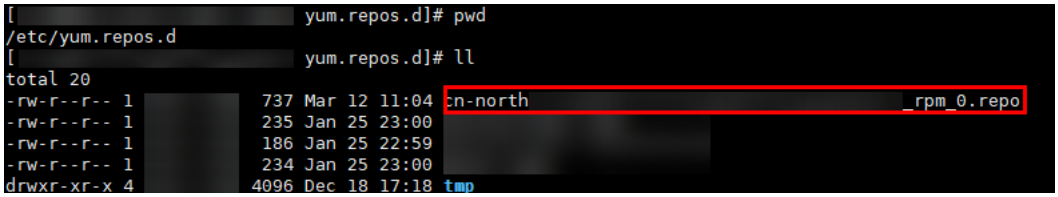
Downloading an RPM Component on the Client

The following section uses the RPM component from **Uploading an RPM Component on the Client** as an example to describe how to obtain dependency packages from the RPM repository.

Step 1 Download the RPM configuration file by referring to **2** and **3**.

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

Downloading a Conan Component on the Client

Step 1 Select the target Conan repository from the self-hosted repo page and click **Tutorial** on the right 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 check the downloaded Conan software package.

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


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

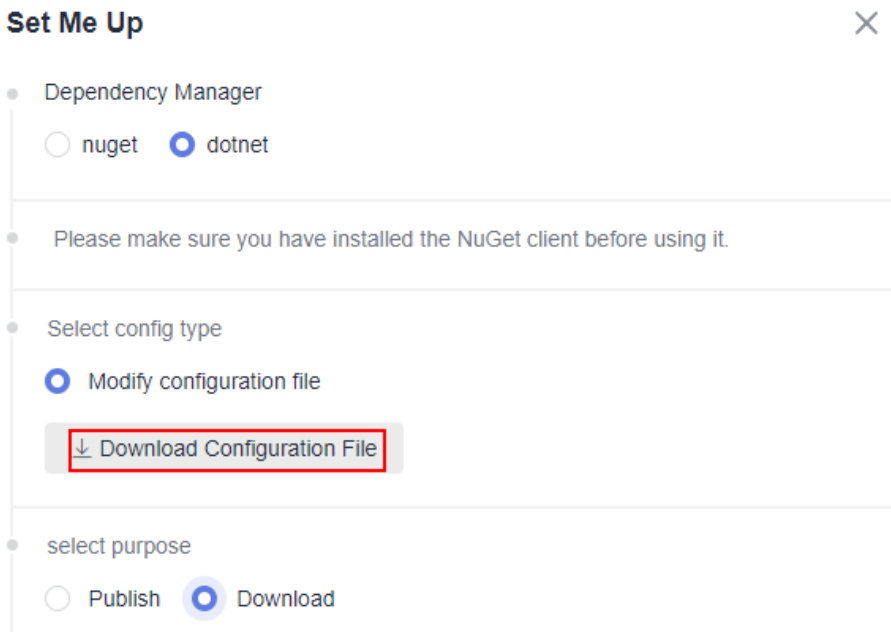
```
$ conan remove ${package_name}/${package_version}@${package_username}/${channel}
```

----End

Downloading a NuGet Component on the Client

Ensure that you have installed the NuGet.

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



Step 2 Open the 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 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**.)

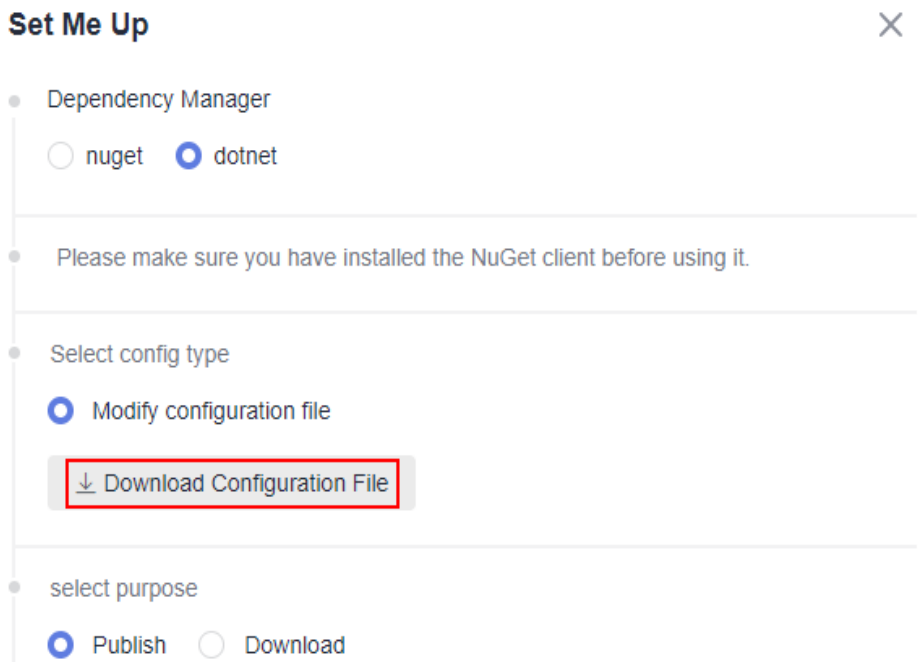
```
##----------------------NuGet Download----------------------##
nuget install <PACKAGE>
```

----End

Downloading a .NET Component on the Client

Ensure that you have installed the .NET.

Step 1 Select the target NuGet repository from the self-hosted repo page and click **Tutorial** on the right 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

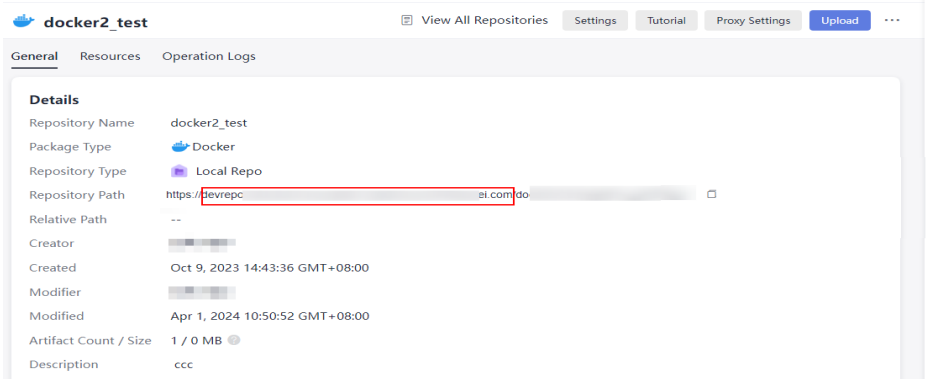
Downloading a Docker Component on the Client

Prerequisites

- You have installed the Docker client.
- You have created a Docker repository in the self-hosted repo.
- Run the following commands on the Docker client to modify the configuration and ignore the certificate:

  ```
  vi /etc/docker/daemon.json
  {
    "insecure-registries": ["url"]
  }
  ```

  {url}: repository path, as shown in the following figure.



Downloading a Docker component on the client

Run the following command on the local client to download the Docker component:

```
sudo docker pull {url}/${image_name}:${image_version}
```

url: repository path.

image_name: component name.

image_version: component version.

Downloading a CocoaPods Component on the Client

Prerequisites

- You have installed the Ruby client and cocoapods-art plug-in.
- You have created a CocoaPods repository in the self-hosted repo.

Downloading the provided configuration file to download CocoaPods components

Step 1 Select the target CocoaPods repository from the self-hosted repo page and click **Tutorial** on the right of the page.

Step 2 Select **Download the provided configuration file**.

Step 3 In the **Select purpose** area, click **Download**.

Step 4 Run the following commands to download the local client:

Run the following command to download the component from the remote repository:

```
pod repo-art add {package_name} {url}
```

package_name: name of the CocoaPods dependency package to be downloaded.

url: repository path of the self-hosted repo.

Run the following command to change the path of the CocoaPods repository:

```
pod repo-art update {package_name} {url}
```

package_name: The CocoaPods dependency package cannot be modified.

url: enter the repository path of the target self-hosted repo.

Run the following command to display the downloaded components.


```
pod repo-art list
```

Run the following command to remove the local dependency package.

```
pod repo-art remove <repo-name>//repo_name: name of the CocoaPods dependency package.
```

----End

Following the command line configuration to download the CocoaPods components

Step 1 Select the target CocoaPods repository from the self-hosted repo page and click **Tutorial** on the right of the page.

Step 2 Select **Follow the command line configuration**.

Step 3 Run the following command to check whether the Rudy client has been installed:

```
rudy -v
```

Step 4 Run the following command to install the cocoapods-art plug-in:

```
sudo gem install cocoapods-art
```

Step 5 Run the following command to add the self-hosted repo to your CocoaPods client.

```
pod repo-art add <repo_name> "{url}"
```

repo_name: name of the folder for storing components of self-hosted repos on the local client.

url: repository path of the CocoaPods repository.

Step 6 In the **Select purpose** area, click **Download**.

Step 7 Run the following commands to download the local client:

Run the following command to download the component from the remote repository:

```
pod repo-art update {package_name}//package_name: component name.
```

Run the following command to display the downloaded components.

```
pod repo-art list
```

Run the following command to remove the local dependency package.

```
pod repo-art remove <repo-name>//repo-name: name of the CocoaPods dependency package
```

----End

3.7 Managing Components 2.0

3.7.1 Viewing a Component

Viewing Components in the Repo View

After you access the self-hosted repo, the repo view is displayed by default. The uploaded components are stored in their respective folders within this view.

Step 1 Go to the self-hosted repo and click  in front of the repository and folder to find the component.

Step 2 Click the component name. The repository details and checksums page are displayed.

Click in the **Details** and **Checksums** tab to copy the information. In the search box, find the target component using the pasted information.

----End

Viewing Components in the Version View

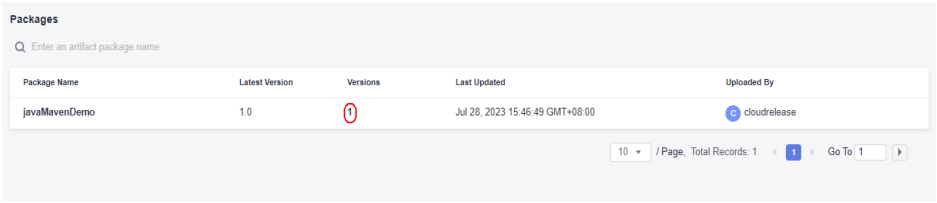
A self-hosted repo displays different component types by version. In **Version View** tab page, you can filter and display artifact packages by name and version number, and sort files by their update time.

Step 1 Go to the self-hosted repo page.

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 package version list is displayed. For details about how to set versions for different components, see **Uploading/Downloading Components on the Self-Hosted Repo Page**.

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

Step 4 Click a number in the **Versions** column. The version list of the software package is displayed.



Click a number in the **Version No.** column. The **Overview** and **Files** pages of the software package are displayed. In the **Files** tab, click a name in the **File Name** column. The path of the software package is displayed.

----End

3.7.2 Editing a Component

Searching for a Component

Step 1 Go to the self-hosted repo page and click **Advanced Search** in the upper right corner of the page.

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. You can search for information using either of the following methods:

- The default file name mode is as follows:
  a. Enter the keyword of the file name in the search box, and click to search for the component.

b. In the search result list, click a file name to view the component details.

- Select **Checksums** mode.
  a. Click the drop-down list on the left of the search box and select **Checksums** (The default value is **File Name**).
  b. Or enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click to find the corresponding component.

----End

Downloading a Component

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be downloaded, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

----End

Deleting a Component

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be deleted, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

Step 3 In the displayed dialog box, enter the name of the component to be deleted and click **Confirm**.

----End

Favoriting/Unfavoriting

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be downloaded, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

When the icon changes to , click **My favorites** in the lower left corner of the page to view the list of favorited components. Click a value in the **path** column to go to the component details page.

----End

3.8 Recycle Bin 2.0

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

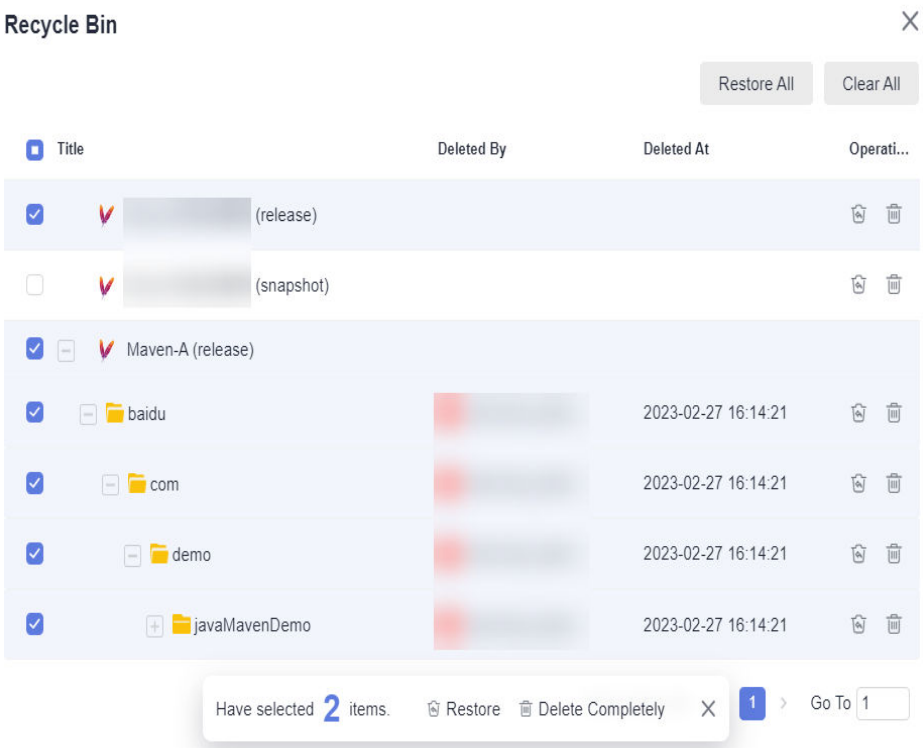
You can manage all deleted components in the recycle bin on the homepage.

Step 1 Access self-hosted repos by referring to **Accessing Self-hosted Repos**.

Step 2 Click **Recycle Bin** in the upper right corner of the page, which will appear on the right side.

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

If both icons and are displayed in the **Operation** column, the repository in that row has been deleted. If only one icon is displayed, the repository still exists, but its components have been deleted. Click the repository name to view the deleted components.



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 them in batches.
	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 the 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.
	Batch delete components	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to permanently delete them in batches.
	Clear all	Click **Clear All** in the upper right corner of the page to delete all repositories and components in the recycle bin.

----End

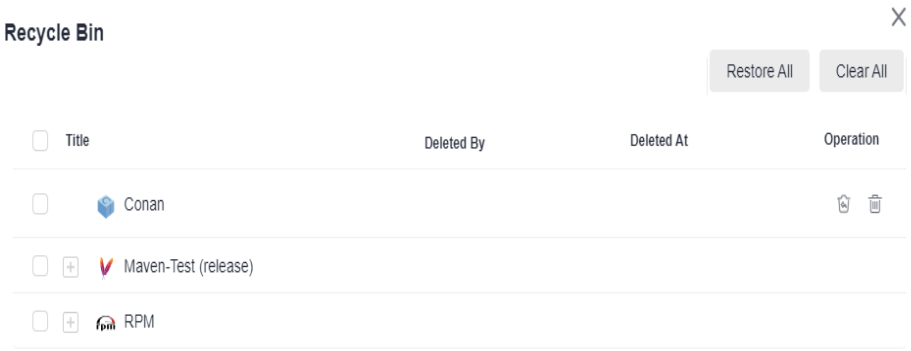
Project Recycle Bin

Step 1 Click a project card to access the project, choose **Artifact** > **Self-hosted Repos** from the navigation pane, and click the target repository name.

Step 2 Click **Recycle Bin** in the lower left 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 both icons  and  are displayed in the **Operation** column, the repository in that row has been deleted. If only one icon is displayed, the repository still exists, but its components have been deleted. Click the repository name to view the deleted components.



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 them in batches.
	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 the 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.
	Batch delete components	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to permanently delete them in batches.
	Clear all	Click **Clear All** in the upper right corner of the page to delete all repositories and components in the recycle bin.

----End

4 Release Repos 1.0

4.1 Accessing Release Repos 1.0

CodeArts Artifact was upgraded in March 2023. Inventory Release Repos and resources belonging to users who registered before this update are stored in the old Release Repos.

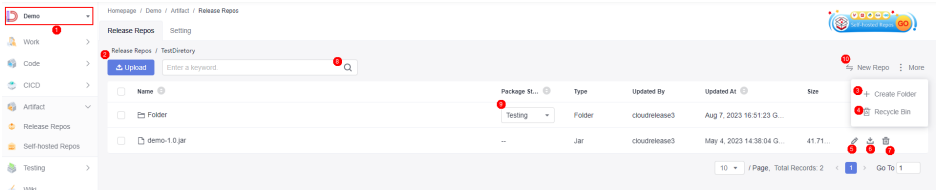
If you have existing repositories and resources, you can upload and manage software packages in the legacy Release Repos. After you create a project, Release Repos with the same project name is automatically generated in the new Release Repos.

Accessing Release Repos 1.0

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

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 Repos.

No.	Operation	Description
2	Upload	Click **Upload** in the upper left corner of the list to manually upload local software packages to Release Repos. 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 Repos.
3	Create folder	Click **More** and select **Create Folder** in the upper right corner to create a folder on the current page.
4	Recycle bin	Click **More** and select **Recycle Bin** in the upper right corner to access the recycle bin, where you can manage deleted software packages or folders.
5	Edit	Click in the **Operation** column to edit the 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 use space in Release Repos. The space will be freed up only after a package is permanently deleted from the recycle bin.
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.

No.	Operation	Description
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 Repos.

----End

4.2 Managing Software Packages 1.0

Uploading Software Packages 1.0

Step 1 Access Release Repos 1.0.

Step 2 Click **Upload**.

Step 3 In the displayed dialog box, enter the required information and click **Upload**.

- **Target Repository**: current Release Repos.
- **Version**: set the version number for software packages.
- **Upload Mode**: select **Single file** or **Multiple files**. **Single file** is selected by default here.
- **Path**: After you set the path name, a folder with that name is created in the **Repo View**, where the uploaded software packages will be stored.
- **File**: select software packages from your local PC to upload.

Step 4 The uploaded software package is saved in the file list of Release Repos.

Click in the **Operation** column of a software package to change its name.

Click in the **Operation** column of a software package to download it to the local PC.

Click in the **Operation** column of a software package to delete it.

----End

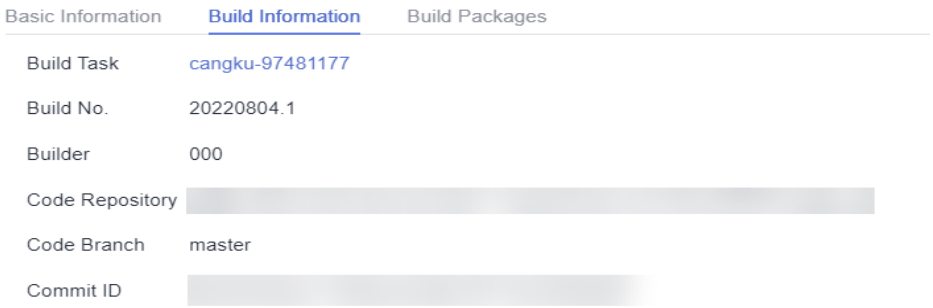
Viewing/Editing Software Packages in Release Repos 1.0

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

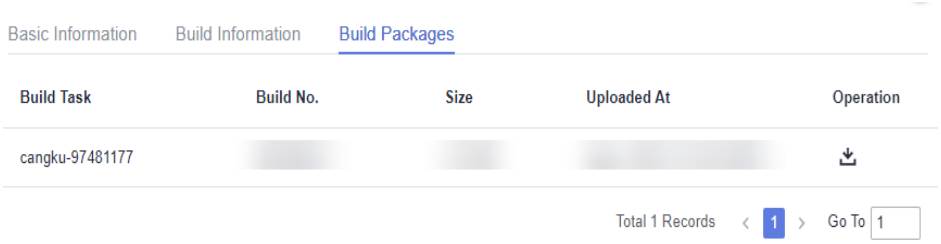
Access the Release Repos homepage and click the name of a software package. A drawer is displayed, showing the software package details. The Release Repos details are displayed on the **Basic Information**, **Build Information**, and **Build Packages** tab pages.

- 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.



4.3 Clearing Policies 1.0

Release Repos automatically clears 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.

Setting Clearing Policies

Step 1 Click a project card to access the project, and choose **Artifact** > **Release Repos** from the menu bar.

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

Step 3 Click the **Settings** tab page.

Step 4 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 Recycle Bin**: 15 days
- **Clear from Recycle Bin**: 15 days

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

NOTE

Parameters below are optional.

- **Skip files with released status**: The system retains files in the production package state when clearing files.
- **Skip specified paths**: The system retains software packages that match the file paths set by users when clearing files. You can set multiple file paths, each starting with a slash (/) and separated by semicolons (;).

----End

4.4 Recycle Bin 1.0

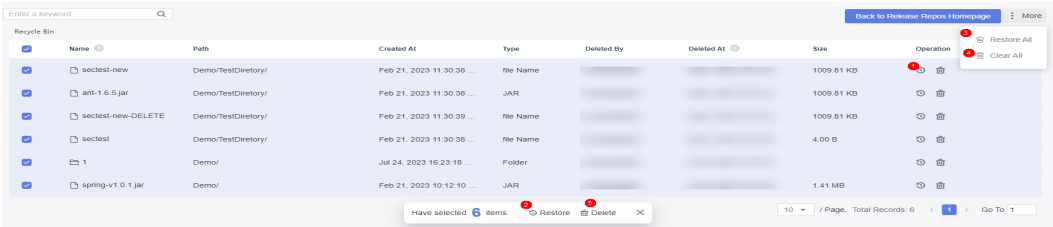
Software packages or folders deleted from the Release Repos are moved to the recycle bin, where you can manage them.

Recycle Bin

Step 1 Click **Old Repo** in the lower part of the page.

Step 2 Click **More** and select **Recycle Bin**.

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



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

NOTICE

Once you delete a software package or folder from the recycle bin, it cannot be recovered. Exercise caution when performing this operation.

----End

5 Self-Hosted Repos 1.0

5.1 Accessing a Self-hosted Repo 1.0

In March 2023, CodeArts Artifact was upgraded. Before this upgrade, existing self-hosted repos were not associated to projects. Consequently, repos and their resources from before the upgrade remain in the old repos.

The features available in Self-hosted Repos 2.0 are also supported in Self-hosted Repos 1.0.

Accessing a Self-hosted Repo 1.0

Step 1 Click a project card to access the project.

Step 2 Choose **Artifact** > **Self-hosted Repos** from the navigation pane.

Step 3 Click **Old Repo** in the lower left corner of the page to view the existing self-hosted repos.

----End

5.2 Configuring a Self-hosted Repo 1.0

A new member must be assigned a specified role to use CodeArts Artifact. For details, see **Configuring Repository Permissions 2.0**.

Managing Repository Permissions

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

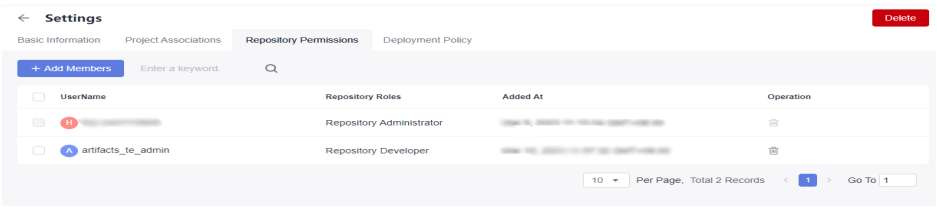
- The project creator and project manager are repository administrators.
- The developer, test manager, tester, and operation manager are repository developers.
- The participant, viewer, and custom roles are repository viewers.

To add or remove 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.

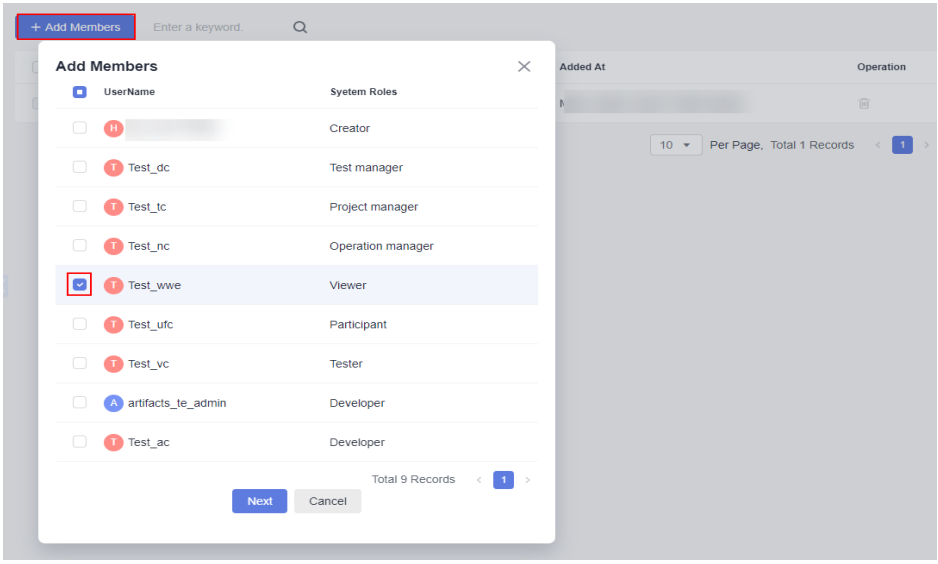
Step 3 Click the **Repository Permissions** tab. The added repository members are displayed in the list.



Step 4 Add members.

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

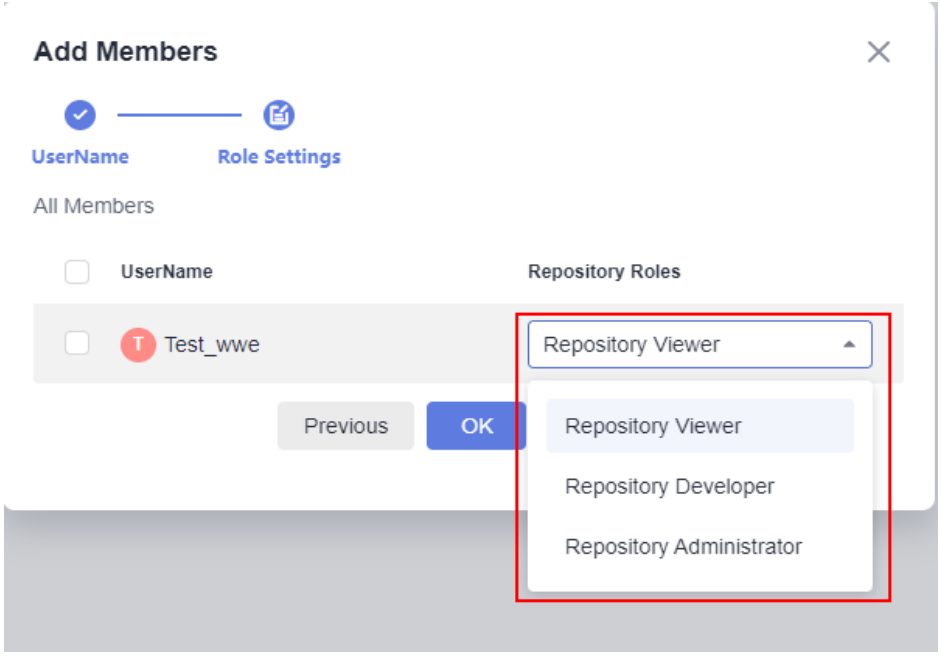
Figure 5-1 Adding members



Step 5 Assign roles to members.

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

Figure 5-2 Assigning roles to members



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

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

Figure 5-3 Configuring repository roles in batches

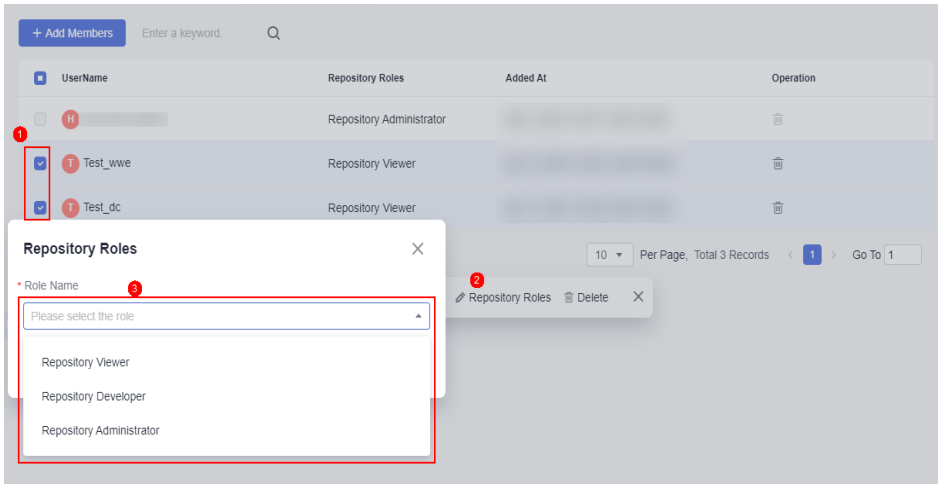


Table 5-1 lists the operation permissions of each repository role.

Table 5-1 Role permissions of a self-hosted repo

Operation/Role	Tenant administrator	Non-tenant administrator

	Repository administrator	Developer	Viewer	Repository administrator	Developer	Viewer
Create a repository	√	√	√	×	×	×
Edit a repository	√	√	√	×	×	×
Manage the association between repositories and projects	√	√	√	×	×	×
Upload a component	√	√	×	√	√	×
Download a component	√	√	√	√	√	√
Delete components	√	√	×	√	√	×
Restore components	√	√	×	√	√	×
Permanently delete a component	√	√	×	√	√	×
Delete a repository	√	×	×	×	×	×
Restore a repository	√	√	×	√	√	×

Permanently delete a repository	√	×	×	×	×	×
Clear all	√	√	√	×	×	×
Restore all	√	√	√	×	×	×
Manage user permissions	√	√	√	√	×	×

----End

Managing User Permissions in Batches by Tenant Account/Repository Administrator

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

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

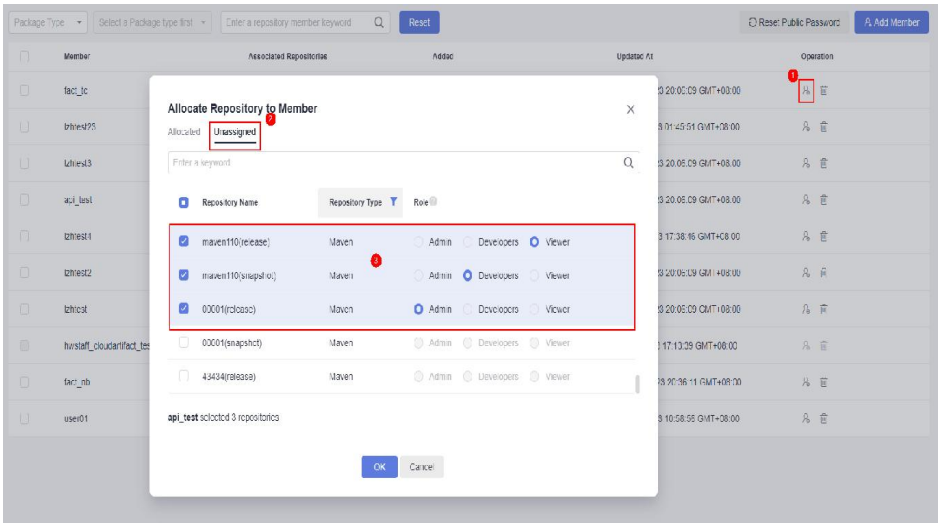
Step 2 Choose **Artifact** > **User Permissions** from the navigation pane.

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

Step 4 Assign roles to members.

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

Figure 5-4 Assigning repository roles



Step 5 (Optional) Perform the operations listed in **Table 5-2** on the **User Permissions** page.

Table 5-2 Related operations

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 members	Select a repository and click in the member list to remove members from the repository. **NOTE** Removing a member from a repository does not affect their role or permissions in other repositories. Deleting a member removes them from all related repositories, along with their associated permissions.
Search for members	Enter a member name or keyword in the search box at the top of the page, and click to search for a repository member.

Operation	Description
Reset public password (only for tenant administrators)	The public password is used by the CodeArts Build to upload and download components to a 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.

----End

5.3 Managing a Self-Hosted Repo 1.0

5.3.1 Checking Basic Information and Adding Path Patterns

Step 1 Go to the self-hosted repo page. In the left pane, click the name of the target repository to be edited.

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

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

NOTE

On the basic information page, the repository name, package type, project, and version policy cannot be modified.

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

Click 🗑 to delete paths.

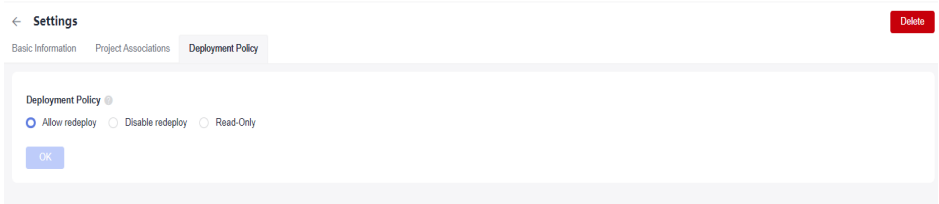
----End

5.3.2 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 Go to the self-hosted repo page. In the left pane, click the name of the target repository.

Step 2 Click **Settings** on the right of the page and click the **Deployment Policies** tab.



- **Allow redeploy** (enabled by default): Artifacts in the same path can be uploaded, replacing the original package.
- **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 Once you complete the settings, they are automatically saved by the system.

----End

5.3.3 Configuring Clearing Policy for a Maven Repository

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

A Maven snapshot is a special version that reflects the current state of ongoing development and differs from regular versions. Maven checks for new snapshots in the remote repository with each build and allows you to set limits on how many snapshot versions to retain, as well as automatically clear out expired snapshots.

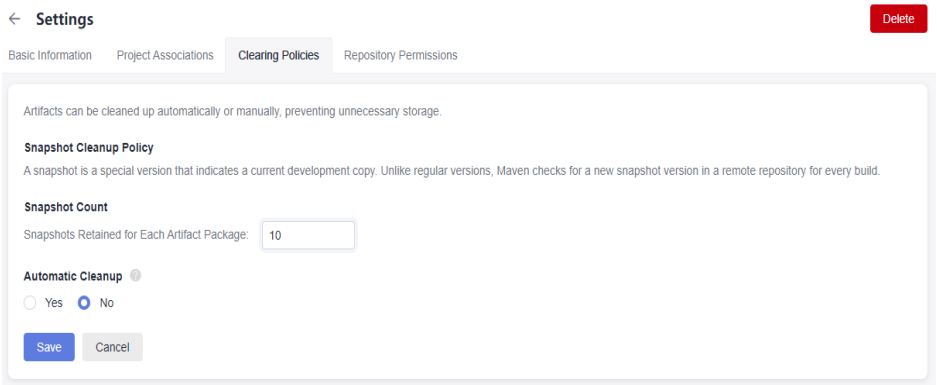
The clearing policy reduces the waste of storage space, makes artifacts in the repository clear, and ensures that artifacts are transferred in order during development, testing, deployment, and release.

Step 1 Click a project card to access the project and choose **Artifact** > **Self-hosted Repos** from the navigation pane.

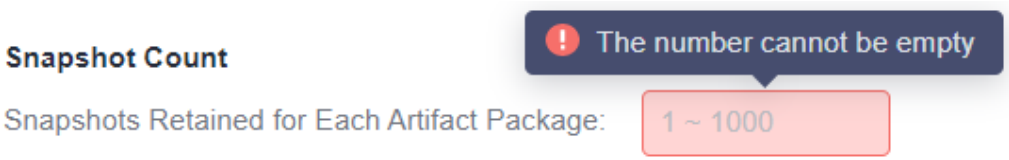
Step 2 Select a Maven repository (**Snapshot**) from the list on the left and click **Settings** in the upper right corner of the page.

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

Figure 5-5 Clearing policies



Step 4 Set the maximum number of **Snapshot Count**. The value ranges from 1 to 1,000.



When the number of retained snapshots exceeds the set limit, the oldest snapshot is replaced by the latest version.

Step 5 Enable automatic cleanup (**No** by default). Click **Yes** and enter the number of days. Snapshots older than the specified number of days will be automatically cleaned up.

The automatic cleanup time must be set between 1 and 100 days.

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

----End

5.3.4 Associating Maven Repository with Projects

After the Maven repository is associated with multiple projects, you can select this Maven repository in the build step of the build task in the project to store the build product to the repository.

Step 1 Go to the self-hosted repo page. In the left pane, click the name of a Maven repository.

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

Step 3 Find the target project to be associated in the list and click in the **Operation** column of the corresponding row.

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

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

----End

In CodeArts Build, you need to upload build products to self-hosted repos. For details, see **Using the File From the Self-hosted Repo to Build with Maven and Uploading the Resulting Software Package (Built-in Executors, GUI)**.

5.4 Managing Components 1.0

5.4.1 Uploading Components on the Self-Hosted Repo Page

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 Go to the self-hosted repo page. In the left pane, click the target repository to which the private component is to be uploaded.

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

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

Detailed configuration for each component type is described below.

NOTE

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

----End

Maven Components

- 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 run, Maven searches for the POM in the current directory, reads its content, obtains the required configuration information, and builds 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 component package. It stands for **Group ID**, **Artifact ID**, and **Version**. **Group ID** indicates a company or organization. For example, Maven core components are in the **org.apache.maven** organization. **Artifact ID** indicates the name of the component package. **Version** indicates the version of the component package.
- Maven dependencies are crucial for POM files. The building and running of most projects depend on the dependency on other components. Add the dependency list to your 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>
```

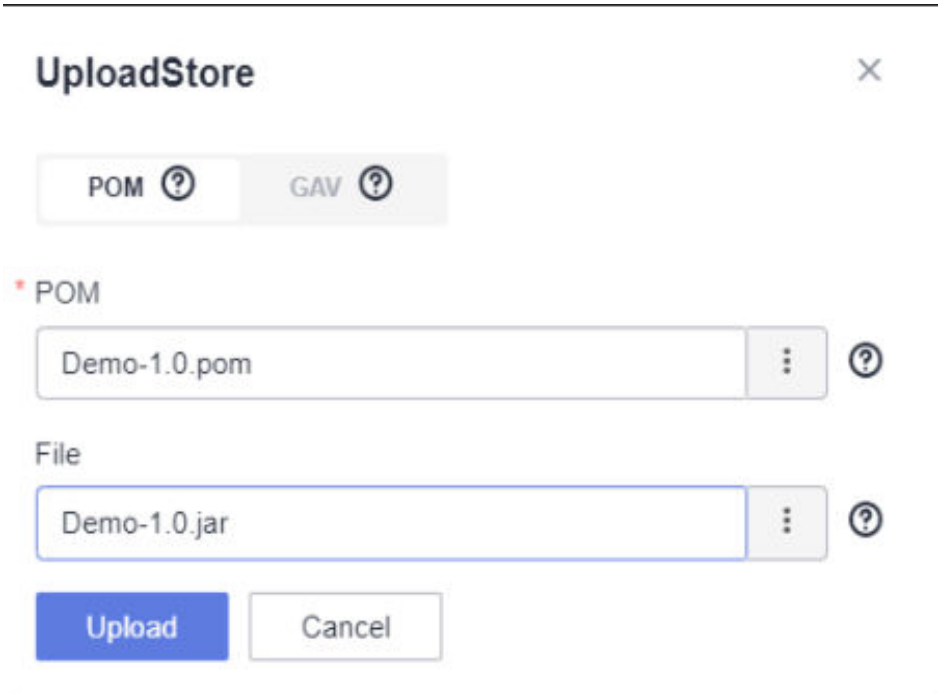
Uploading a Maven Component

POM and GAV upload modes are supported.

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 either upload just the POM file or both the POM file and its related components. The uploaded file must match the **artifactId** and **version** values specified in the POM. As shown in the following figure, if the **artifactId** value is **demo** and the **version** value is **1.0** in the POM, then the uploaded file must be named **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** values in **POM** must match the file name in **File**. For example, if the **artifactId** value is **demo** and the **version** value is **1.0** in **POM**, then the uploaded file must be named **demo-1.0** in **File**.

- GAV

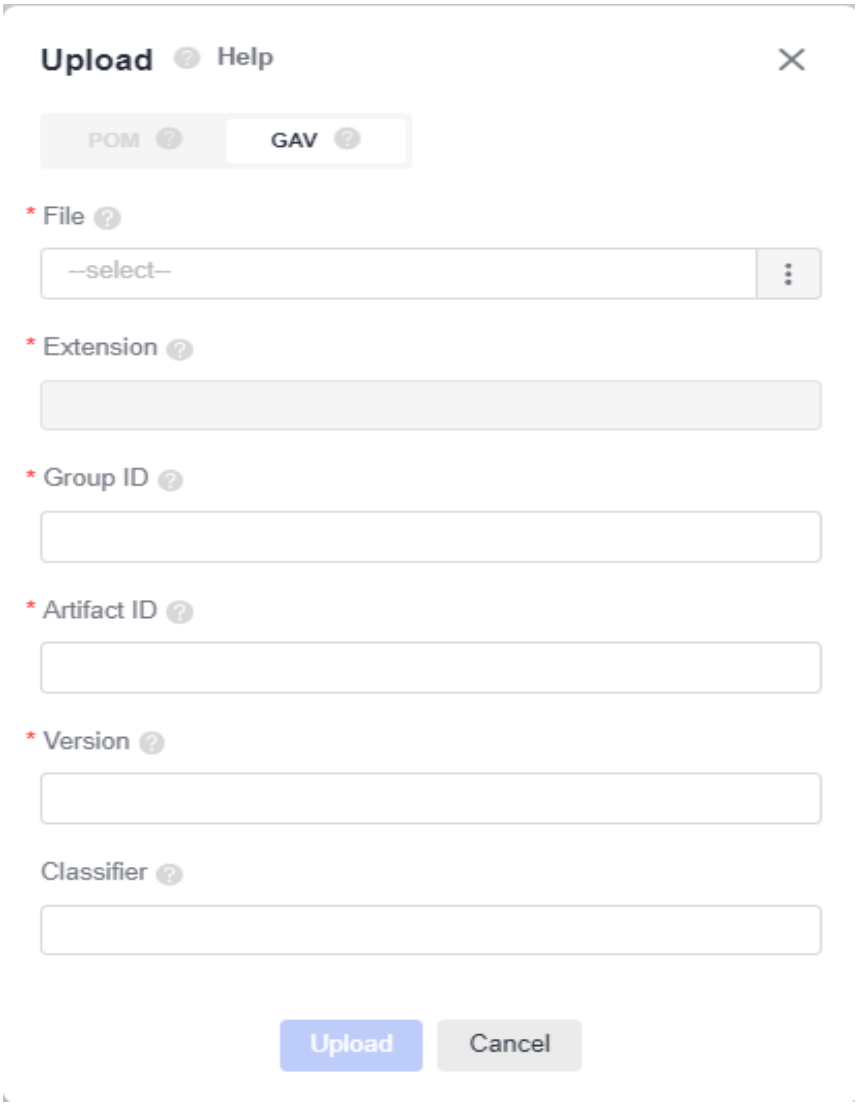
  In the GAV mode, the **Group ID**, **Artifact ID**, and **Version** parameters must be manually specified and they determine the name of the file to upload. **Extension** indicates the packaging type and determines the file type to be uploaded.

  Classifier is used to differentiate artifacts that are built from the same POM but contain different contents. This field is optional. It can contain letters, digits, underscores (_), hyphens (-), and dots (.). If you specify this field, it will be appended to the file name.

  Common usage scenario

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

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



npm Components

Node Package Manager (npm) is a JavaScript package management tool. An npm component package is the item managed by npm, and the npm repository is used to store and manage these packages.

The npm component package consists of a 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 command
    "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 the 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"
}
```

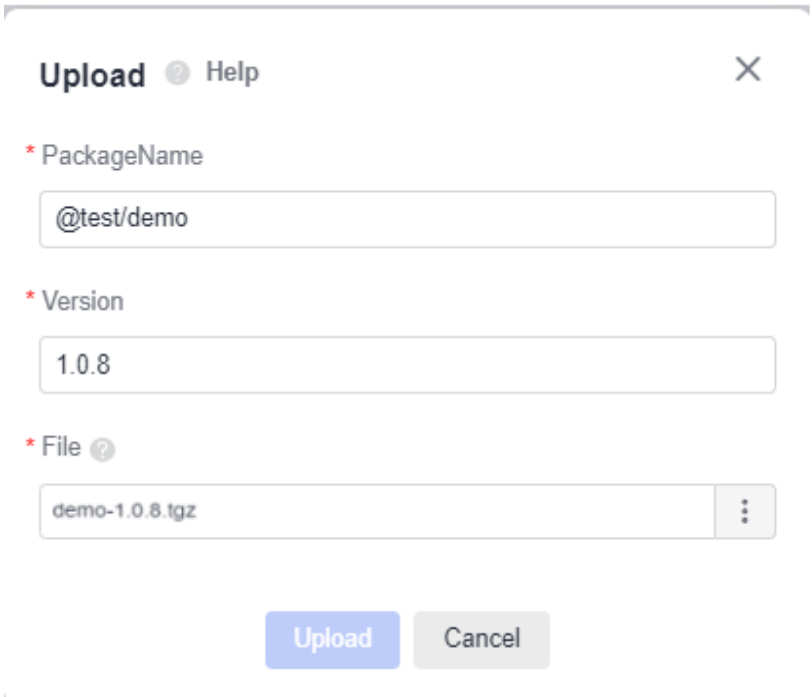
version indicates the version of the package, which is in the x.y.z format.

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

Uploading an npm Component

npm component packages in .tgz format can be uploaded to self-hosted repos. When uploading packages, you need to set the following parameters.

Parameter	Description
PackageName	The value must be the same as that 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 when you create an npm repository.

When uploading components to the repository, make sure that the **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 find the component in .tgz format in the repository list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Component

Go, also known as Golang, is a programming language developed by Google. Golang 1.11 and later support modular package management. A module is a unit for source code exchange and versioning in Go. A MOD file identifies and manages a module. A ZIP file is a source code package. There are two types of Go modules: v2.0 and later and v2.0 and earlier. The management of the Go module is different between these two versions.

To upload a Go component, you need to upload both a ZIP file and a MOD file and 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, this package is generated in the **dist** directory of your project directory. Note that the Python 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 that of **name** in the **setup.py** file.
Version	The value must be the same as that of **version** in the **setup.py** file.

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

Uploading an RPM Component

Red Hat Package Manager (RPM), developed by Red Hat, is used by many Linux distributions. It is a software management system that installs software on 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
Version	Version of the RPM binary package

Step 1 Go to the self-hosted repo page. In the left pane, click the target repository to which the private component is to be uploaded.

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

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

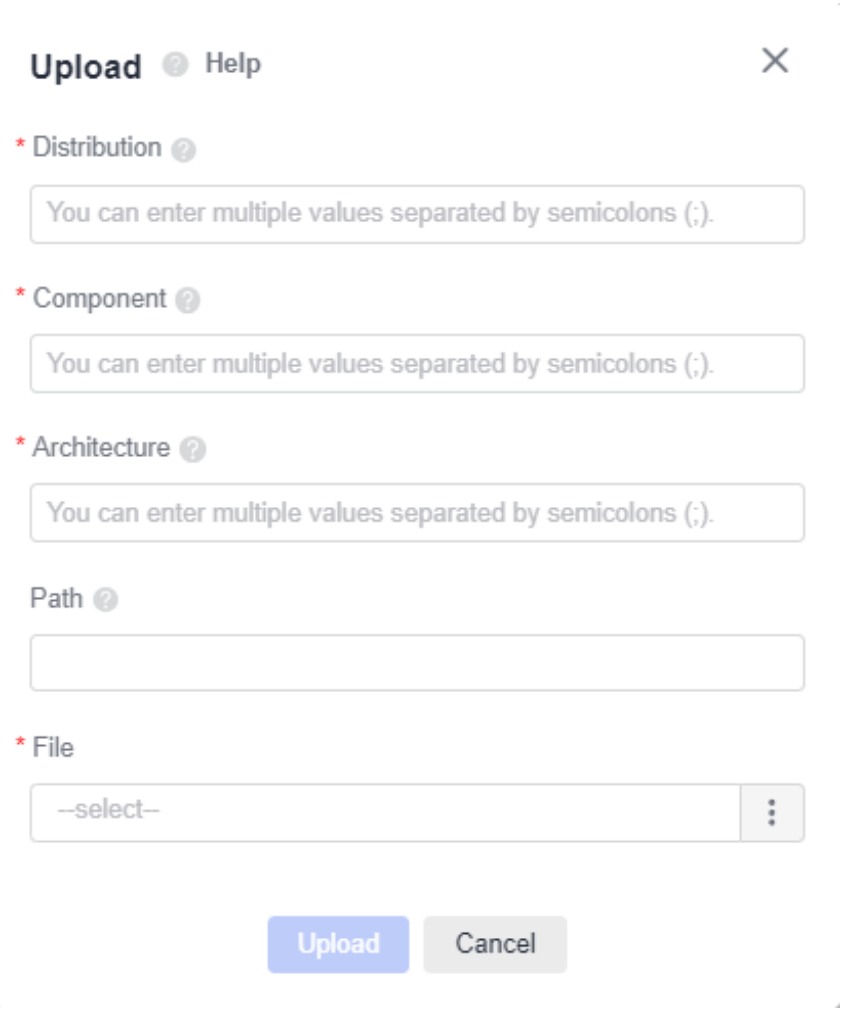
----End

After the upload is successful, you can find the RPM binary package in the repository 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 to be uploaded



After the upload is successful, you can find the installation package in **.deb** format in the repository 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. Users can use the NuGet package to share code specific to an organization or workgroup.

CodeArts Artifact supports uploading local NuGet packages to the self-hosted repo.

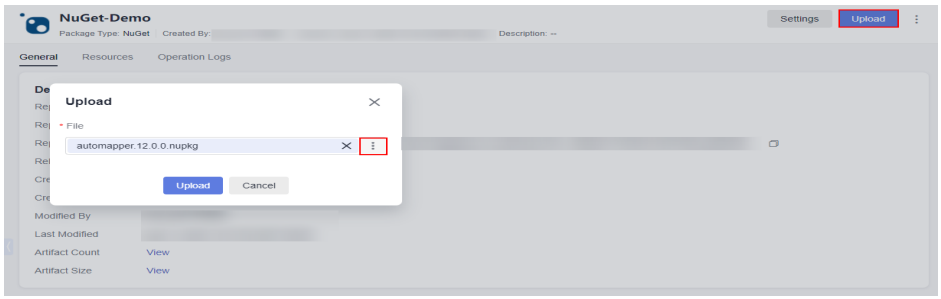
You are advised to package and name the NuGet file according to the following rules:

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

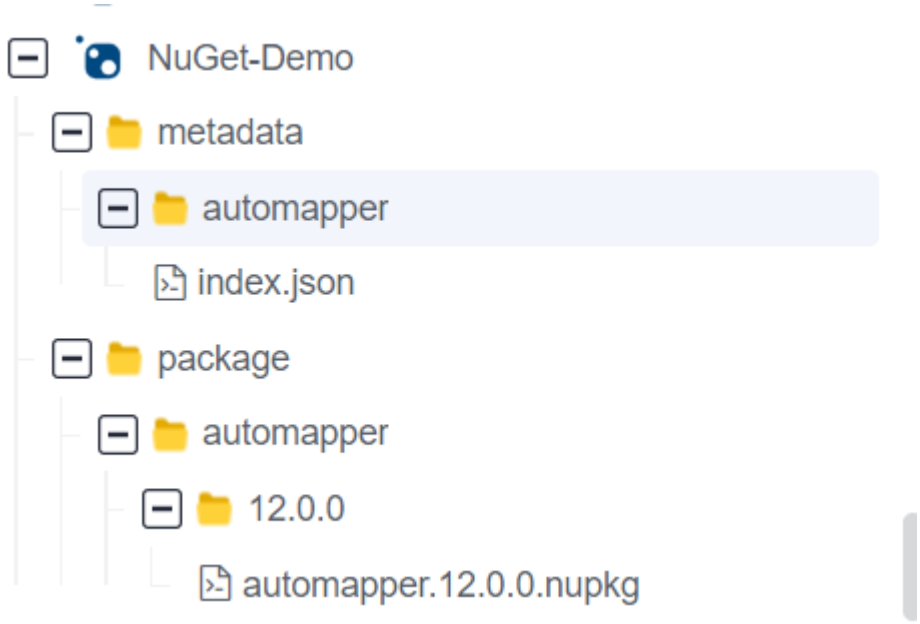
Example: **automapper.12.0.0.nupkg**

Step 1 Go to the self-hosted repo page. In the left pane, click the target 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 manually. It will be deleted or added automatically when the corresponding component is deleted or restored.

package stores components.

----End

5.4.2 Viewing a Component

Viewing Components in the Repo View

After you access the self-hosted repo, the repo view is displayed by default. The uploaded components are stored in their respective folders within this view.

Step 1 Go to the self-hosted repo and click in front of the repository and folder to find the component.

Step 2 Click the component name. The repository details and checksums page are displayed.

Click in the **Details** and **Checksums** tab to copy the information. In the search box, find the target component using the pasted information.

----End

Viewing Components in the Version View

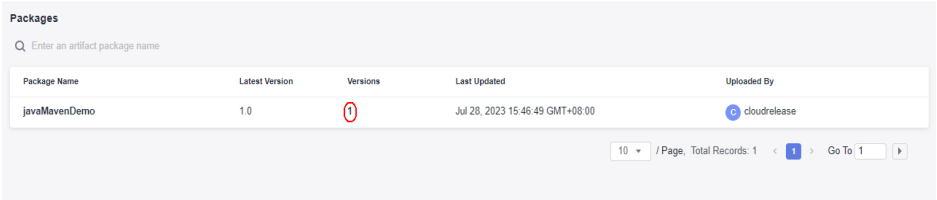
A self-hosted repo displays different component types by version. In **Version View** tab page, you can filter and display artifact packages by name and version number, and sort files by their update time.

Step 1 Go to the self-hosted repo page.

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 package version list is displayed. For details about how to set versions for different components, see **Uploading/Downloading Components on the Self-Hosted Repo Page**.

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

Step 4 Click a number in the **Versions** column. The version list of the software package is displayed.



Click a number in the **Version No.** column. The **Overview** and **Files** pages of the software package are displayed. In the **Files** tab, click a name in the **File Name** column. The path of the software package is displayed.

----End

5.4.3 Editing a Component

You can search for, download, delete, favorite, and unfavorite private components.

Searching for a Component

Step 1 Go to the self-hosted repo 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. You can search for information using either of the following methods:

- The default file name mode is as follows:
  a. Enter the keyword of the file name in the search box, and click to search for the component.
  b. In the search result list, click a file name to view the component details.

- Select **Checksums** mode.
  a. Click the drop-down list on the left of the search box and select **Checksums** (The default value is **File Name**).
  b. Or enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click to find the corresponding component.

----End

Downloading a Component

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be downloaded, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

----End

Deleting a Component

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be deleted, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

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

----End

Favoriting/Unfavoriting

Step 1 Go to the self-hosted repo page. In the left pane, locate the component to be favorited, and click its name.

If there are too many repositories or components, you can search for the desired one by referring to **Searching for a Component**.

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

When the icon changes to , click **My favorites** in the lower left corner of the page to view the list of favorited components. Click a value in the **path** column to go to the component details page.

----End

5.5 Recycle Bin 1.0

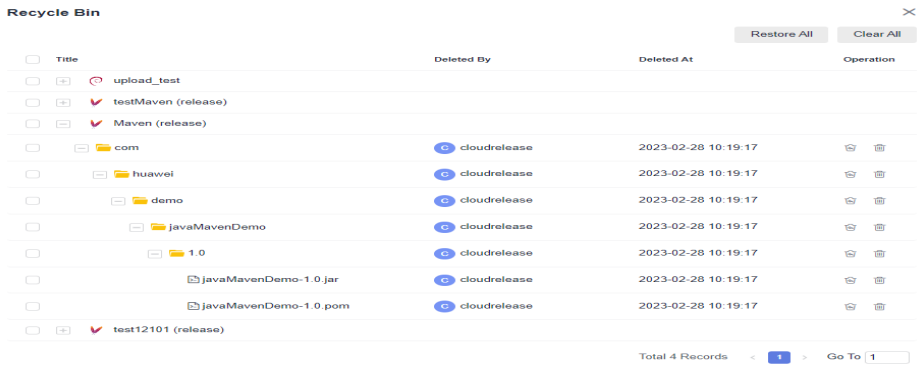
Repositories and components deleted from a self-hosted repo are moved to the recycle bin, where you can manage them.

Step 1 Go to the self-hosted repo page.

Step 2 Click **Recycle Bin**.

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

If both icons and are displayed in the **Operation** column, the repository in that row has been deleted. If only one icon is displayed, the repository still exists, but its components have been deleted. Click the repository name to view the deleted components.



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 them in batches.
	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 the 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
	Batch delete components	Go to the repository where the components to delete are located, select the components, and click **Clear** below the list to permanently delete them in batches.
	Clear all	Click **Clear All** in the upper right corner of the page 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 recovered anymore. Exercise caution when performing this operation.

----End

5.6 Connecting Self-Hosted Repos to Your Local Development Environment 1.0

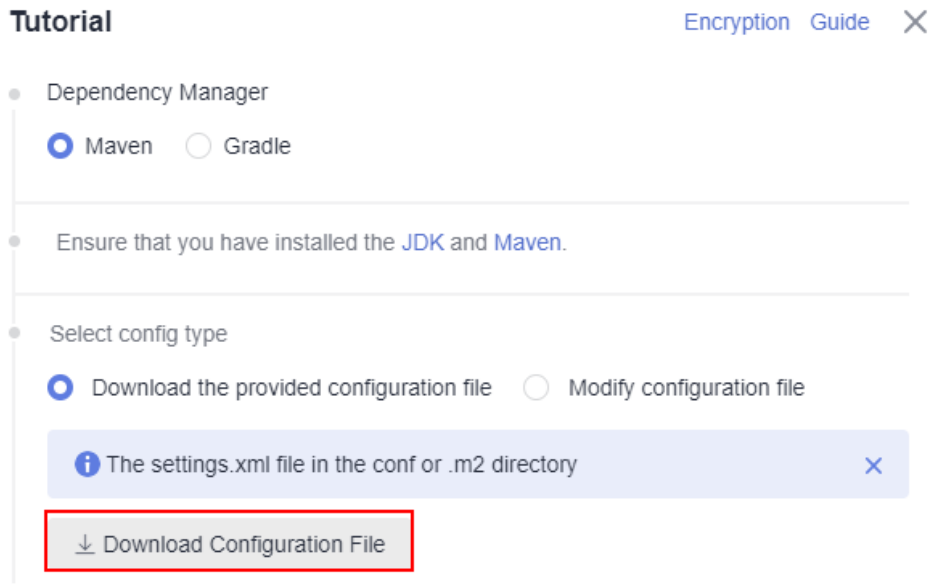
Downloading Configuration Files of Self-Hosted Repos

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 Go to the self-hosted repo page. In the left pane, click the name of the repository to be connected to your 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

Resetting Repository Password

You can download the configuration file to connect the self-hosted repo with your local development environment. The repository password is the password in the self-hosted repo configuration file. Therefore, after the repository password is reset, download the latest configuration file again to replace the original file.

Step 1 Go to the self-hosted repo page. Click ••• above the repository list on the left and choose **Reset Repository Password**.

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

----End

You can also upload and download private components through a client. The procedure is the same as that of self-hosted repo 2.0. For details, see **Uploading Components to Self-Hosted Repos on the Client** and **Downloading Components from Self-Hosted Repos on the Client**.

6 Whitelist for All Accounts

The whitelist for all accounts includes IP address segments and several access control settings. The whitelist restricts users' access, upload, and download permissions to enhance repository security. You can set the access permission for self-hosted repos by configuring the whitelist for all accounts.

Creating Whitelist for All Accounts

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, choose **Artifact** > **IP Address Whitelists**.

Step 3 Click **Add Address** in the upper right corner of the page.

Step 4 In the displayed dialog box, select **IP address** or **CIDR** and enter an IP address.

Table 6-1 IP address formats

Format	Description
IP address	This is the simplest IP address format. You can add the IP address of your PC to the whitelist, for example, 100.*.*.123.
CIDR	• If your server is on a LAN and uses the CIDR, you can add a 32-bit egress IP address of the LAN with a specified number of bits for the network prefix. • Requests from the same IP address are accepted if the network prefix is the same as the specified one.

Step 5 Select **I have read and agree to the Privacy Statement and CodeArts Service Statement.**, and click **OK**.

----End

