CodeArts Artifact

User Guide

Issue	01
Date	2025-08-06

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. CodeArts Artifact 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, 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
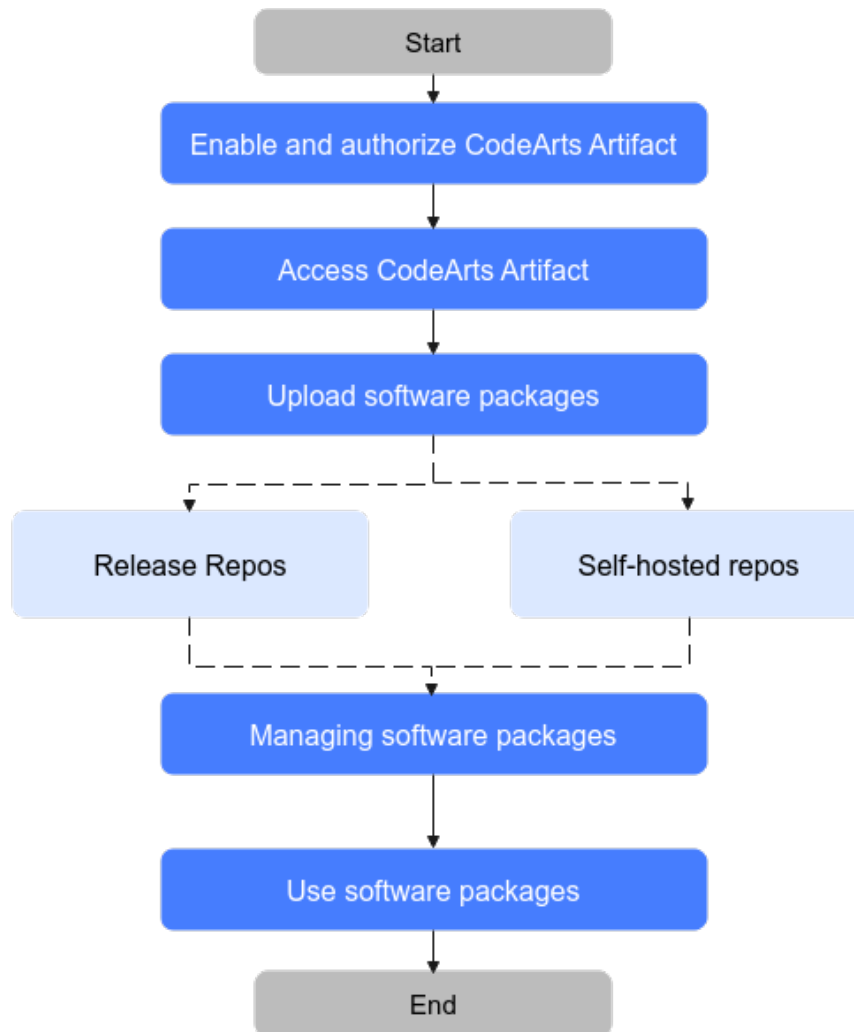
2 Purchasing CodeArts Artifact

Prerequisites

You have registered with Huawei Cloud and completed real-name authentication. If you do not have a HUAWEI ID yet, follow these steps to create one:

1. Visit the **Huawei Cloud official website**.

2. Click **Sign Up** and create your account as instructed.

   Once your account is created, the system automatically redirects you to your personal information page.

3. Complete individual or enterprise real-name authentication. For details, see **Real-Name Authentication**.

Purchasing CodeArts Artifact

For details, see **Purchasing CodeArts**.

3 Release Repos 2.0

3.1 Overview

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, and access permission control. It is a standardized way for enterprises to process all artifact types generated during software development.

Constraints

- 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 manually create Release Repos. Release Repos with the same name as the project is automatically generated when you create the project.

Preparations for Using CodeArts Artifact

- You have subscribed to CodeArts Artifact by referring to **Purchasing a CodeArts Package**.
- You have created a **project**.
- You have added CodeArts project members and assigned roles to them. For details, see **Adding Project Members**.

Accessing Release Repos

Step 1 **Log in to the Huawei Cloud console**.

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

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

Step 4 Click the **Release Repos** tab to view the list of repository names under the current tenant. You can then perform the following operations as required.
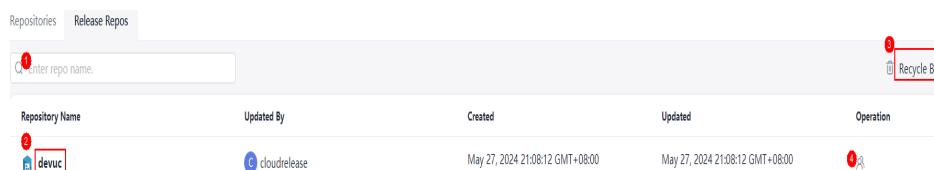
No.	Operation	Description
1	Search for repositories	Enter the project name in the search box to find the Release Repos of the project.
2	View folder details	Click any folder to view the list of archived software packages in folders. You can upload, download, and edit software packages or folders.
3	Manage 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**.

If you choose **Services** > **Artifact** > **Release Repos** from the top navigation bar, the project list is displayed. But you cannot upload files or create folders there. To perform such operations, click a project name to access it first.

Step 5 Click a repository name to go to the Release Repos page.

----End

Checking Basic Information

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

Step 2 View the repository name, package type, and description.

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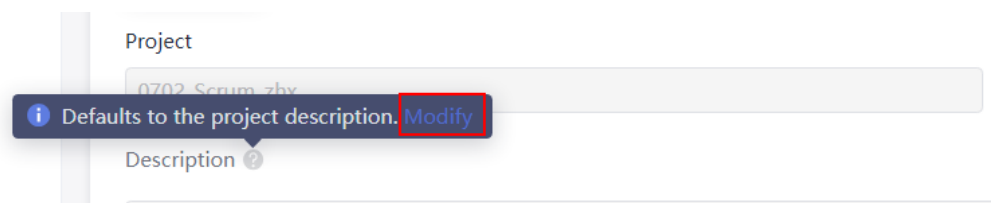
----End

3.2 Configuring Permissions

In CodeArts Artifact, different project roles have different permissions. You can view the default permissions of each project role on the permissions management page of CodeArts Artifact and adjust the role permissions as required.

Constraints

- By default, the project administrator role has all operation permissions and their permissions cannot be modified.
- **Custom roles** created in CodeArts Artifact do not have preset permissions. You can contact the project administrator to configure roles and permissions on corresponding resources.
- By default, the project administrator, project manager, and test manager roles have the **Privilege Config** permission. They can assign other roles permissions for the Release Repos. If other roles have the **Privilege Config** permission, they can continue to manage permissions for different roles in Release Repos.

Prerequisites

- You have created a **project**.
- You have added CodeArts project members and assigned roles to them. For details, see **Adding Project Members**.
- To assign permissions to other roles in the Release Repos, you must have the **Privilege Config** permission. By default, the project administrator, project manager, and test manager roles have this permission.

Setting Permissions

Step 1 A user with the **Privilege Config** permission accesses the **Release Repos**.

Step 2 Click ••• in the upper right corner and select **Set Project Permissions** from the drop-down list, as shown in the following figure.
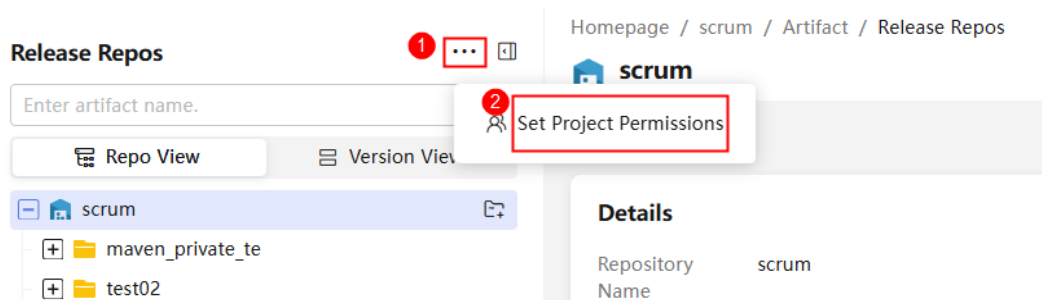
Step 3 In the **Roles** area, select the role to configure. On the right of the page, choose **CodeArts Artifact**. The following table lists the default permissions of each role in the Release Repos.

Table 3-1 Default permissions of each role

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	✘	✔	✔	✘	✔	✔	✔	✔	✔
Operation manager	✔	✔	✔	✔	✘	✔	✔	✘	✘
System engineer	✘	✔	✔	✘	✔	✔	✔	✘	✘
Committer	✘	✔	✔	✘	✔	✔	✔	✘	✘
Developer	✘	✔	✔	✘	✔	✔	✔	✘	✘
Tester	✘	✔	✘	✘	✘	✔	✔	✘	✘
Participant	✘	✘	✘	✘	✘	✘	✔	✘	✘
Viewer	✘	✘	✘	✘	✘	✘	✘	✘	✘
project administrator	✔	✔	✔	✔	✔	✔	✔	✔	✔

Step 4 Click **Edit** at the bottom of the page and select or deselect permissions as required.

Step 5 Click **Save**.

Each role can access the **Release Repos** and perform their authorized operations.

----End

3.3 Uploading a Package

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, support software development activities, enable efficient team collaboration, provide reliable software packages for deployment, and provide dependencies for build tasks.

Constraints

When **Multiple files** is selected, a maximum of 20 files can be uploaded at a time.

Prerequisites

- You have been added to a CodeArts project and assigned a role. For details, see **Adding Project Members**.
- You must have permissions to upload packages. For details about the default permissions of each role, see **Table 3-1**. For details about how to obtain permissions, see **Configuring Permissions**.

Uploading a Package

Step 1 **Access Release Repos**.

Step 2 Click **Upload** in the upper right corner of the page to manually upload local packages to Release Repos.

Alternatively, after selecting a folder, click **Upload** to manually upload local packages to the target folder.

**Step 3** In the **Upload** dialog box, set the parameters described in the following table.

Table 3-2 Parameters for uploading packages

Parameter	Description
Target Repository	Retain the default settings.
Version	Set the version number for the packages.
Upload Mode	Select **Single file** or **Multiple files** as the upload mode. **Single file** is selected by default here. When **Multiple files** is selected, a maximum of 20 files can be uploaded at a time.

Parameter	Description
Path	After you set the path name, a folder with that name is created in the **Repo View**, where the uploaded packages will be stored.
File	**CAUTION** **You are advised not to upload files containing sensitive information such as accounts and passwords to the Release Repos.** Select packages from your local PC to upload.

Step 4 Click **Upload**.

When the progress bar in the lower right corner of the page shows 100% (see the following figure), it indicates the upload is complete. If the upload fails, try again or contact customer service for technical support.

| Uploaded: 1	Failed: 0			
File Name	**File Size**	**Progress**		
text.txt	15.00 B	Uploaded	100%	

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

----End

Follow-Up Operations

After uploading the software packages, you can perform the operations listed in **Table 3-3**.

Table 3-3 Follow-up operations

Operation	Description
Create a subfolder	Click ⊞ next to the folder name where a software package is stored to add a subfolder.
Download a package or folder	Click ↓ next to a package or folder name to download it.
Edit a software package or folder	Click ✎ next to a software package or folder name to edit it.
Delete a software package or folder	Click 🗑 next to a software package or folder name to either permanently delete it or move it to the recycle bin.

Helpful Links

- CodeArts Artifact allows you to upload software packages either on the Release Repos page or through CodeArts Build. For details, see **Uploading a Package**.
- If files cannot be uploaded or directories cannot be created on the Release Repos homepage, see **Why Can't I Upload Files or Create Directories on the Release Repos Homepage?**
- If you choose to move a software package or folder to the recycle bin when deleting it, you can restore or permanently delete it from the recycle bin. For details, see **Recycle Bin**.

3.4 Managing Packages

Managing Packages in Repo View

Step 1 **Access Release Repos**.

Step 2 On the Release Repos page, click the **Repo View** tab.

Step 3 Click a package name. The details about the package are displayed. The details include the **General**, **Build Metadata**, and **Build Packages** tabs.

- **General**: displays the repository name, relative path, repository URL (for download), version, creator, creation time, modifier, modified time, size, and checksums.

  Click ✎ to change the version number of an uploaded package. By default, the release version of a software package archived by CodeArts Build is the version number set when the build task is executed, as shown in the following figure.
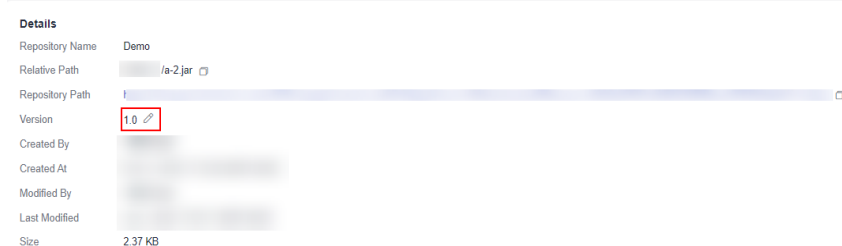
- **Build Metadata**: displays the build task, size, build number, builder, code repository, and code branch of the package. Click **Build Task** to link to the task in CodeArts Build.
- **Build Packages**: displays records of packages archived from build tasks. Click ⤓ to download the package.

----End

Managing Packages in Version View

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

Step 1 **Access Release Repos**.

Step 2 On the Release Repos page, click the **Version View** tab. The list of packages with version numbers is displayed. Release Repos stores packages of different versions with the same name in the same file.

Step 3 Click a file name in the **File Name** column. The latest version of the package is displayed.

Step 4 Click a number in the **Versions** column. The version list of the package is displayed.
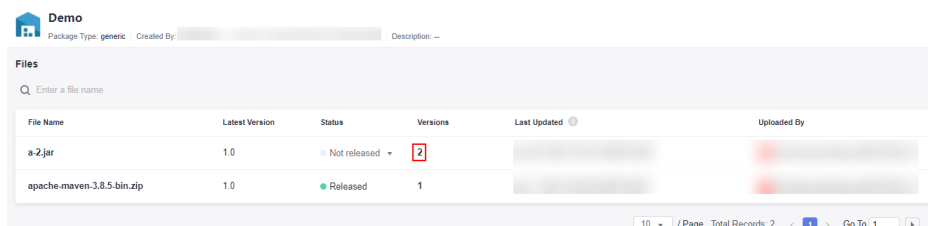
Step 5 Click a number in the **Version No.** column. The **Overview** and **Files** pages of the package are displayed. In the **Files** tab, click a name in the **File Name** column. The path of the package is displayed.

Step 6 Change the status if needed. By default, the package status is set to **Not released** after the version is configured.

- In **Files**, set the status to **Released**. This will update the 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**.

  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

Prerequisites: You have the **changePkgStatus** permission. For details about the default permissions of each role, see **Table 3-1**. For details about how to configure permissions, see **Configuring Permissions**.

Procedure

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).

Step 2

- 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.

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

----End

Downloading a Package

Prerequisites: You have the **download** permission. For details about the default permissions of each role, see **Table 3-1**. For details about how to configure permissions, see **Configuring Permissions**.

Procedure

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 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.
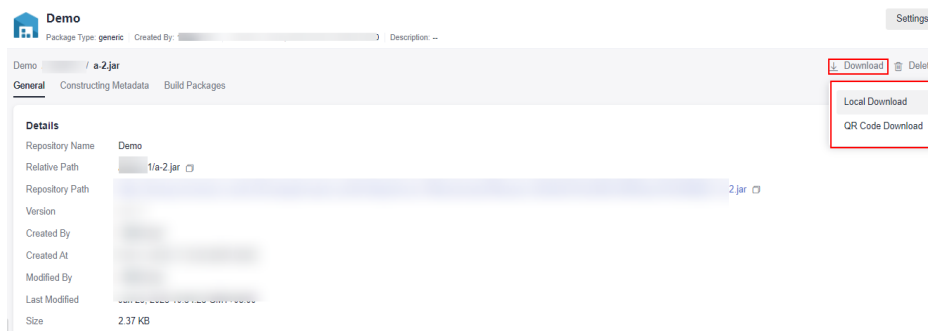
– **Local Download**: Download the package to the localhost.
– **QR Code Download**: Use your mobile phone to scan the QR code to download the package.

● Method 2: Hover over the package you want to download, and click ⭳ on the right.

----End

Searching for a Package

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 project or file name) in the search box above the list to search for the package whose name contains the keyword.

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

----End

Deleting a Package

Step 1 Go to the Release Repos page. In the left pane, locate the package you want to delete, and click its name.

If there are too many packages, you can find the desired package and directory by referring to **Searching for a Package**.

Step 2 Click **Delete** on the right of the package or directory.

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

----End

3.5 Clearing Policies

To keep your Release Repos organized and efficient, you can enable a scheduled auto-clearing policy to move expired files to the recycle bin or permanently delete outdated ones.

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.
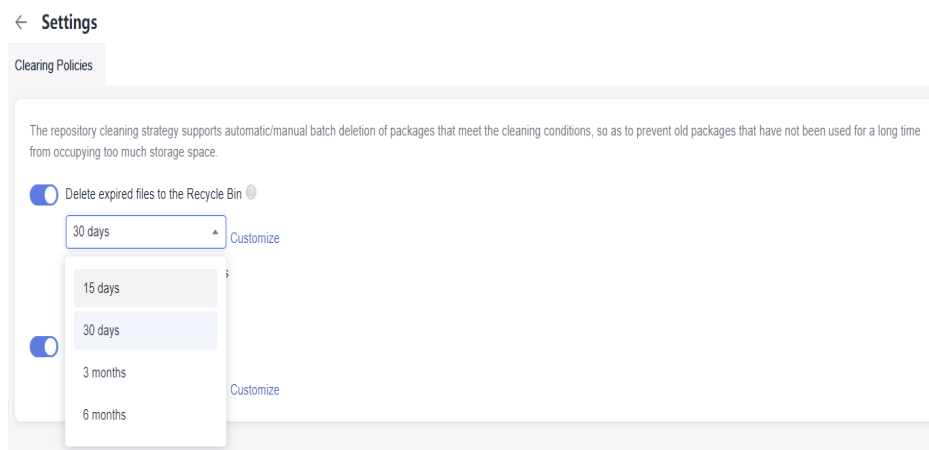
Default retention periods:

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

Figure 3-1 Setting clearing policies
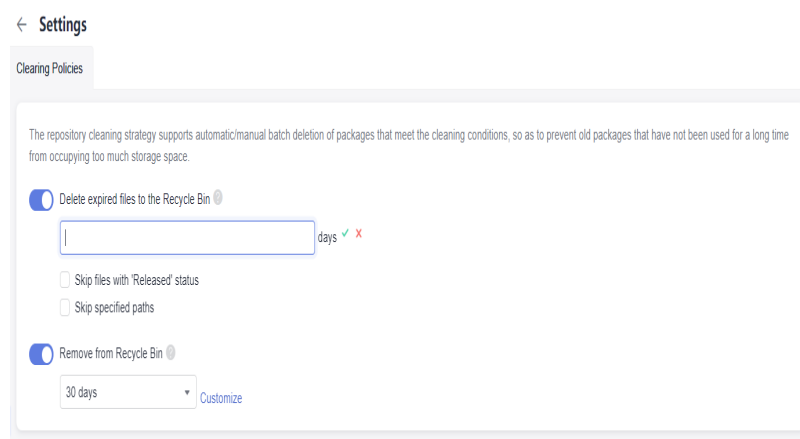
- You can also set a period. Click **Customize**, enter a number, and click √ to save the setting.

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 packages that match the file paths you set when clearing files.

▪ You can set multiple file paths, each starting with a slash (/) and separated by semicolons (;) but file path names cannot contain the project name. Example: **/test/folder1/;/1.0.0/test.txt**

▪ Spaces before and after the file path name will be automatically removed.

▪ The file path uses prefix matching. Ignore directories ending with a slash (/) and ignore files with a full file path. Otherwise, the ignored file may not match.

----End

3.6 Recycle Bin

Packages or folders deleted from Release Repos are moved to the recycle bin. You can restore or permanently delete them there. CodeArts Artifact includes both a global and a project-level recycle bin.

Global Recycle Bin

In the global recycle bin, you can manage 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** on the right of the page.
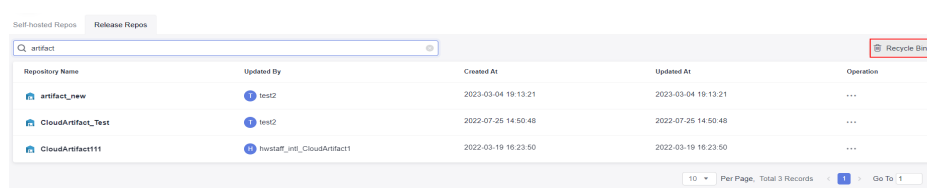
Step 3 Deleted files from different projects are listed on this page. Perform the following operations on a package or folder as required.
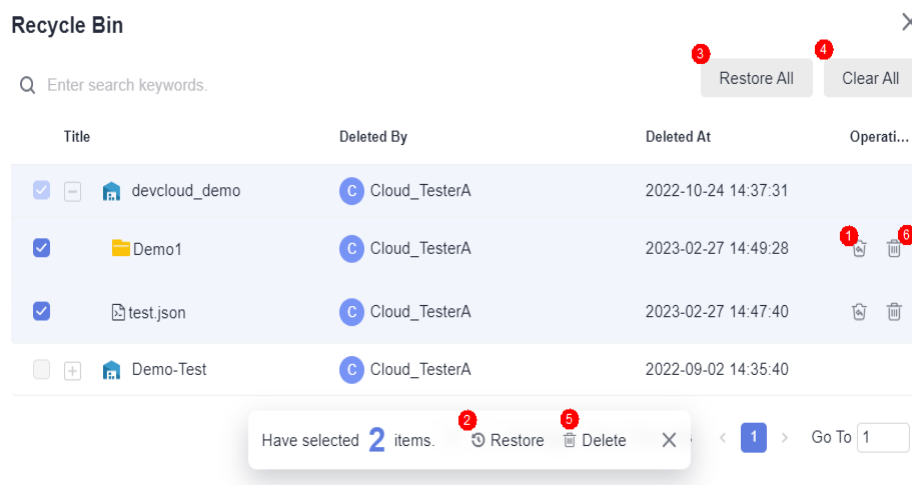

Table 3-4 Operations in the global recycle bin

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

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

● 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
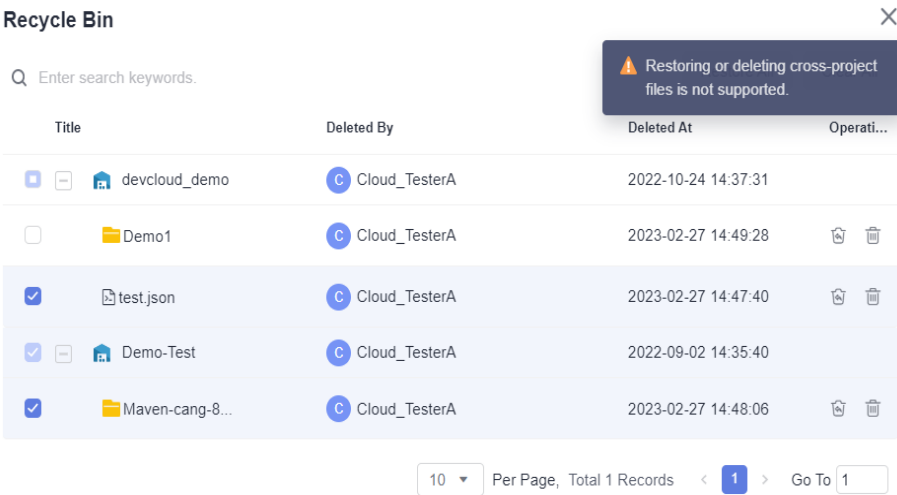
Project-Level Recycle Bin

You can manage deleted 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 Deleted files of this project are listed on this page. Perform the following operations on a package or folder as required. **Once you delete a package or folder from the recycle bin, it cannot be recovered. Exercise caution when performing this operation.**
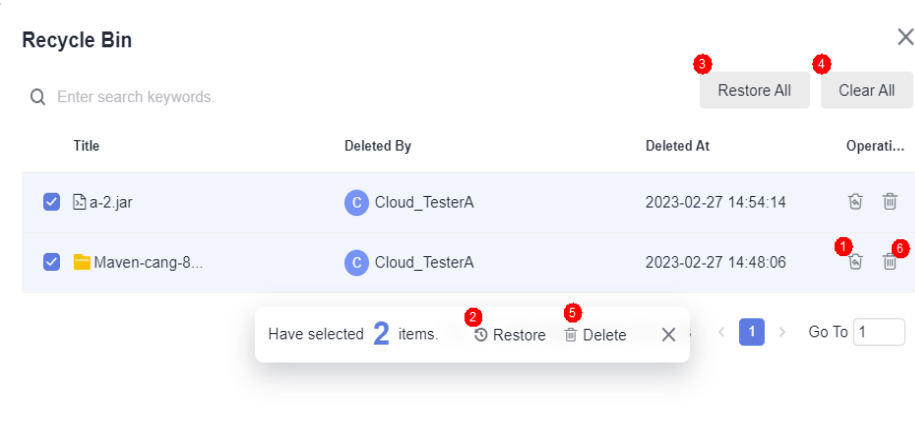
No.	Operation	Description
1	Individual restore	Click in the **Operation** column to restore the package or folder.

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

----End

Helpful Links

A file cannot be restored from the recycle bin page. A message indicating that duplicate file exists is displayed. For details, see

Can I Restore Files in the Recycle Bin of My Release Repos?

4 Self-Hosted Repos 2.0

4.1 Overview

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

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

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**.

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 All Types ▾ . 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

4.2 Creating a Repository

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: You can configure proxy sources in a virtual repository to connect with local third-party repositories. A virtual repository also offers local repository functions and provides a unified entry to make setup easier for you.

Constraints

Docker registry and CocoaPods repository currently are only available in the AP-Singapore region.

After a repository is created, its name and project cannot be changed.

Prerequisites

- You have created a **project**.
- You have been added to a CodeArts project and assigned a role. For details, see **Adding Project Members**.
- You have the **createRepository** permission. For details about the default permissions of each role, see **Table 4-3**. For details about how to configure permissions, see **Configuring Repository Permissions**.

Procedure

Step 1 **Access Self-hosted Repos**.

Step 2 Choose **Artifact** > **Self-hosted Repos** from the navigation pane. Click **Create Repository** in the upper right corner of the page. The **Create Repository** page is displayed.

Step 3 Configure the basic information by referring to **Table 4-1** and **Table 4-2**.

Table 4-1 Parameters for configuring a repository

Item	Mandatory	Description
Repository Type	Yes	You can choose **Local Repo** or **Virtual Repo**. - 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.

Item	Mandatory	Description
Repository Name	Yes	Enter up to 20 characters. Only letters, digits, underscores (_), hyphens (-), and periods (.) are supported. **After a repository is created, its name cannot be changed.**
Package Type	Yes	Local repository supports Maven, npm, RPM, PyPI, Go, Debian, Conan, NuGet, CocoaPods, Generic, OHPM, and Docker artifacts. Virtual repository supports Maven, npm, PyPI, Conan, OHPM, and Docker artifacts. The parameters vary based on the package type. Configure them by referring to **Table 4-2**. Docker registry and CocoaPods repository currently are only available in the AP-Singapore region.
Project	Yes	Choose a project to associate with the new repository. **After the settings are complete, the project to which the repository belongs cannot be changed.**
Description	No	Enter up to 200 characters.

Table 4-2 Parameters of different package types

Package Type	Parameter	Mandatory	Description
Maven	Associated Projects	Yes	Select the name of the project to be associated from the drop-down list.
	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. By default, all artifact paths are allowed. During builds, only Maven files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.

Pack age Type	Param eter	Mand atory	Description
	Exclud e Patter ns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, Maven files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo.
			For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed.
			You can click **+** to add multiple exclusion rules.
	Versio n Policy	Yes	Two repositories will be generated: **Release** and **Snapshot**. The **Release** repository is selected by default and the **Snapshot** repository can also be selected as required.
			● The **Release** repository is used to store release versions with stable functions that will no longer be updated.
			● The **Snapshot** repository is used to store development versions with unstable functions in the development stage.
npm	Includ e Patter ns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only npm files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo.
			For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed.
			You can click **+** to add multiple inclusion rules.
	Exclud e Patter ns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, npm files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo.
			For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed.
			You can click **+** to add multiple exclusion rules.
RPM	Includ e Patter ns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only RPM binary files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo.
			For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed.
			You can click **+** to add multiple inclusion rules.

Package Type	Parameter	Mandatory	Description
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, RPM binary files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo. You can click **+** to add multiple exclusion rules.
PyPI	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only PyPI dependencies in which the **name** value in the **setup.py** file matches the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, PyPI dependencies in which the **name** value in the **setup.py** file matches the preceding path prefix cannot be uploaded to the self-hosted repo. You can click **+** to add multiple exclusion rules.
Go	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only Go files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, Go files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo. You can click **+** to add multiple exclusion rules.
Conan	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only Conan files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.

Pack age Type	Param eter	Mand atory	Description
	Exclud e Patter ns	No	Enter rules to specify paths not allowed for artifact uploads. Conan files whose paths start with the preceding path prefix cannot be uploaded from a local client to the self-hosted repo. You can click **+** to add multiple exclusion rules.
NuGe t	Includ e Patter ns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only NuGet files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclud e Patter ns	No	Enter rules to specify paths not allowed for artifact uploads. NuGet files whose paths start with the preceding path prefix cannot be uploaded from a local client to the self-hosted repo. You can click **+** to add multiple exclusion rules.
Coco aPods	Includ e Patter ns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only CocoaPods files whose paths start with the preceding path prefix can be uploaded to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclud e Patter ns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, CocoaPods files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo. You can click **+** to add multiple exclusion rules.
Gener ic	Includ e Patter ns	No	Enter path rules not allowed for artifact uploads. You can click **+** to add multiple paths. During builds, only Generic files whose paths start with the preceding path prefix can be pushed to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.

Package Type	Parameter	Mandatory	Description
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, Generic files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo. During builds, Generic files whose paths start with the preceding path prefix cannot be pushed to the self-hosted repo. You can click **+** to add multiple exclusion rules.
OHPM	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only OHPM files whose paths start with the preceding path prefix can be pushed to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. During builds, OHPM files whose paths start with the preceding path prefix cannot be uploaded to the self-hosted repo. You can click **+** to add multiple exclusion rules.
Docker	Include Patterns	No	Enter rules to specify paths allowed for artifact uploads. During builds, only image files whose paths start with the preceding path prefix can be pushed to the self-hosted repo. For example, if you add **demo**, that is, **demo/**/***, all paths starting with **demo/** are allowed. You can click **+** to add multiple inclusion rules.
	Exclude Patterns	No	Enter rules to specify paths not allowed for artifact uploads. Image files whose paths start with the preceding path prefix cannot be pushed to the self-hosted repo. You can click **+** to add multiple exclusion rules.

Step 4 Click **OK**. The new repository displays in the repository list on the left.

----End

4.3 Configuring Repository Permissions

By default, new users in self-hosted repos have no permissions. You need to add them to a project and assign roles to them. Each role has different permissions.

You can view role permissions on the CodeArts Artifact permissions page and edit user permissions as needed.

Constraints

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

Prerequisites

- You have created a **project**.
- You have added CodeArts project members and assigned roles to them. For details, see **Adding Project Members**.
- To assign permissions to other roles in the self-hosted repo, you must have the **Privilege Config** permission. By default, the project administrator, project manager, and test manager roles have this permission.

Configuring Repository Permissions

Step 1 A user with the permission configuration privilege accesses the **self-hosted repo**.

Step 2 Choose **Settings** > **General** > **Permissions** from the navigation pane. The **Permissions** page is displayed.

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

Table 4-3 Project-level permissions

Role/ Operation	Create a repository	Edit a repository	Delete a repository	Restore	Permanently delete	Restore all	Clear all	Upload a package	Download/ View a package	Delete/ Redeploy uploaded packages
Project manager	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅

Role/ Operation	Create a repository	Edit a repository	Delete a repository	Restore	Permanently delete	Restore all	Clear all	Upload a package	Download/ View a package	Delete/ Redeploy uploaded packages
Product manager	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗
Test manager	✗	✓	✗	✓	✗	✓	✗	✓	✓	✗
Operation manager	✗	✓	✗	✓	✓	✓	✓	✓	✓	✗
System engineer	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗
Committer	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗
Developer	✗	✓	✗	✓	✓	✗	✓	✓	✓	✗
Tester	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗
Participant	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Viewer	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Project administrator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

- Tenant administrators (IAM user with the **Tenant Administrator** permissions) can manage all projects of a tenant. Project creators who are not tenant administrators have the permissions listed in the preceding table.
- IAM users created without being added to any groups do not have permissions. Administrators can assign permissions to these users on the IAM

console. Then users can use cloud resources in the account based on the assigned permissions.

- Custom roles do not have preset permissions. You can contact the administrator to grant permissions for the resources needed for your role.

----End

Managing Repository Permissions

CodeArts Artifact allows you to configure permissions on all self-hosted repos in a project. For details, see **Configuring Repository Permissions**.

You can also configure permissions for a single self-hosted repo.

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 roles in the current project are displayed.

Step 4 In the **Roles**, select or deselect permission items of the target role, and click **Save**.

- By default, created self-hosted repos connect with the role permission of **Permissions** in **Settings** in the project. Any changes made to these role permissions in **Permissions** will be synced to the repo's permissions.

- If you do not change the repository permission of a role in the self-hosted repo, any changes made on the **Permissions** page will be synchronized to the repository permission of the self-hosted repo as well.

- If you change the repository permission of a role directly in the self-hosted repo, any changes made on the **Permissions** page will not be synchronized to the repository permission of the self-hosted repo. You need to change the permission of the role in the repo itself.

----End

4.4 Managing a Repository

4.4.1 Viewing, Configuring, and Deleting a Repository

Viewing a Repository

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 desired repository name to go to its details page.

General, **Repository Permissions**, **Resources**, and **Operation Logs** tabs are displayed.

- **General**: displays the repository name, repository type, repository URL, relative path, creator, creation time, modifier, modification time, artifact count, and artifact size.
- **Repository Permissions**: displays the permission scope (repository and package) of the repository and the corresponding permissions. This page is view-only. To edit permissions, choose **Settings** > **Repository Permissions**. For details, see **Managing Repository Permissions**.
- **Resources**: displays dynamic statistics on **File Counts** and **Storage Capacity (Byte)** for uploaded artifacts in the repository.
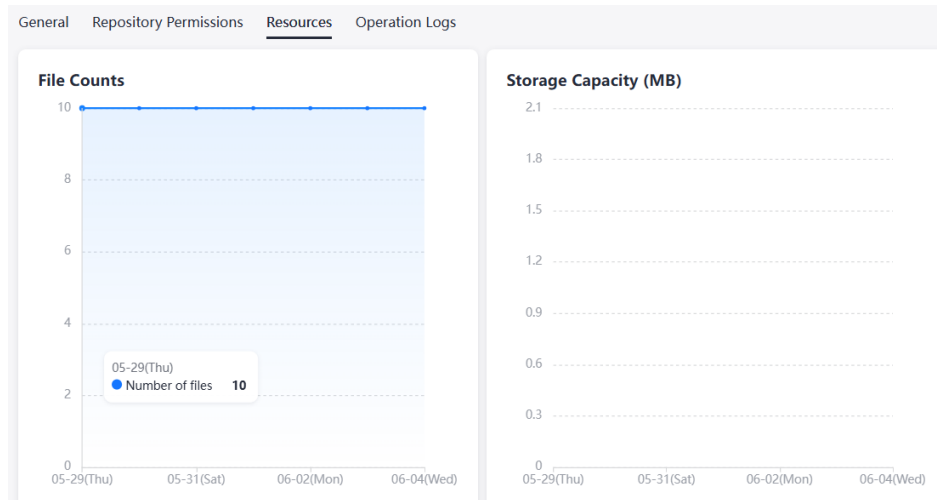
- **Operation Logs**: displays the operation history of uploading, deleting, and restoring data from the recycle bin in the repository.

----End

Obtaining Repository URLs

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

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

Step 2 The repository URL is displayed on the **General** page. Click ⧉ to obtain the URL.
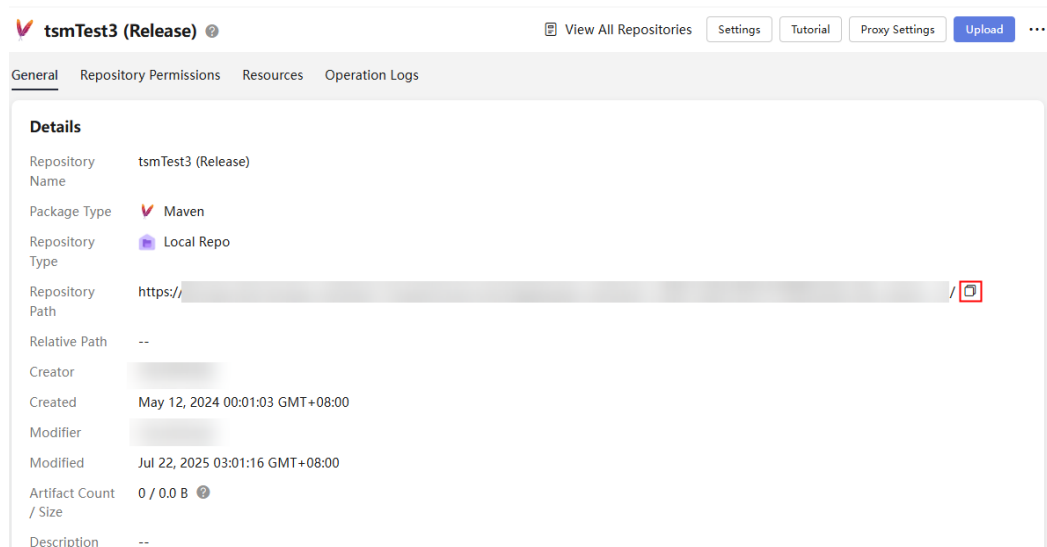
----End

Configuring a Repository

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 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, you can click ＋ to add path rules for Maven, npm, Go, PyPI, RPM, Docker, CocoaPods, OHPM, and Conan repositories.

To delete path rules, click 🗑.

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

----End

Deleting a Repository

You can delete repositories, 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. Check that the deleted repository is no longer displayed in the repository list in the left pane.

----End

4.4.2 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 4-1 Configuring deployment policies
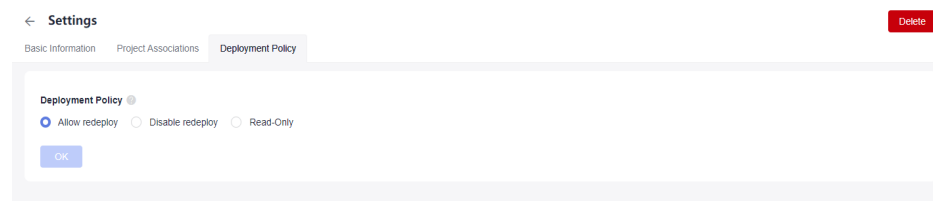
- **Allow redeploy** (enabled by default): Artifacts in the same path can be uploaded, replacing the original package.
- **Disable redeploy**: Artifacts in the same path cannot be uploaded.
- **Read-only**: Artifacts cannot be uploaded, updated, or deleted. You can download an uploaded artifact.

Step 3 Settings are automatically saved by the system.

----End

4.4.3 Configuring Clearing Policies for a Maven Repository

The clearing policy allows you to clear artifacts in batches. This helps optimize storage, keep artifacts organized, and ensure 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. During each build, Maven checks for new snapshots in the remote repository. You can set limits on how many snapshots to retain, as well as automatically clear out expired snapshots.

Snapshot Clearing 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

4.4.4 Associating Maven Repositories 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**.

When the system displays a message indicating that the operation is successful, the value of **Associated Repositories** for the project is updated based on 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)**.

4.4.5 Configuring and Adding a Proxy in a Virtual Repository

You can configure proxy sources in the virtual repository to connect with local third-party repositories. A virtual repository also offers local repository functions and provides a single entry to make setup easier.

Configuring a Proxy in a Virtual Repository

Custom proxy repositories allow you to add proxies for open-source and third-party dependency repositories. After files are downloaded from proxy repositories, they can be cached to CodeArts Artifact, speeding up the third-party dependency download speed to that of the local repository.

The proxy settings are available for Maven, npm, Docker, and PyPI in repositories.

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** tab and click **Create Proxy** at the upper right corner of the page.

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

- The proxy repository 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   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 on the **Proxy Settings** page of the corresponding repository and return to this page to delete the proxy source.

----End

Adding a Proxy in a Virtual Repository

Step 1 Create a virtual repository by referring to **Step 1**.

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

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

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

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

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

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

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

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

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

- Click ✎ in the **Operation** column to change the proxy name, proxy username, and proxy password. You cannot edit the proxy source of the Huawei local repository.

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

----End

4.5 Uploading/Downloading Packages on the Self-Hosted Repo Page

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

Uploading a Package

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 package 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 package type is described below.

- The Conan package cannot be uploaded on the page.

- 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 Artifacts

- 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 package.

- 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 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 package. **Version** indicates the version of the 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 Artifact

POM and GAV upload modes are supported.

Table 4-4 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 4-2 Uploading a package 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>
```

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 4-3 Uploading a package in GAV mode

npm Packages

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

The npm 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 Package

npm 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.

When uploading a package, ensure that the **PackageName** starts with a path in the path list added during repository creation. For details, see **Table 4-2** in the help guide.

Example:

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

When uploading the package to the registry, 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 package in .tgz format in the repository list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Package

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 package, 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 Package

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 package 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 Package

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 packages.

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 package is to be uploaded.

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

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

Figure 4-4 Uploading a package

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 package name directory. You can use Yum to install the package.

----End

Uploading a Debian Package

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

Parameter	Description
Distribution	Release version of the package
Component	Name of the package
Architecture	Package architecture

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

Upload ⓘ Help   ✕

* Distribution ⓘ

> You can enter multiple values separated by semicolons (;).

* Component ⓘ

> You can enter multiple values separated by semicolons (;).

* Architecture ⓘ

> You can enter multiple values separated by semicolons (;).

Path ⓘ

>

* File

> --select--   ⋮

          Upload   Cancel

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.

⊟ Ⓒ debian-test
    ⊟ 📁 dists
        ⊞ 📁 trusty
        📄 a2jmidid_8_dfsg0-1_amd64.deb

Uploading a NuGet Package

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 package is to be uploaded.

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

Step 3   View the package that is successfully uploaded in the repository list.

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

package stores components.

----End

Uploading a Docker Image

Step 1 Go to the self-hosted repo page. In the left pane, click the target Docker registry to which the private image 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 image 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 Package

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

Uploading a Generic Package

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

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

Parameter	Description
Upload Mode	Select **Single file** or **Multiple files**. **Single file** is selected by default here. When **Multiple files** is selected, a maximum of 20 files can be uploaded at a time.
Path	After you set the path name, a folder with that name is created in the **Repo View**, where the uploaded package will be stored.
File	Select the Generic package 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 an OHPM Package

OHPM packages

OpenHarmony Package Manager (OHPM) is an ArkTS package management tool. OHPM manages packages, which are stored and organized in the OHPM repository.

The OHPM package contains the Harmony Archive (HAR) package and Harmony Shared Package (HSP) package.

- HAR: Static shared package, which is reused in the build phase. It is mainly used as a second-party or third-party library for other applications to depend on. It includes resource files, *.so files, and metadata files.

- HSP: Dynamic shared package, which is reused in the running phase It provides shared code and resources to improve code reusability and maintainability.

The metadata file, **oh-package.json5**, in the HAR package describes the component, including its name, version, and dependency information. The **ohpm install** command automatically reads the dependencies in **oh-package.json5** and downloads them.

Example: **oh-package.json5**

```
{
  "name": "@scope/demo",              // Component name
  "version": "0.0.1",             // Version
  "author": "artifact",                // Author
  "license": "Apache-2.0",              // License agreement
  "main": "index.ts",               // Entry file
  "description": "basic information.",     // Description
  "dependencies": {                  // Dependency
     "@scope/demo1": "1.0.0",
     "@scope/demo2": "file:./demo2.har"
  },
  "devDependencies": {                  // Development dependency
     "@scope/demo3": "1.0.0",
     "@scope/demo4": "1.0.0"
```

```
    },
    "metadata": {                        // Metadata information
        "sourceRoots": ["./src/main"]
    }
}
```

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 OHPM 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 OHPM package

You can upload OHPM packages in .har or .hsp format to the self-hosted repo. The procedure is as follows:

Step 1 Go to the self-hosted repo page. In the left pane, click the target OHPM repository to which the private package 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 package name. The value must be the same as that of **name** in the **oh-package.json5** file.
	Ensure that the **PackageName** starts with a path in the path list added during repository creation. For details, see **Table 4-2** in the help guide.
	Example:
	The path **@test** is added when you create an OHPM repository.
	When uploading the package to the repository, make sure that the **PackageName** starts with **@test**. If a path outside the path list is used, for example, **@ohos**, the upload will fail.
Version	Enter the version number. The value must be the same as that of **version** in the **oh-package.json5** file.
File	Select the OHPM 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.

You can find the package in the repository list and the corresponding metadata is generated in the **.ohpm** directory.

----End

Downloading a Package

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

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

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

----End

4.6 Uploading/Downloading Pakckages on the Client

4.6.1 Connecting Self-Hosted Repos 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 4-5 Downloading the configuration file
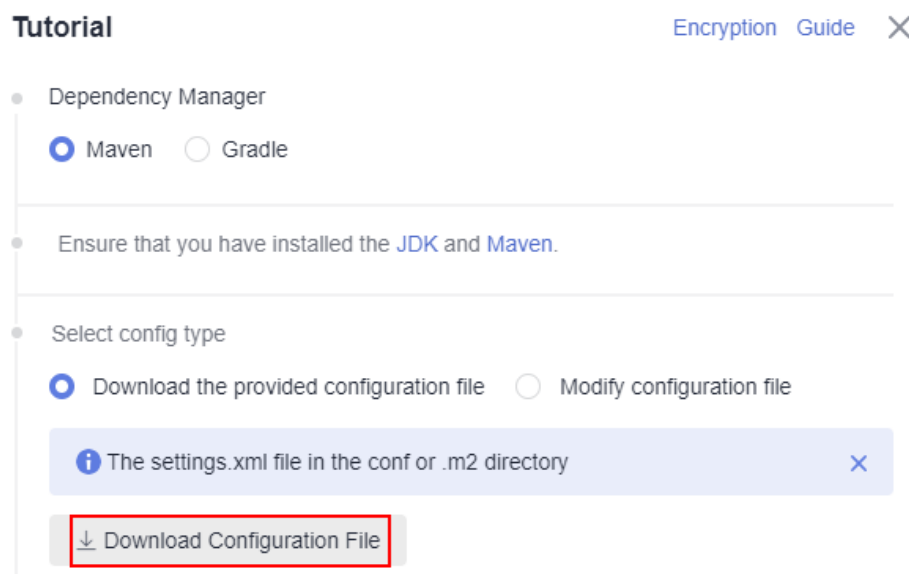
Step 4 Copy the downloaded file to the corresponding Maven directory based on the instructions in the information dialog box.

----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

4.6.2 Uploading Packages to Self-Hosted Repos from the Client

The repository password for a self-hosted repo is account-specific. If you are prompted for a password when using different accounts, download the configuration file from the repository's Tutorial tab to obtain it.

Uploading a Maven Artifact from 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 command (example) 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.

- 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 artifact.

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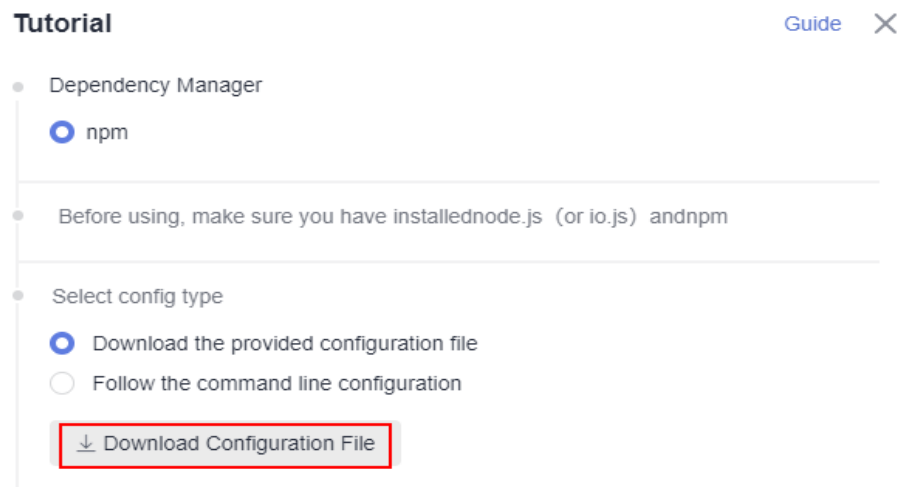
   2. Copy the file to the user directory. In Linux, the path is **~/.npmrc** (C:\Users\<*UserName*>\.npmrc).

   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. Upload the npm package to the repository:

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

```
npm notice === Tarball Details ===
npm notice name:          @test/demo
npm notice version:       1.0.0
npm notice package size:  8.7 MB
npm notice unpacked size: 10.6 MB
npm notice shasum:
npm notice integrity:
npm notice total files:   102
npm notice
+ @test/demo@1.0.0
```

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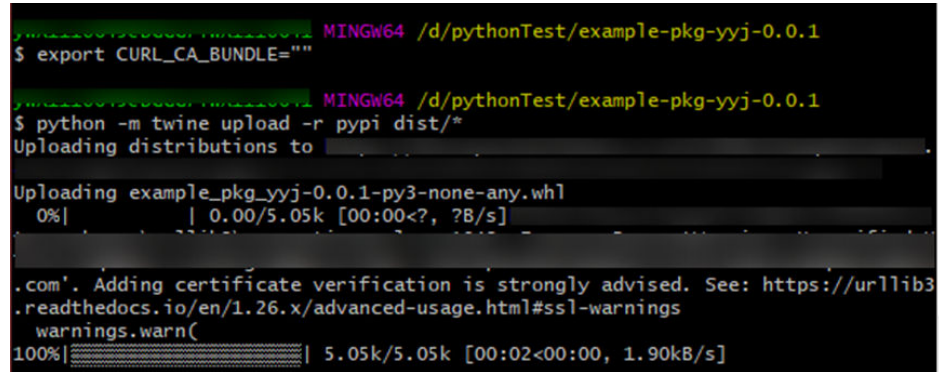
2. Copy the file to the user directory. In Linux, the path is **~/.pypirc**. In Windows, the path is **C:\Users**<*UserName*>**\.pypirc**.

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. 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 on Windows) to set environment variables to skip certificate verification (skip this step if you are using Twine 3.8.0 or earlier and Requests 2.27 or earlier.)

```
export CURL_CA_BUNDLE=""
```

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 Package from 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 packages through Go modules. 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 package into a ZIP package.
     ```
     zip -D -r [package name] [package root directory]
     ```

  d. Upload the 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 package into a ZIP package.

   Run the following command to go to the upper-level directory of the root directory where the ZIP package is located:
   ```
   cd ~
   ```

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

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

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

```
go.mod

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

a. Create a source folder in the working directory.

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

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

b. Copy the code source to the source folder.

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

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

c. Compress the package into a ZIP package.

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

```
cd ~
```

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

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

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

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

```
go.mod

    1    module example.com/demo
```

a. Create a source folder in the working directory.

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

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

b. Copy the code 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 package into a ZIP package.

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

```
cd ~
```

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

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

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

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

Uploading a Debian Package from the Client

Prerequisites

- You have logged in to the CodeArts homepage.

- The client tool is APT. Ensure that the Ubuntu or Debian system has been installed. The APT repository source configuration file is **sources.list** in **/etc/apt/** of the user root directory.

Procedure

Step 1 Before downloading the package, save the obtained public key information to the **public.asc** file and add the information to the system key list.

```
gpg --import <PUBLIC_KEY>
gpg --list-signatures
gpg --export --armor <SIG_ID> | apt-key add -
```

Step 2 Upload a package.

```
curl -k -u "<USERNAME>:<PASSWORD>" -X PUT "https://<DeHelmbian repository URL>/
<DEBIAN_PACKAGE_NAME>;deb.distribution=<DISTRIBUTION>;deb.component=<COMPONENT>;deb.archite
cture=<ARCHITECTURE>" -T <PATH_TO_FILE>
```

----End

Upload an RPM Package from 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, upload an RPM package:
```
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 package to be uploaded. The **hello-0.17.2-54.x86_64.rpm** package is used as an example.

– *component*: software name, for example, **hello**.

– *version*: software version, for example, **0.17.2**.

– *localFile*: RPM package, 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 package.

----End

Uploading a Conan Package from 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.

Procedure

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
```

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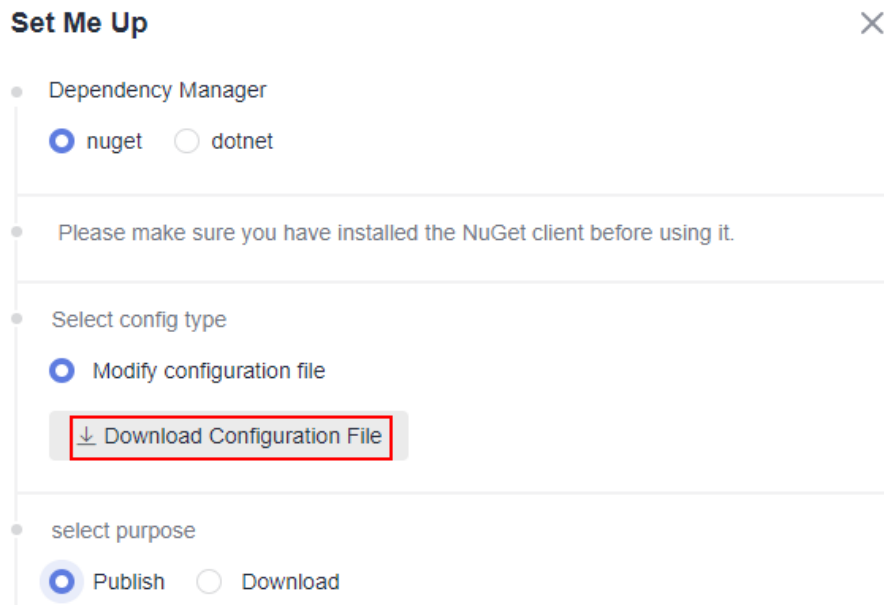
```
$ conan search hello/0.1@demo/testing -r=my_local_server
```

----End

Uploading a NuGet Package from 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 Package from the Client

Ensure that you have installed the .NET.

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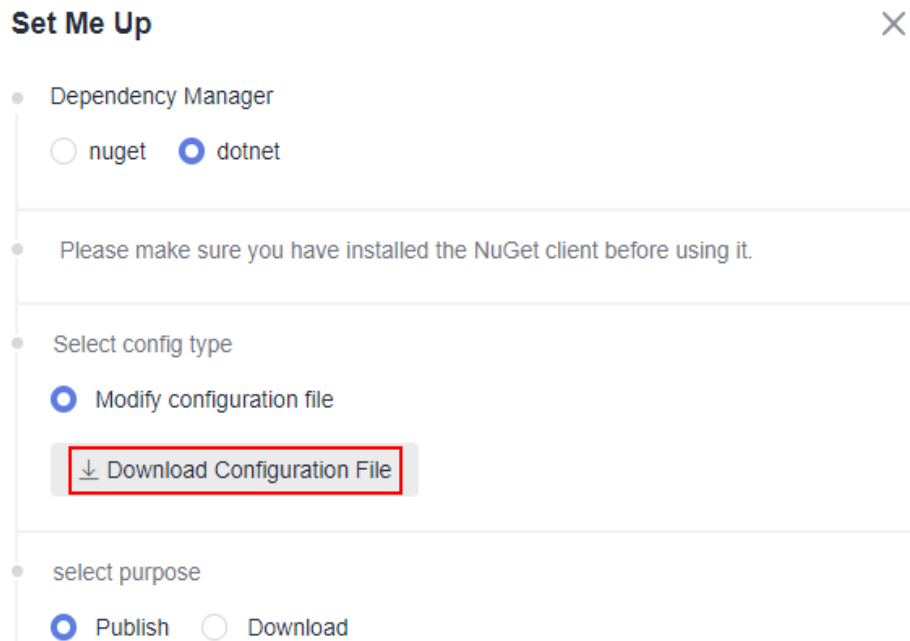
  2. In Windows, you need to use PowerShell to add the certificate trust.

  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 Image from the Client

Prerequisites

- You have installed the Docker client.
- You have created a Docker registry in the self-hosted repo.

Procedure

Step 1 Select the target Docker registry 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 registry:

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

url: repository URL.

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:

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

image_name1: local image name.

image_version1: local image version.

url: repository URL, as shown in the following figure.
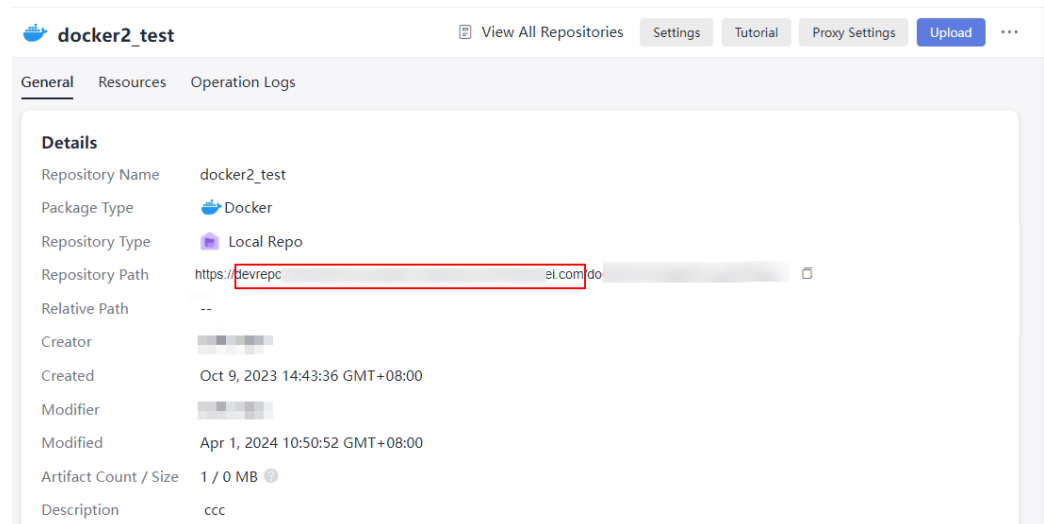
image_name2: You can name the uploaded image. The image name is displayed in the image list of the Docker registry.

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 image to the self-hosted repo:

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

url: repository URL, as shown in the following figure.
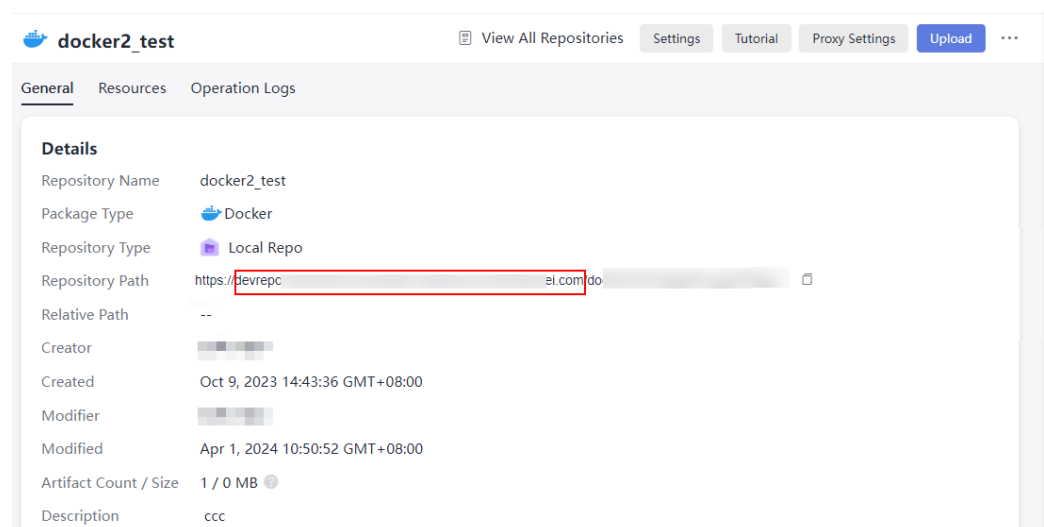
image_name: Enter **image_name2** in **Step 5**.

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

Step 7 Check the uploaded image in the Docker registry.

----End

Uploading a CocoaPods Package from 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 packages 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 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 URL of the CocoaPods repository.

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

PATH_TO_FILE: local path of the package to be uploaded.

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

Step 5 Check the uploaded package in the CocoaPods repository.

----End

Following the command line configuration to upload CocoaPods packages 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 Check whether the Rudy client has been installed:

```
rudy -v
```

Step 4 Install the cocoapods-art plug-in:

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

Step 5 Add the self-hosted repo to your CocoaPods client:

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

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

url: repository URL of the CocoaPods repository.

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

Step 7 Upload the local package 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 URL of the CocoaPods repository.

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

PATH_TO_FILE: local path of the package to be uploaded.

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

Step 8 Check the uploaded package in the CocoaPods repository.

----End

Uploading a Generic Package from the Client

Prerequisites

You have logged in to the CodeArts homepage.

Procedure

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

Step 2 In the displayed dialog box, click **Download Configuration File** to download the **generic.txt** file.

Tutorial   ✕

● Config type

[↓ Download Guide File]

● Select purpose

◉ Upload   ◯ Download

1.Upload a package

ℹ Obtain the username and password from the downloaded   ✕
configuration guide file.

```
1   curl -k -u "<USERNAME>:<PASSWORD>" -X PUT https://devrepo.devclo
```

Step 3 Upload the Generic package to the repository.

curl -k -u "*{{username}}:{{password}}*" -X PUT *{{repo_url}}*/*{{filePath}}* -T *{{localFile}}*

file path: Path (including the name) of the Generic repository to be uploaded.

localFile: Path (including the name) of the local Generic package.

Obtain the values of *username*, *password*, and *repo_url* from the downloaded
generic.txt file in **Step 2**, as shown in the following figure.

```
generic.txt⊠
 1  repo_url = https://devrepo.devcloud.cn-north-7.ulanqab.huawei.com/artgalaxy/cn-north-7_09d2ca2f5080d5b60f51c00ae5bad0a0_generic_12
 2  username = ████████                   ████████████████████████████
 3  password = ████████
```

----End

Uploading an OHPM Package from the Client

OHPM packages

OpenHarmony Package Manager (OHPM) is an ArkTS package management tool.
OHPM manages packages, which are stored and organized in the OHPM
repository.

The OHPM package contains the Harmony Archive (HAR) package and Harmony
Shared Package (HSP) package.

● HAR: Static shared package, which is reused in the build phase. It is mainly
used as a second-party or third-party library for other applications to depend
on. It includes resource files, *.so files, and metadata files.

● HSP: Dynamic shared package, which is reused in the running phase It
provides shared code and resources to improve code reusability and
maintainability.

The metadata file, **oh-package.json5**, in the HAR package describes the
component, including its name, version, and dependency information. The **ohpm**

---

install command automatically reads the dependencies in **oh-package.json5** and downloads them.

Example: **oh-package.json5**

```
{
    "name": "@scope/demo",              // Component name
    "name": "0.0.1",              // Version
    "author": "artifact",                // Author
    "license": "Apache-2.0",              // License agreement
    "main": "index.ts",              // Entry file
    "description": "basic information.",      // Description
    "dependencies": {              // Dependency
        "@scope/demo1": "1.0.0",
        "@scope/demo2": "file:./demo2.har"
    },
    "devDependencies": {                  // Development dependency
        "@scope/demo3": "1.0.0",
        "@scope/demo4": "1.0.0"
    },
    "metadata": {                  // Metadata information
        "sourceRoots": ["./src/main"]
    }
}
```

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 OHPM 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"
}
```

Prerequisites

- You have logged in to the CodeArts homepage.
- The client tool is npm, and the dependency management tool is OHPM. Ensure that node.js 16.x or later has been installed.

Procedure

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

Step 2 In the displayed dialog box, download the **ohpmrc** file and rename it **.ohpmrc**.

Step 3 Copy the file to the user directory. The path is as follows:

Linux: **~/.ohpmrc**

Windows: **C:\Users**<*UserName*>**\.ohpm\.ohpmrc**

Step 4 Upload the OHPM package to the repository.

ohpm publish *{file path}\{package name}*

file path: path of the local OHPM package.

package name: name of the local OHPM package.

----End

4.6.3 Downloading Packages from Self-Hosted Repos to the Client

The repository password for a self-hosted repo is account-specific. If you are prompted for a password when using different accounts, download the configuration file from the repository's Tutorial tab to obtain it.

Downloading a Maven Artifact to the Client

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

Step 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.
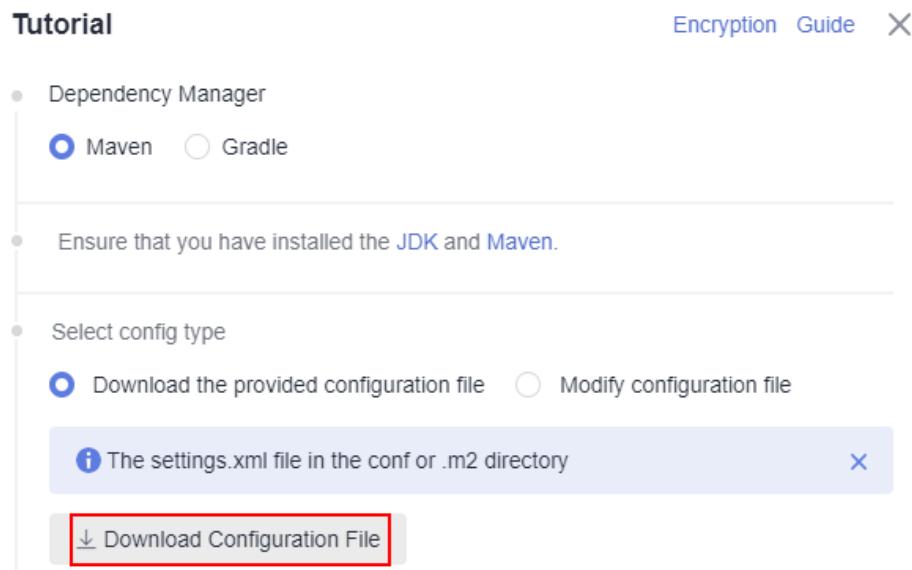
Step 2 Download the client.

```
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
```


----End

Downloading an npm Package to the Client

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

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

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

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

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


----End

Downloading a PyPI Package to the Client

The client tools are Python and PIP. Ensure that Python and PIP have been installed.

Step 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)

Step 2 Install Python:

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


----End

Downloading a Go Package to 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:

Step 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.

Step 2 Add the certificates to the root certificate trust list.

Step 3 Run the go commands to download the dependency:

```
##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}}
```
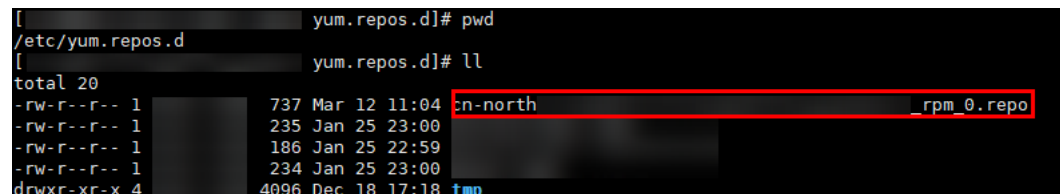
##b. The ZIP package does not contain **go.mod** or the first line in **go.mod** does not end with **/vN**.
go get -v {moduleName}}@{{version}}+incompatible

----End

Downloading an RPM Package to the Client

The following section uses the RPM package from **Uploading an RPM Package to the Client** as an example to describe how to obtain dependencies 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 Download the RPM package: Replace **hello** with the actual value of **component**.
yum install hello

----End

Downloading a Conan Package to 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).
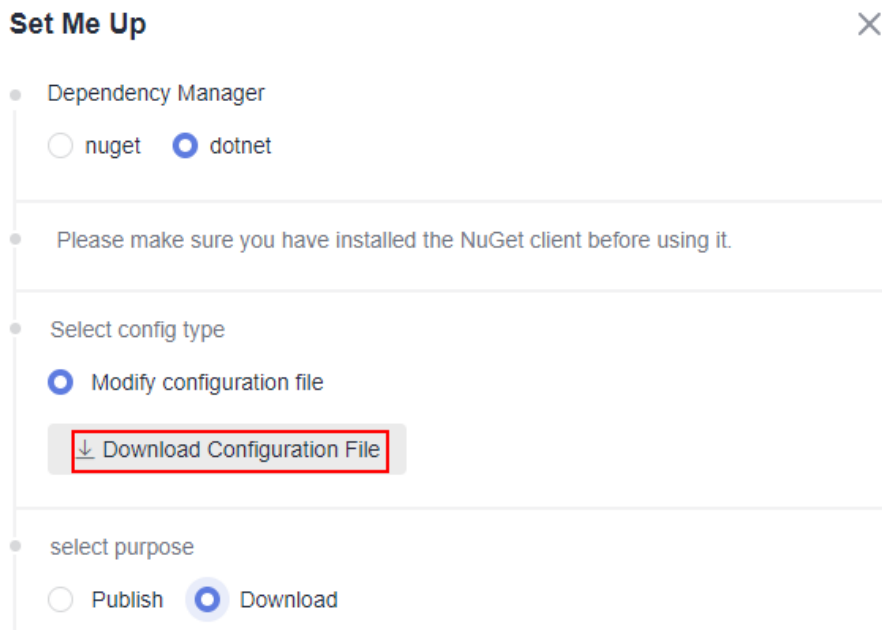
Step 2 Download the Conan dependency from the remote repository:
$ conan install ${package_name}/${package_version}@${package_username}/${channel} -r=cloud_artifact

Step 3 Check the downloaded Conan package:
$ conan search "*"

Step 4 Remove the package from the local cache:
$ conan remove ${package_name}/${package_version}@${package_username}/${channel}

----End

Downloading a NuGet Package to 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 package to be downloaded, and run the download statement. (If a configuration source exists, use the configured source name as the parameter following **-source**.)
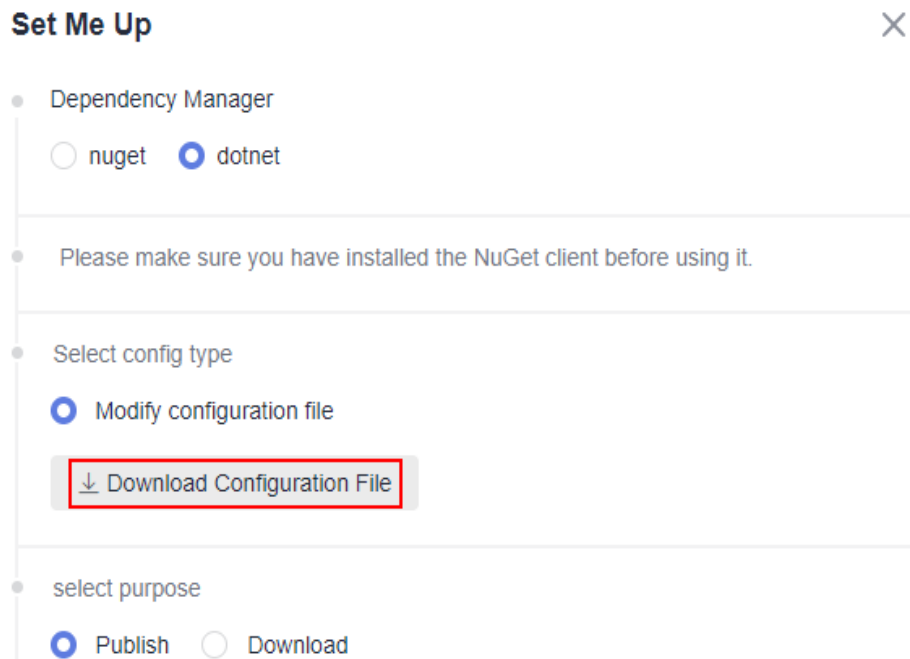
```
##----------------------NuGet Download---------------------##
nuget install <PACKAGE>
```

----End

Downloading a .NET Package to 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**.

Set Me Up

- Dependency Manager
  - ○ nuget ● dotnet
- Please make sure you have installed the NuGet client before using it.
- Select config type
  - ● Modify configuration file
    - [↓ Download Configuration File]
- select purpose
  - ● Publish   ○ Download

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 package to be downloaded, and run the download statement.

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

----End

Downloading a Docker Image to the Client

Prerequisites

- You have installed the Docker client.
- You have created a Docker registry in the self-hosted repo.

Downloading a Docker image to the client

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

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

url: repository URL, as shown in the following figure.
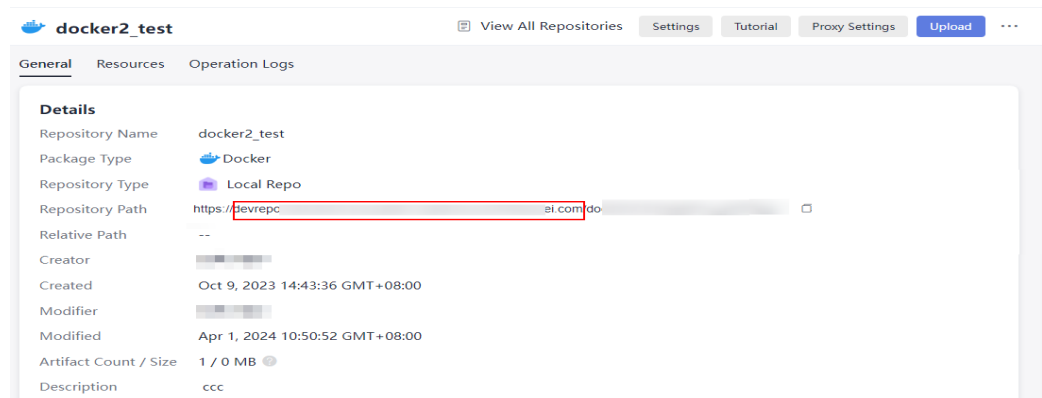
image_name: component name.

image_version: component version.

Downloading a Generic Package to the Client

Download the client.

```
curl -o {{localFileName}} -k -u "{{username}}:{{password}}" -X GET {{repo_url}}/{{filePath}}
```

localFileName: local path (including the name) for downloading the Generic package.

filePath: path (including the name) of the package in the Generic repository.

Obtain the values of *username*, *password*, and *repo_url* from the downloaded **generic.txt** file in **Step 2**, as shown in the following figure.

Downloading a CocoaPods Package to 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 packages

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

Step 4 Download the local client.

1. Perform Cocoapods authentication:
   ```
   sudo sh -c " echo 'machine {url}
   login {username}
   password {password}' > <.cocoapods_path>/.netrc"
   ```

2. Download the package from the remote repository:

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

package_name: name of the CocoaPods dependency to be downloaded.

url: repository URL of the self-hosted repo.

3. Download the artifacts from the CocoaPods repository.

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

package_name: The CocoaPods dependency cannot be modified.

url: enter the repository URL of the target self-hosted repo.
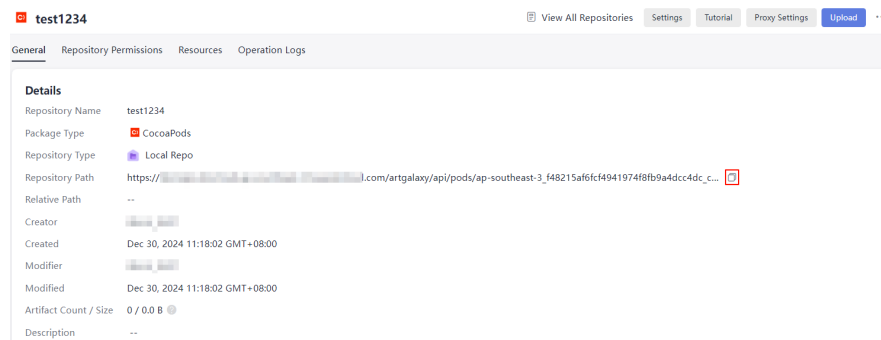
4. Query the downloaded package:

`pod repo-art list`

5. Remove the local dependency:

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

----End

Following the command line configuration to download the CocoaPods package

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 Check whether the Rudy client has been installed:

`rudy -v`

Step 4 Install the cocoapods-art plug-in:

`sudo gem install cocoapods-art`

Step 5 Add the self-hosted repo to your CocoaPods client:

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

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

url: repository URL of the CocoaPods repository.

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

Step 7 Download the local client.

Download the package from the remote repository:

pod repo-art update {package_name}//*package_name*: package name.

Query the downloaded package:

pod repo-art list

Remove the local dependency:

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

----End

Downloading an OHPM Package to the Client

Download the client.

ohpm install *{package name}*

package name: **name** parameter in the return value in **Uploading an OHPM Package from the Client**, as shown in the following figure.
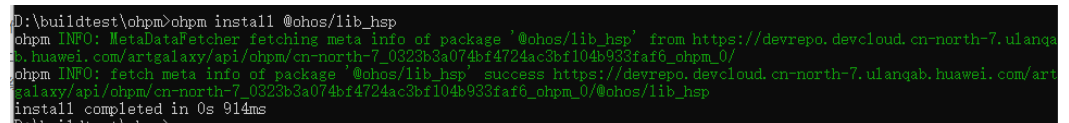
The following figure indicates the download success.

Helpful Links

Why Is Error 401 Returned When Uploading Maven Artifacts to Self-Hosted Repos?

4.7 Managing Packages 2.0

4.7.1 Viewing a Package

Viewing a Package in the Repo View

After you access the self-hosted repo, the repo view is displayed by default. The uploaded packages 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 package.

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

Click 🗗 in the **Details** and **Checksums** tabs to copy the information. In the search box, find the target package using the pasted information.

----End

Viewing a Package in the Version View

A self-hosted repo displays different package types by version. In **Version View** tab, you can filter and check artifacts 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 packages, see **Uploading/Downloading Packages on the Self-Hosted Repo Page**.

Step 3 Self-hosted repos host 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 package is displayed.

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

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

----End

4.7.2 Editing a Package

Searching for a Package

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

Step 2 Select the type of package to be searched for in the upper part of the page. By default, all types are selected. You can search for the package using either of the following methods:

- Select **File Name** mode.

    a. Enter the keyword of the file name in the search box, and click Q to search for the package.

    b. In the search result list, click a file name to view the package 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. Enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click Q to find the desired package.

    ----End

Downloading a Package

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

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

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

----End

Deleting a Package

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

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

Step 2 Click **Delete** on the right of a package or directory in the repository to delete it.

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

----End

Favoriting/Unfavoriting

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

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

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 items. Click a value in the **Path** column to go to the package details page.

----End

4.8 Recycle Bin

Repositories and packages 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 packages in the recycle bin on the homepage.

Step 1 Access the self-hosted repos page from the **homepage**.

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 packages 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 packages have been deleted. Click the repository name to view the deleted packages.

Operations are as follows.

Operation Type	Operation	Description
Restore	Restore a repository	Click in the **Operation** column to restore the repository.
	Restore a package	Go to the repository where the package to restore is located, and click in the **Operation** column to restore it.
	Batch restore packages	Go to the repository where the packages to restore are located, select them, 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 selected items in the recycle bin.
Delete	Delete a repository	Click in the **Operation** column to delete the repository.
	Delete a package	Go to the repository where the package to delete is located, and click in the **Operation** column to delete it.
	Batch delete packages	Go to the repository where the packages to delete are located, select them, and click **Clear** below the list to permanently delete them in batches.

Oper ation Type	Operatio n	Description
	Clear all	Click **Clear All** in the upper right corner of the page to delete all selected items in the recycle bin.

----End

Project-Level 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 packages 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 packages have been deleted. Click the repository name to view the deleted packages.

Operations are as follows.

Oper ation Type	Operatio n	Description
Resto re	Restore a repository	Click in the **Operation** column to restore the repository.
	Restore a package	Go to the repository where the package to restore is located, and click in the **Operation** column to restore it.

Oper ation Type	Operatio n	Description
	Batch restore packages	Go to the repository where the packages to restore are located, select them, 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 selected items in the recycle bin.
Delet e	Delete a repository	Click 🗑 in the **Operation** column to delete the repository.
	Delete a package	Go to the repository where the package to delete is located, and click 🗑 in the **Operation** column to delete it.
	Batch delete packages	Go to the repository where the packages to delete are located, select them, 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 selected items in the recycle bin.

----End

5 Release Repos 1.0

5.1 Accessing Release Repos

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

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:
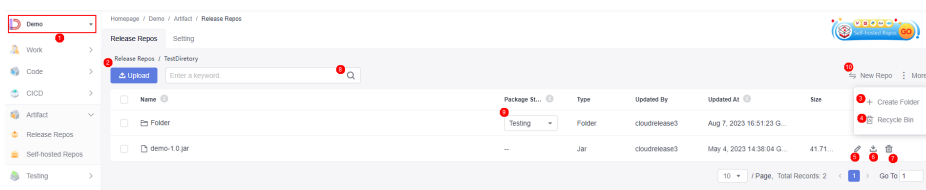
No.	Operation	Description
1	Switch project	Click the project name drop-down list to switch the project to view the legacy Release Repos.

No.	Operation	Description
2	Upload	Click **Upload** in the upper left corner of the list to manually upload local packages to Release Repos. The size of each file to upload cannot exceed 2 GB. **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 packages or folders.
5	Edit	Click ✎ in the **Operation** column to edit the package or folder. ● Package: You can change the name and release version of the 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 package.
7	Delete	● Click 🗑 in the **Operation** column. In the displayed dialog box, click **Remove to Recycle Bin** to move the corresponding package or folder to the recycle bin, or click **Clear** to delete it permanently. ● Select multiple 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. 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 package whose name contains the keyword.
9	Change status	After entering a level-1 folder, you can change the status of a level-2 folder by clicking the drop-down list in the **Package Status** column. The status can be **Not Released** or **Released**. A folder can change from **Not Released** to **Released**, but such a change is irreversible. If a folder is **Released**, the folder cannot be changed or edited (changing the folder name, changing the file name in the folder, uploading the folder, changing the version number, or creating a subfolder). You can only download or delete it.

No.	Operation	Description
10	New repo	Click **New Repo** to return to the new version of Release Repos.

----End

5.2 Managing Packages

Uploading a Package

Step 1 **Access Release Repos**.

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 the package.
- **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 package will be stored.
- **File**: select the package from your local PC to upload.

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

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

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

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

----End

Viewing/Editing a Package in Release Repos

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

Access the Release Repos homepage and click the name of a package. A drawer is displayed, showing the 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 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 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 package. Click **Build Task** to link to the task in CodeArts Build.

Basic Information	Build Information	Build Packages
Build Task	cangku-97481177	
Build No.	20220804.1	
Builder	000	
Code Repository		
Code Branch	master	
Commit ID		

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

Basic Information	Build Information	Build Packages		
Build Task	**Build No.**	**Size**	**Uploaded At**	**Operation**
cangku-97481177				⬇

Total 1 Records ‹ **1** › Go To 1

5.3 Clearing Policies

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.

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.

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 packages that match the file paths you set when clearing files. You can set multiple file paths, each starting with a slash (/) and separated by semicolons (;).

----End

5.4 Recycle Bin

Packages or folders deleted from 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 packages or folders as required. Once you delete a package or folder from the recycle bin, it cannot be recovered. Exercise caution when performing this operation.

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

No.	Operation	Description
6	Clear	Click in the **Operation** column to delete the package or folder.

----End

6 Self-Hosted Repos 1.0

6.1 Accessing a Repository

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 Repository

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

6.2 Configuring a Repository

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

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.
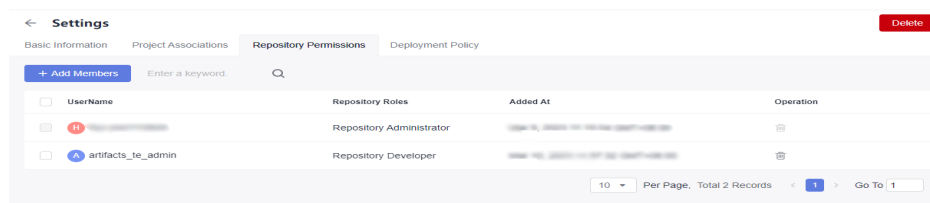
Step 4 Add members.

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

Figure 6-1 Adding members
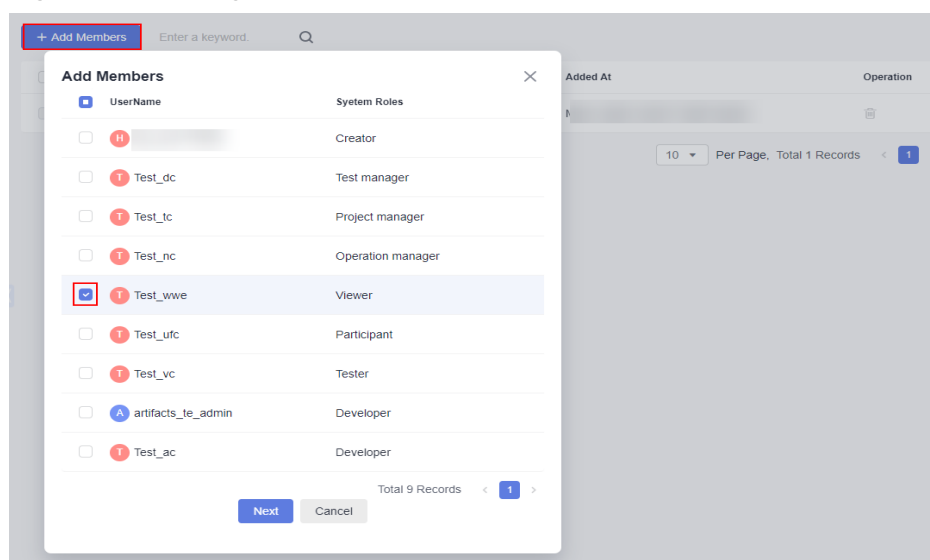
Step 5 Assign roles to members.

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

Figure 6-2 Assigning roles to members
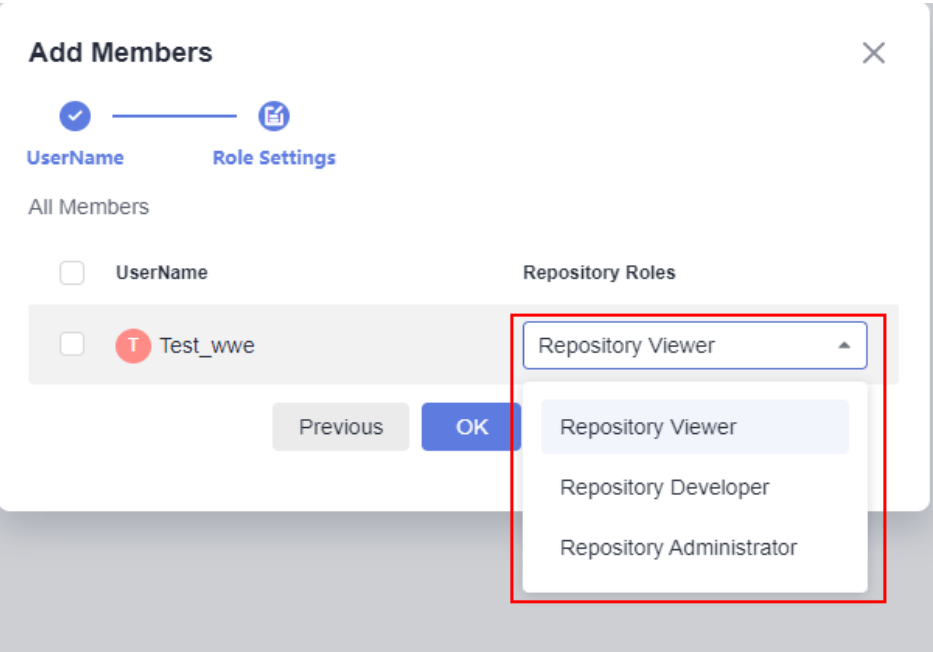
Step 6 Click **OK**. The member is added and the role is configured. The new member is displayed in the list.

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

Figure 6-3 Configuring repository roles in batches
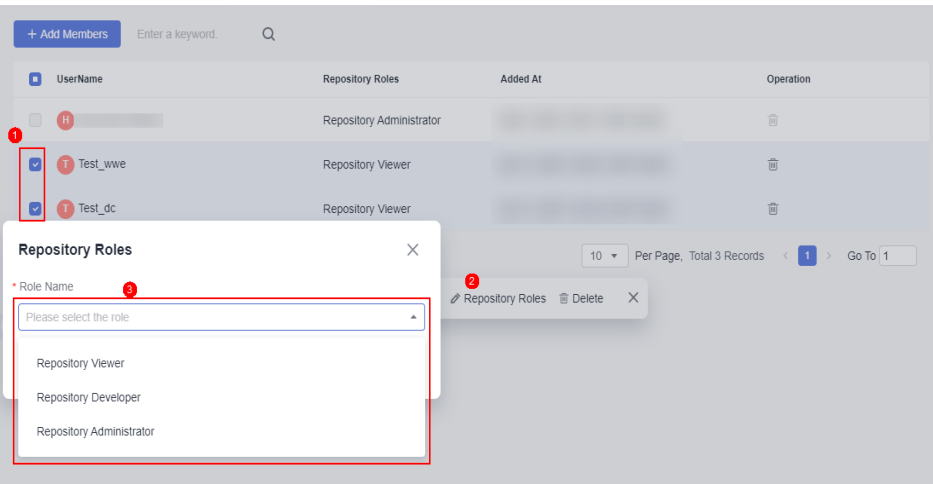
Table 6-1 lists the operation permissions of each repository role.

Table 6-1 Role permissions

Operatio n/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 package	√	√	×	√	√	×
Download a package	√	√	√	√	√	√
Delete a package	√	√	×	√	√	×
Restore a package	√	√	×	√	√	×
Permanently delete a package	√	√	×	√	√	×
Delete a repository	√	×	×	×	×	×
Restore a repository	√	√	×	√	√	×
Permanently delete a repository	√	×	×	×	×	×
Clear all	√	√	√	×	×	×

Restore all	√	√	√	×	×	×
Manage user permissi ons	√	√	√	√	×	×

----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. Click 🧑‍🦱 in the **Operation** column of the target member.
2. Click the **Unassigned** tab.
3. Select the desired repositories and roles, and click **OK**.

Figure 6-4 Assigning repository roles
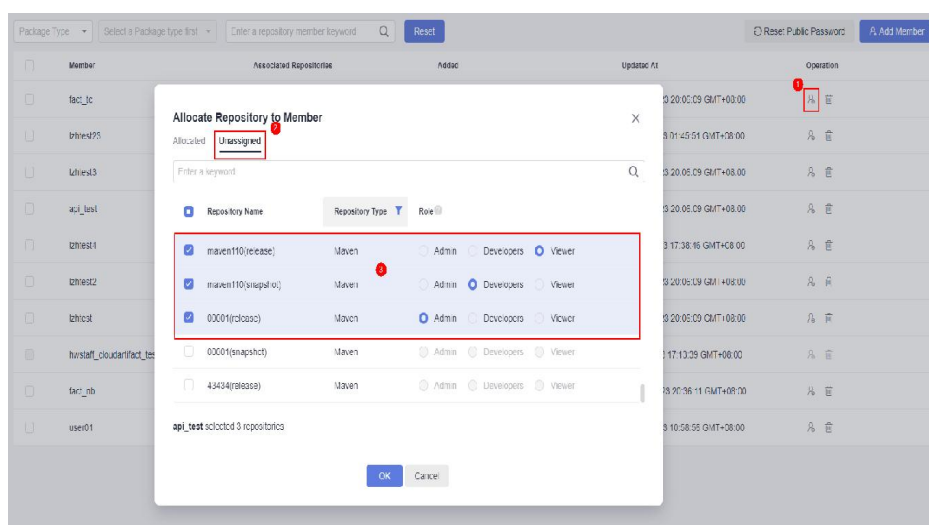
Step 5 (Optional) Perform the operations listed in **Table 6-2** on the **User Permissions** page.

Table 6-2 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. ● 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.
Reset public password (only for tenant administrators)	The public password is used by the CodeArts Build to upload and download packages 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

6.3 Managing a Repository

6.3.1 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**.

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.

To delete path rules, click 🗑.

----End

6.3.2 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.
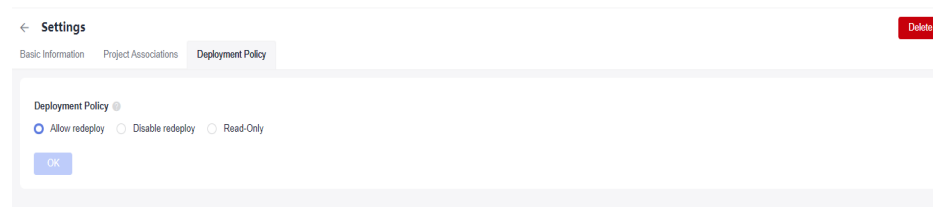
- **Allow redeploy** (enabled by default): Artifacts in the same path can be uploaded, replacing the original package.

- **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 Settings are automatically saved by the system.

----End

6.3.3 Configuring Clearing Policies for a Maven Repository

You can delete artifacts that meet deletion conditions in batches. When you create 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. During each build, Maven checks for new snapshots in the remote repository. You can set limits on how many snapshots 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 6-5 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

6.3.4 Associating Maven Repositories 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)**.

6.4 Managing Packages

6.4.1 Uploading Packages on the Self-Hosted Repo Page

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

Procedure

To upload a package, 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 package is to be uploaded.

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

Step 3 Set the package parameters, select the file, and click **Upload**. **You are advised not to upload files containing sensitive information such as plaintext accounts and passwords to self-hosted repos.**

Detailed configuration for each package type is described below.

----End

Maven Artifacts

- 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 package.

- 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 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 package. **Version** indicates the version of the 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 Artifact

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>
```

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 Packages

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

The npm 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 Package

npm 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.

When uploading a package, ensure that the **PackageName** starts with a path in the path list added during repository creation. For details, see **Table 4-2** in the help guide.

Example:

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

When uploading the package to the registry, 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 package in .tgz format in the repository list and the corresponding metadata is generated in the **.npm** directory.

Uploading a Go Package

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 package, 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 Package

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 package 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
PackageNam e	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 Package

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.	Compilati on Times	Applicable Hardware Platform
hello	0	17	2	54	x86_64

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

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 package is to be uploaded.

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

Step 3 Set the package 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 package name directory. You can use Yum to install the package.

Uploading a Debian Package

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

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

A NuGet package is a single ZIP file with a **.nupkg** extension. You 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 package is to be uploaded.

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

Step 3 View the package that is successfully uploaded in the repository list.

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

package stores components.

----End

6.4.2 Viewing a Package

Viewing a Package in the Repo View

After you access the self-hosted repo, the repo view is displayed by default. The uploaded packages 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 package.

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

Click [icon] in the **Details** and **Checksums** tabs to copy the information. In the search box, find the target package using the pasted information.

----End

Viewing a Package in the Version View

A self-hosted repo displays different package types by version. In **Version View** tab, you can filter and check artifacts 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 packages, see **Uploading/Downloading Packages on the Self-Hosted Repo Page**.

Step 3 Self-hosted repos host 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 package is displayed.

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

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

----End

6.4.3 Editing a Package

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

Searching for a Package

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 Select the type of package to be searched for in the upper part of the page. By default, all types are selected. You can search for the package using either of the following methods:

- Select **File Name** mode.

  a. Enter the keyword of the file name in the search box, and click [icon] to search for the package.

  b. In the search result list, click a file name to view the package 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. Enter the **MD5/SHA-1/SHA-256/SHA-512 checksum** and click Q to find the desired package.

----End

Downloading a Package

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

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

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

----End

Deleting a Package

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

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

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 package to be favorited, and click its name.

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

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 items. Click a value in the **Path** column to go to the package details page.

----End

6.5 Recycle Bin

Repositories and packages 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 packages in the list as required. **If you choose to delete a repository or package in the recycle bin, it cannot be restored anymore. Exercise caution when performing this operation.**

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 packages have been deleted. Click the repository name to view the deleted packages.

Operations are as follows.

Operation Type	Operation	Description
Restore	Restore a repository	Click 🗑 in the **Operation** column to restore the repository.
	Restore a package	Go to the repository where the package to restore is located, and click 🗑 in the **Operation** column to restore it.
	Batch restore packages	Go to the repository where the packages to restore are located, select them, 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 selected items in the recycle bin.
Delete	Delete a repository	Click 🗑 in the **Operation** column to delete the repository.
	Delete a package	Go to the repository where the package to delete is located, and click 🗑 in the **Operation** column to delete it.
	Batch delete packages	Go to the repository where the packages to delete are located, select them, and click **Clear** below the list to permanently delete them in batches.

| Oper ation Type	Operatio n	Description
	Clear all	Click **Clear All** in the upper right corner of the page to delete all selected items in the recycle bin.

----End

6.6 Connecting Self-Hosted Repos to Your Local Development Environment

Downloading Configuration Files of Self-Hosted Repos

You can connect the self-hosted repo to a local development environment so that private packages 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.
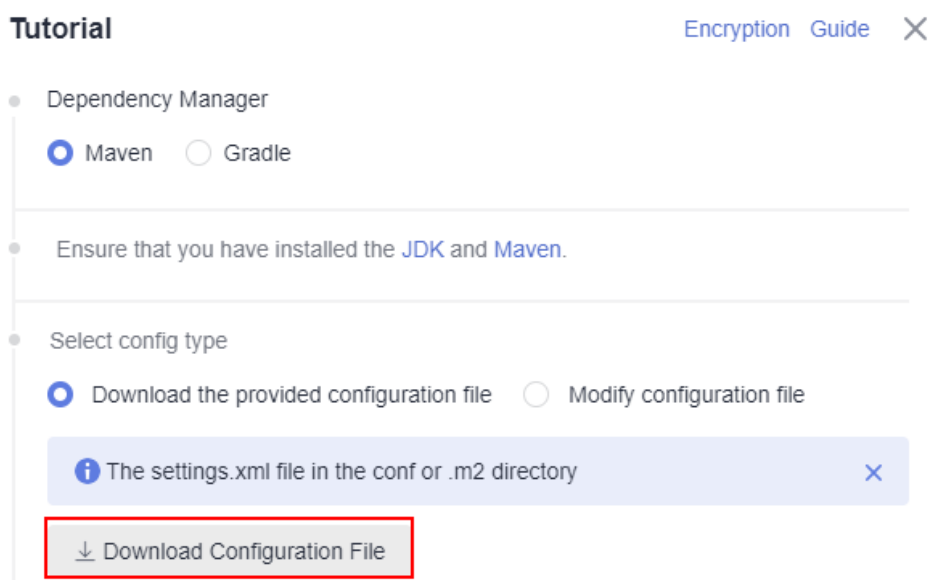
Step 4 Copy the downloaded file to the corresponding directory based on the instructions in the information dialog box.

----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 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 packages through a client. The procedure is the same as that of self-hosted repo 2.0. For details, see **Uploading Packages to Self-Hosted Repos from the Client** and **Downloading Packages from Self-Hosted Repos to the Client**.

7 IP Address Whitelist

The IP address whitelist 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 IP address whitelist.

Creating a Tenant-Level IP Address Whitelist

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 7-1 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