CodeArts Artifact

# Getting Started

| | |
|---|---|
| **Issue** | 01 |
| **Date** | 2023-11-30 |

HUAWEI TECHNOLOGIES CO., LTD.

# Security Declaration

## Vulnerability

Huawei's regulations on product vulnerability management are subject to "Vul. Response Process". For details about the policy, see the following website:https://www.huawei.com/en/psirt/vul-response-process
For enterprise customers who need to obtain vulnerability information, visit:https://securitybulletin.huawei.com/enterprise/en/security-advisory

# Contents

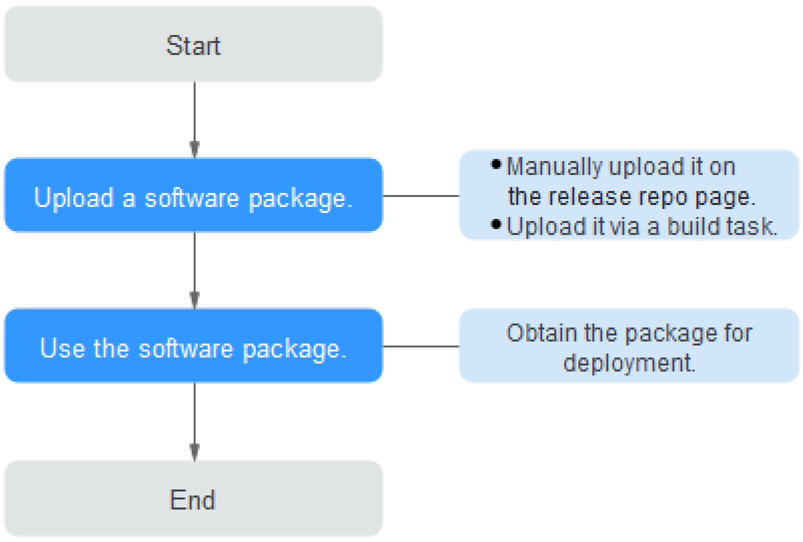
# 1 Getting Started with a Release Repo

This section describes the general procedure of a release repo to help you quickly get started with it.

Before using a release repo, ensure that you already have a project. If no project is available, **create one**.

**Figure 1-1** shows the general procedure of a release repo.

**Figure 1-1** General procedure of a release repo


Start
Upload a software package.
• Manually upload it on the release repo page.
• Upload it via a build task.
Use the software package.
Obtain the package for deployment.
End

## Manually Uploading a Software Package on the Release Repo Page

**Step 1** Log in to CodeArts, choose **Services** > **Artifact** from the top menu, and click the **Release Repos** tab.

**Step 2** Go to the repository with the same name as the project and click **Upload** in the upper right corner.

**Step 3** Select the target software package and click **Open**.

**----End**

## Releasing a Software Package via a Build Task to a Release Repo

The following procedure uses Maven as an example to describe how to release a software package via a build task to a release repo.

**Step 1** Prepare a code repository.

1. Log in to CodeArts and go to a created project.
2. Go to CodeArts Repo and create a Maven repository. For details, see Creating a Repository Using a Template**Creating a Repository Using a Template**. This procedure uses the **Java Web Demo** template.

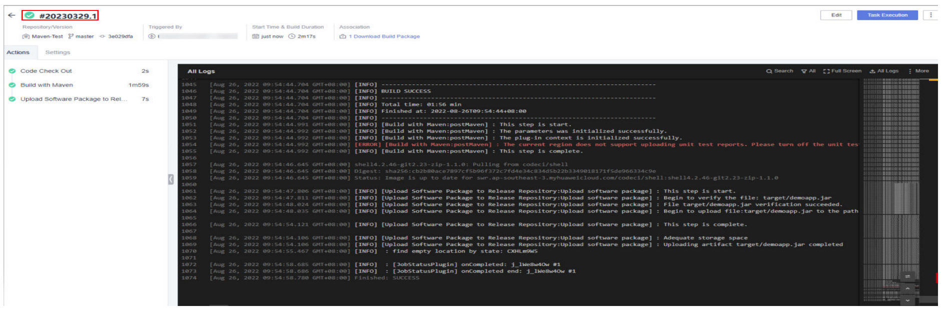
**Step 2** Configure and execute a build task.

1. Go to the code repository, and click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Maven** and click **Next**.
2. Edit the build actions as required. In this example, the default values in the template are used.
3. Click **Create and Run** to start the build task.

   After the task is successfully executed, record the number following **#** in the upper left, as shown in **Figure 1-2**.

**Figure 1-2** Build task



#20230329.1

Repository/Version: Maven-Test master 3e029dfa
Triggered By: [redacted]
Start Time & Build Duration: just now 2m17s
Association: 1 Download Build Package

Actions | Settings

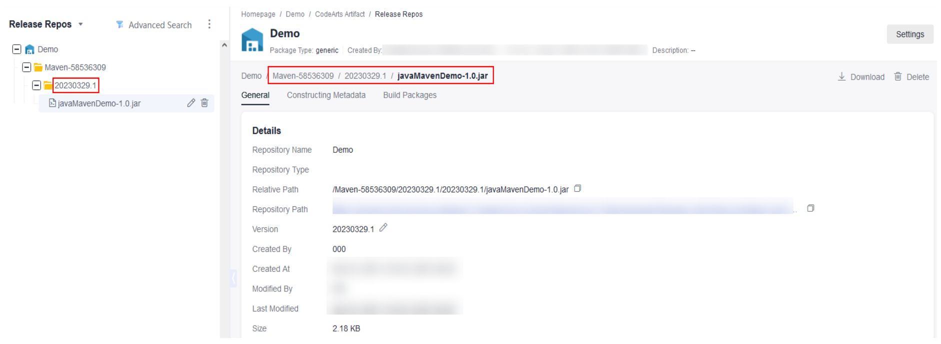
| | |
|---|---|
| Code Check Out | 2s |
| Build with Maven | 1m59s |
| Upload Software Package to Rel... | 7s |

```
All Logs
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] ------------------------------------------------------------------------
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] BUILD SUCCESS
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] ------------------------------------------------------------------------
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] Total time: 01:56 min
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] Finished at: 2022-08-26T09:54:44+08:00
[Aug 26, 2022 09:54:44.704 GMT+08:00] [INFO] ------------------------------------------------------------------------
[Aug 26, 2022 09:54:44.991 GMT+08:00] [INFO] [Build with Maven:postMaven] : This step is start.
[Aug 26, 2022 09:54:44.992 GMT+08:00] [INFO] [Build with Maven:postMaven] : The parameters was initialized successfully.
[Aug 26, 2022 09:54:44.992 GMT+08:00] [INFO] [Build with Maven:postMaven] : The plug-in context is initialized successfully.
[Aug 26, 2022 09:54:44.992 GMT+08:00] [ERROR] [Build with Maven:postMaven] : The current region does not support uploading unit test reports. Please turn off the unit tes
[Aug 26, 2022 09:54:44.992 GMT+08:00] [INFO] [Build with Maven:postMaven] : This step is complete.

[Aug 26, 2022 09:54:46.645 GMT+08:00] shell4.2.46-git2.23-zip-1.1.0: Pulling from codeci/shell
[Aug 26, 2022 09:54:46.645 GMT+08:00] Digest: sha256:cb2b80ace7897cf5b96f372c7fd4e34c834d5b22b3349018171f5de966334c9e
[Aug 26, 2022 09:54:46.645 GMT+08:00] Status: Image is up to date for swr.ap-southeast-3.myhuaweicloud.com/codeci/shell:shell4.2.46-git2.23-zip-1.1.0

[Aug 26, 2022 09:54:47.806 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : This step is start.
[Aug 26, 2022 09:54:47.811 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : Begin to verify the file: target/demoapp.jar
[Aug 26, 2022 09:54:48.024 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : File target/demoapp.jar verification succeeded.
[Aug 26, 2022 09:54:48.035 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : Begin to upload file:target/demoapp.jar to the path

[Aug 26, 2022 09:54:54.121 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : This step is complete.

[Aug 26, 2022 09:54:54.106 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : Adequate storage space
[Aug 26, 2022 09:54:54.106 GMT+08:00] [INFO] [Upload Software Package to Release Repository:Upload software package] : Uploading artifact target/demoapp.jar completed
[Aug 26, 2022 09:54:55.467 GMT+08:00] [INFO]  : find empty location by state: CXHLm9W5

[Aug 26, 2022 09:54:58.685 GMT+08:00] [INFO]  : [JobStatusPlugin] onCompleted: j_lWe8w4Ow #1
[Aug 26, 2022 09:54:58.686 GMT+08:00] [INFO]  : [JobStatusPlugin] onCompleted end: j_lWe8w4Ow #1
[Aug 26, 2022 09:54:58.780 GMT+08:00] Finished: SUCCESS
```

**Step 3** View the software package.

1. Access the release repo of a project.
2. Go to the folder with the same name as the build task.
3. Go to the folder named after the number on the build task page, and view the generated software package, as shown in **Figure 1-3**.

**Figure 1-3** Viewing the software package



Homepage / Demo / CodeArts Artifact / Release Repos

Demo
Package Type: generic Created By: [redacted] Description: --

Demo / Maven-58536309 / 20230329.1 / javaMavenDemo-1.0.jar

General | Constructing Metadata | Build Packages

Details

| | |
|---|---|
| Repository Name | Demo |
| Repository Type | |
| Relative Path | /Maven-58536309/20230329.1/20230329.1/javaMavenDemo-1.0.jar |
| Repository Path | [redacted] |
| Version | 20230329.1 |
| Created By | 000 |
| Created At | [redacted] |
| Modified By | [redacted] |
| Last Modified | [redacted] |
| Size | 2.18 KB |

NOTE

If you have set **Version** for the **Upload Software Package to Release Repos** action, the software package will be saved in a folder named after the release version.

**----End**

## Obtaining a Software Package from a Release Repo for Deployment

This section uses the software package released in **Releasing a Software Package via a Build Task to a Release Repo** as an example to describe how to obtain this package from a release repo for deployment.

**Step 1** Go to the CodeArts homepage and click the target project name to access the project.

**Step 2** Choose **Settings** > **General** > **Basic Resources**. The **Host Clusters** page is displayed.

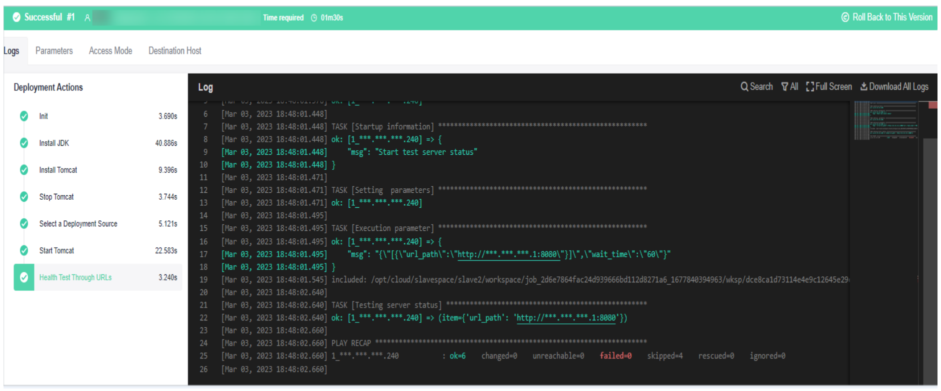
**Step 3** Click **Create Host Cluster**, enter the following information, and click **Save**. Add the target host to the created host cluster.

**Step 4** **Create an application**. Configure it as required. In this example, configure it as follows:

- Select the **Deploy a Tomcat Application** template.
- In the **Select a Deployment Source** action, select the software package stored in the location mentioned in **Releasing a Software Package via a Build Task to a Release Repo**.

**Step 5** On the **Environment Management** tab page, create environment and add hosts. For details, see **Creating a Host Group and Adding Trusted Hosts to the Host Group**.

**Step 6** Click **Save & Deploy**. Execute the deployment application. When the deployment status becomes successful, the deployment application has obtained the software package from the release repo and has deployed it on the target host.



Deployment Actions

| | |
|---|---|
| Init | 3.690s |
| Install JDK | 40.886s |
| Install Tomcat | 9.396s |
| Stop Tomcat | 3.744s |
| Select a Deployment Source | 5.121s |
| Start Tomcat | 22.583s |
| Health Test Through URLs | 3.240s |

```
[Mar 03, 2023 18:48:01.448]
[Mar 03, 2023 18:48:01.448] TASK [Startup information] *****************************************************
[Mar 03, 2023 18:48:01.448] ok: [1_***.***.***.240] => {
[Mar 03, 2023 18:48:01.448]     "msg": "Start test server status"
[Mar 03, 2023 18:48:01.448] }
[Mar 03, 2023 18:48:01.471]
[Mar 03, 2023 18:48:01.471] TASK [Setting  parameters] *****************************************************
[Mar 03, 2023 18:48:01.471] ok: [1_***.***.***.240]
[Mar 03, 2023 18:48:01.495]
[Mar 03, 2023 18:48:01.495] TASK [Execution parameter] *****************************************************
[Mar 03, 2023 18:48:01.495] ok: [1_***.***.***.240] => {
[Mar 03, 2023 18:48:01.495]     "msg": "{\"[{\"url_path\":\"http://***.***.***.1:8080\"}]\",\"wait_time\":\"60\"}"
[Mar 03, 2023 18:48:01.495] }
[Mar 03, 2023 18:48:01.545] included: /opt/cloud/slavespace/slave2/workspace/job_2d6e7864fac24d939666bd112d8271a6_1677840394963/wksp/dce8ca1d73114e4e9c12645e29
[Mar 03, 2023 18:48:02.640]
[Mar 03, 2023 18:48:02.640] TASK [Testing server status] ***************************************************
[Mar 03, 2023 18:48:02.640] ok: [1_***.***.***.240] => (item={'url_path': 'http://***.***.***.1:8080'})
[Mar 03, 2023 18:48:02.660]
[Mar 03, 2023 18:48:02.660] PLAY RECAP *********************************************************************
[Mar 03, 2023 18:48:02.660] 1_***.***.***.240          : ok=6    changed=0    unreachable=0    failed=0    skipped=4    rescued=0    ignored=0
[Mar 03, 2023 18:48:02.660]
```

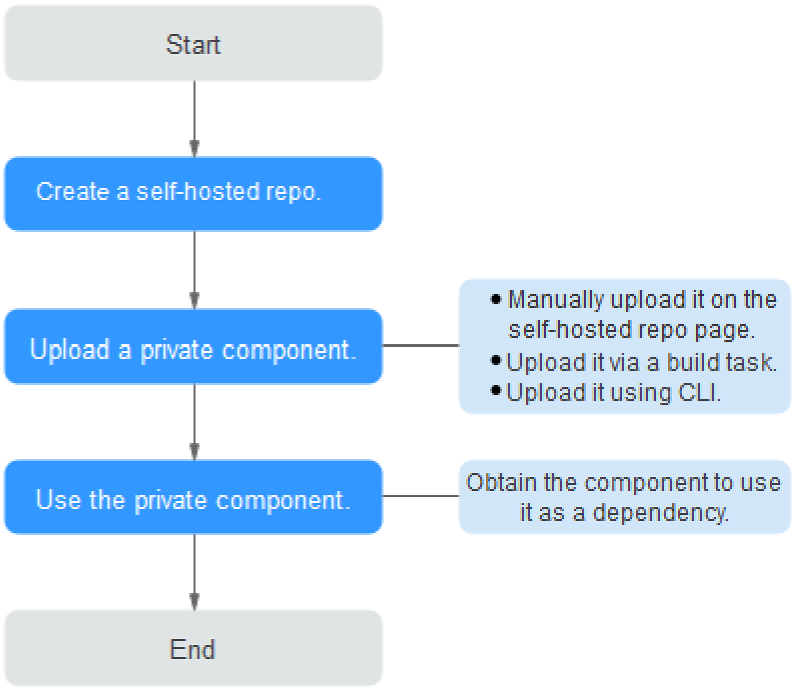
**----End**

# 2 Getting Started with a Self-hosted Repo

This section describes the general procedure of a self-hosted repo to help you quickly get started with it.

Figure 2-1 shows the general procedure of a self-hosted repo.

**Figure 2-1** General procedure of a self-hosted repo


Start
Create a self-hosted repo.
Upload a private component.
• Manually upload it on the self-hosted repo page.
• Upload it via a build task.
• Upload it using CLI.
Use the private component.
Obtain the component to use it as a dependency.
End

## Creating a Self-hosted Repo

**Step 1** Log in to the CodeArts homepage, choose **Services** > **Artifact** on the top navigation bar, and click the **Self-hosted Repos** tab.

**Step 2** Click **Create Self-hosted Repo** on the left.

**Step 3** Configure the basic information and click **OK**.

For details about how to configure a repository, see **Creating a Self-hosted Repo**.

**----End**

## Uploading a Private Component on the Self-hosted Repo Page

**Step 1** Go to the self-hosted repo page, and click the target repository in the left pane.

**Step 2** Click **Upload**.

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

For details about how to set the component parameters, see **Uploading a Private Component**.

**----End**

## Uploading/Obtaining a Private Component via a Build Task

You can upload Maven, npm, Go, and PyPI components to a self-hosted repo and obtain these components from the repository to use them as build dependencies.

For details, see:

- **Releasing/Obtaining a Maven Component via a Build Task**
- **Releasing/Obtaining an npm Component via a Build Task**
- **Releasing/Obtaining a Go Component via a Build Task**
- **Releasing/Obtaining a PyPI Component via a Build Task**

## Uploading/Obtaining an RPM Component Using Linux Commands

By running Linux commands, you can upload RPM and Debian components to or download them from a self-hosted repo.

For details, see **Uploading/Obtaining an RPM Component Using Linux Commands**.

For details, see **Uploading/Obtaining a Debian Component Using Linux Commands**.

# 3 Releasing/Obtaining a Maven Component via a Build Task

This section describes how to release a Maven component to a self-hosted repo via a build task and obtain the component from the repository for deployment.

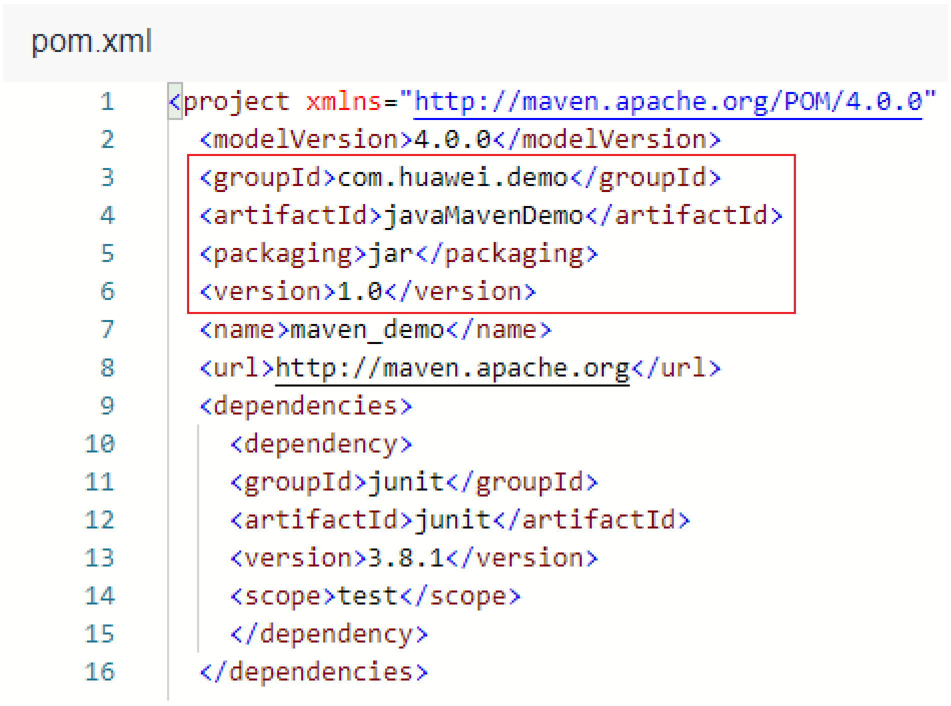
## Prerequisites

1. You already have a project. If no project is available, **create one**.
2. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.
3. You have created a self-hosted Maven repo and **associated it with the project**.

## Releasing a Maven Component to a Self-hosted Repo

**Step 1** Configure a code repository.

1. Log in to CodeArts and go to a created project. Choose **Services** > **Repo** on the top navigation bar.
2. Create a Maven repository. For details, see **Creating a Repository Using a Template**. This procedure uses the **Java Maven Demo** template.
3. Go to the code repository and view the component configuration in the **pom.xml** file.



pom.xml

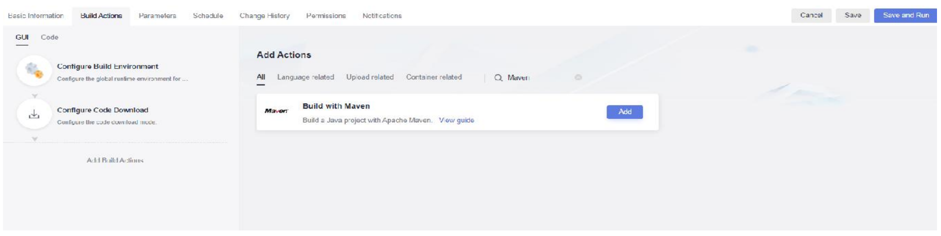
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>maven_demo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
  </dependencies>
```

**Step 2** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Blank Template** and click **Next**.

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



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

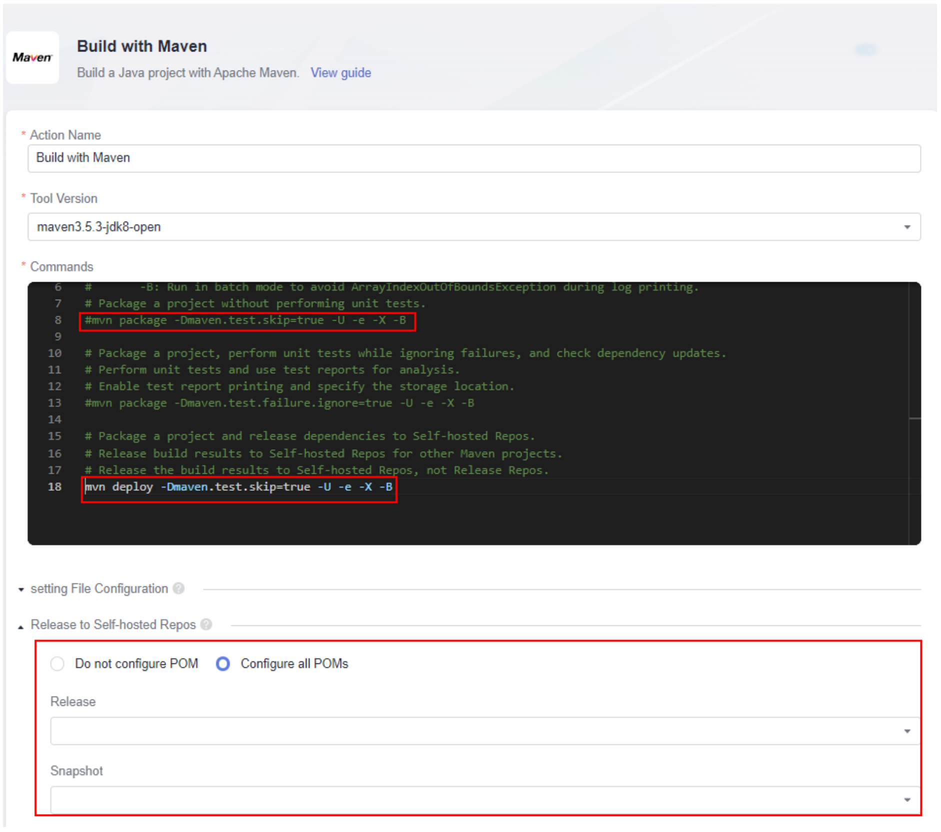
   – Select the desired tool version. In this example, **maven3.5.3-jdk8-open** is used.

   – Find the following command and delete **#** in front of this command:
   ```
   #mvn deploy -Dmaven.test.skip=true -U -e -X -B
   ```
   Find the following command and add **#** in front of this command:
   ```
   mvn package -Dmaven.test.skip=true -U -e -X -B
   ```

   – Select **Configure all POMs** under **Release to Self-hosted Repos**, and select the Maven repository associated with the project.

   NOTE

   If no option is available in the drop-down list, associate the Maven repository with the project of the build task by referring to **Managing the Association Between Repositories and Projects**.



**Build with Maven**

Build a Java project with Apache Maven. View guide

* Action Name: Build with Maven

* Tool Version: maven3.5.3-jdk8-open

* Commands

```
#       -B: Run in batch mode to avoid ArrayIndexOutOfBoundsException during log printing.
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B

# Package a project, perform unit tests while ignoring failures, and check dependency updates.
# Perform unit tests and use test reports for analysis.
# Enable test report printing and specify the storage location.
#mvn package -Dmaven.test.failure.ignore=true -U -e -X -B

# Package a project and release dependencies to Self-hosted Repos.
# Release build results to Self-hosted Repos for other Maven projects.
# Release the build results to Self-hosted Repos, not Release Repos.
mvn deploy -Dmaven.test.skip=true -U -e -X -B
```

setting File Configuration

Release to Self-hosted Repos

Do not configure POM [✓ Configure all POMs]

Release:

Snapshot:

**Step 3** Click **Create and Run** to start the build task.

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

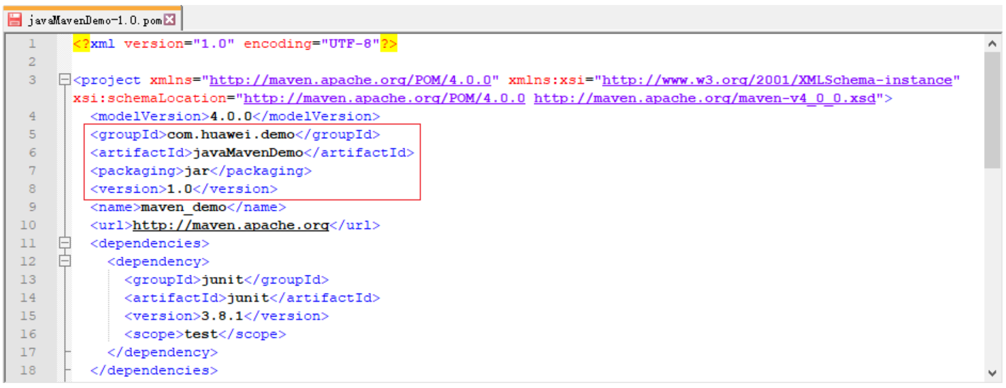
**----End**

## Obtaining a Maven Component from a Self-hosted Repo

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

**Step 1** Configure a code repository.

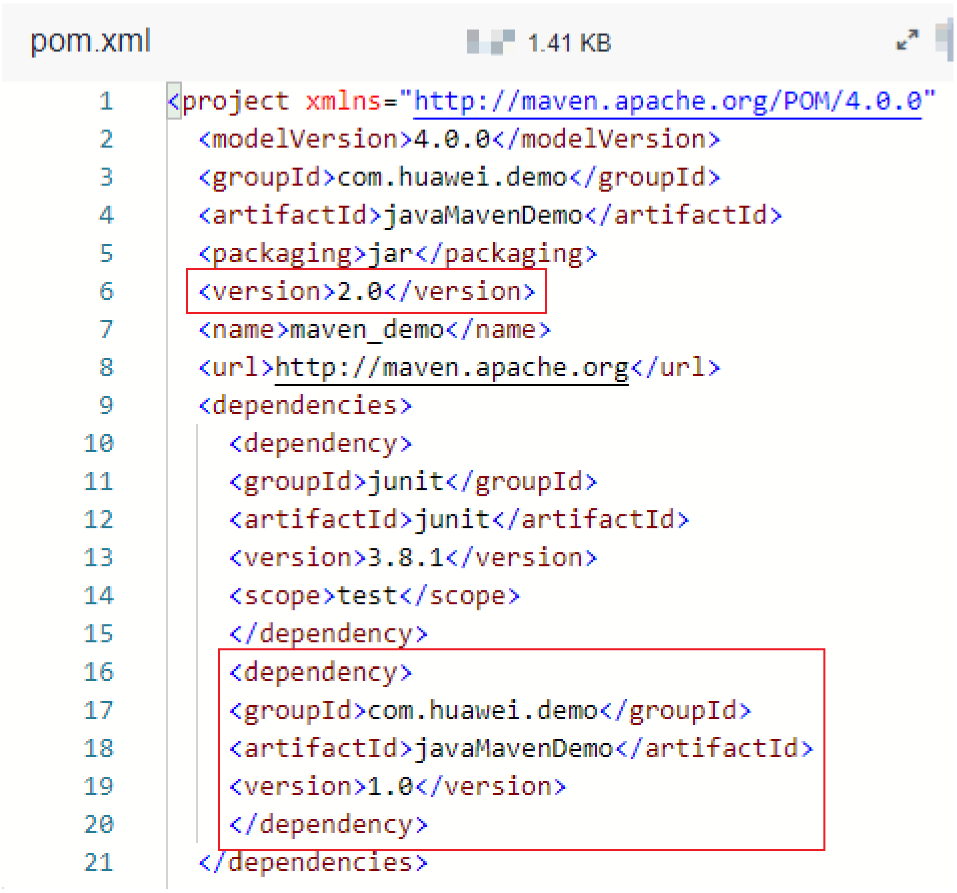
1. Go to the self-hosted Maven repo and find the Maven component. Click the **.pom** file with the same name as the component and click **Download** on the right.
2. Open the downloaded file and locate the **<groupId>**, **<artifactId>**, and **<version>** lines.



javaMavenDemo-1.0.pom

```xml
<?xml version="1.0" encoding="UTF-8"?>

<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>maven_demo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

3. Go to CodeArts Repo. Create a Maven repository. For details, see **Creating a Repository Using a Template**. This procedure uses the **Java Maven Demo** template.
4. Go to the code repository and edit the **pom.xml** file. Copy the dependency code segment to the **dependencies** code segment and modify the version number (for example, **2.0**).



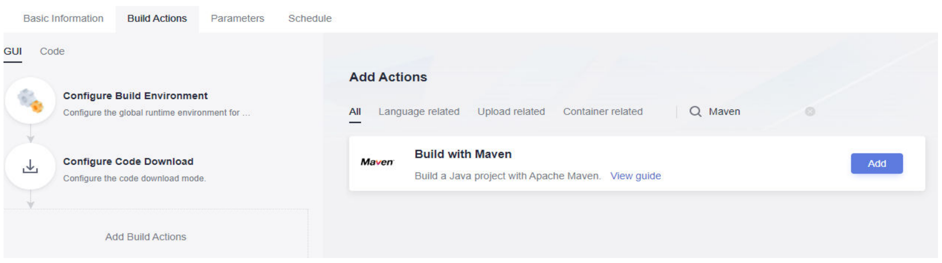
pom.xml 1.41 KB

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.huawei.demo</groupId>
  <artifactId>javaMavenDemo</artifactId>
  <packaging>jar</packaging>
  <version>2.0</version>
  <name>maven_demo</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
    </dependency>
    <dependency>
    <groupId>com.huawei.demo</groupId>
    <artifactId>javaMavenDemo</artifactId>
    <version>1.0</version>
    </dependency>
  </dependencies>
```

**Step 2** Configure and execute a build task.

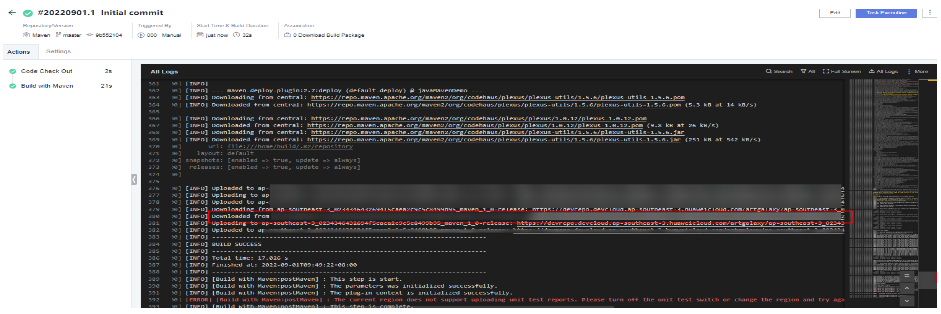
1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Blank Template** and click **Next**.
2. Add the **Build with Maven** action.



3. Click **Create and Run** to start the build task.

   After the task is successfully executed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the self-hosted repo.



```
0] [INFO]
0] [INFO] --- maven-deploy-plugin:2.7:deploy (default-deploy) @ javaMavenDemo ---
0] [INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.5.6/plexus-utils-1.5.6.pom
0] [INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.5.6/plexus-utils-1.5.6.pom (5.3 kB at 14 kB/s)

0] [INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.12/plexus-1.0.12.pom
0] [INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.12/plexus-1.0.12.pom (9.8 kB at 26 kB/s)
0] [INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.5.6/plexus-utils-1.5.6.jar
0] [INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.5.6/plexus-utils-1.5.6.jar (251 kB at 542 kB/s)
0]       url: file:///home/build/.m2/repository
0]    layout: default
0] snapshots: [enabled => true, update => always]
0]  releases: [enabled => true, update => always]
0]

0] [INFO] Uploaded to ap-[redacted]
0] [INFO] Uploading to ap
0] [INFO] Uploaded to ap-
0] [INFO] Downloading from ap-southeast-3_0234346432694f5caea2c9c5c8499b95_maven_1_0-release: https://devrepo.devcloud.ap-southeast-3.huaweicloud.com/artgalaxy/ap-southeast-3_0
0] [INFO] Downloaded from [redacted]
0] [INFO] Uploading to ap-southeast-3_0234346432694f5caea2c9c5c8499b95_maven_1_0-release: https://devrepo.devcloud.ap-southeast-3.huaweicloud.com/artgalaxy/ap-southeast-3_0234
0] [INFO] Uploaded to ap[redacted]
0] [INFO] ------------------------------------------------------------------------
0] [INFO] BUILD SUCCESS
0] [INFO] ------------------------------------------------------------------------
0] [INFO] Total time: 17.026 s
0] [INFO] Finished at: 2022-09-01T09:49:22+08:00
0] [INFO] ------------------------------------------------------------------------
0] [INFO] [Build with Maven:postMaven] : This step is start.
0] [INFO] [Build with Maven:postMaven] : The parameters was initialized successfully.
0] [INFO] [Build with Maven:postMaven] : The plug-in context is initialized successfully.
0] [ERROR] [Build with Maven:postMaven] : The current region does not support uploading unit test reports. Please turn off the unit test switch or change the region and try aga
0] [INFO] [Build with Maven:postMaven] : This step is complete.
```

**----End**

# 4 Releasing/Obtaining an npm Component via a Build Task

This section describes how to release a component to a self-hosted npm repo via a build task and obtain a dependency from the repository for deployment.

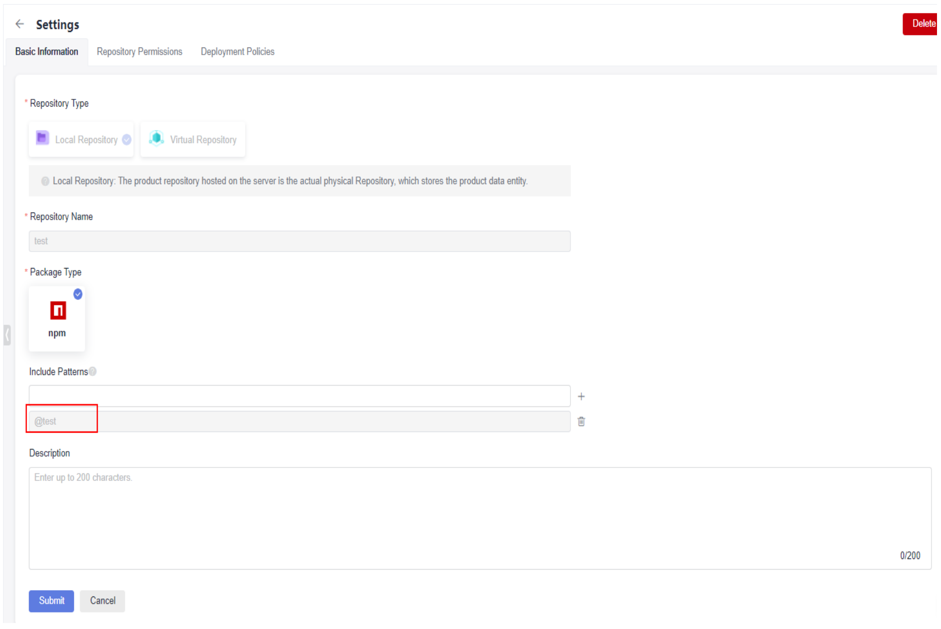
## Prerequisites

1. You already have a project. If no project is available, **create one**.
2. You have created a self-hosted npm repo.
3. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.

## Releasing a Component to a Self-hosted npm Repo

**Step 1** Download the configuration file of the self-hosted repo.

1. Log in to the CodeArts homepage and access the npm self-hosted repo for npm. Click **Settings** in the upper right corner and record the repository path.



Settings

Basic Information | Repository Permissions | Deployment Policies

* Repository Type: Local Repository [✓] | Virtual Repository

Local Repository: The product repository hosted on the server is the actual physical Repository, which stores the product data entity.

* Repository Name: test

* Package Type: npm [✓]

Include Patterns

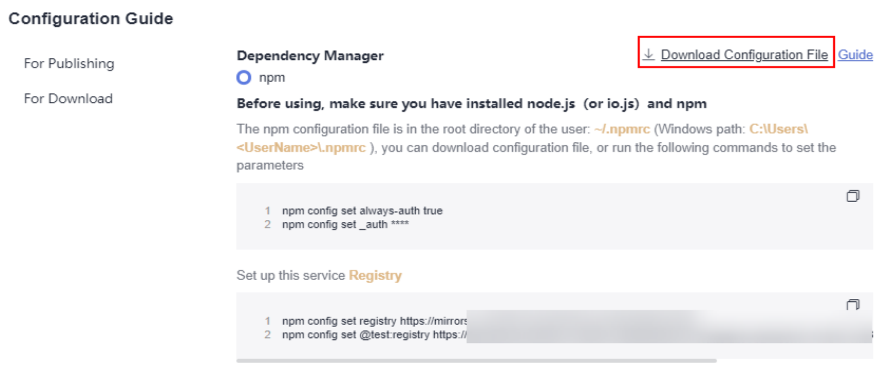
@test

Description

Enter up to 200 characters.

0/200

2. Click **Cancel** to return to the self-hosted repo page. Click **Set Me Up**.
3. In the displayed dialog box, click **Download Configuration File**.



Configuration Guide

Dependency Manager

npm

Before using, make sure you have installed node.js (or io.js) and npm

The npm configuration file is in the root directory of the user: ~/.npmrc (Windows path: C:\Users\<UserName>\.npmrc ), you can download configuration file, or run the following commands to set the parameters

```
npm config set always-auth true
npm config set _auth ****
```

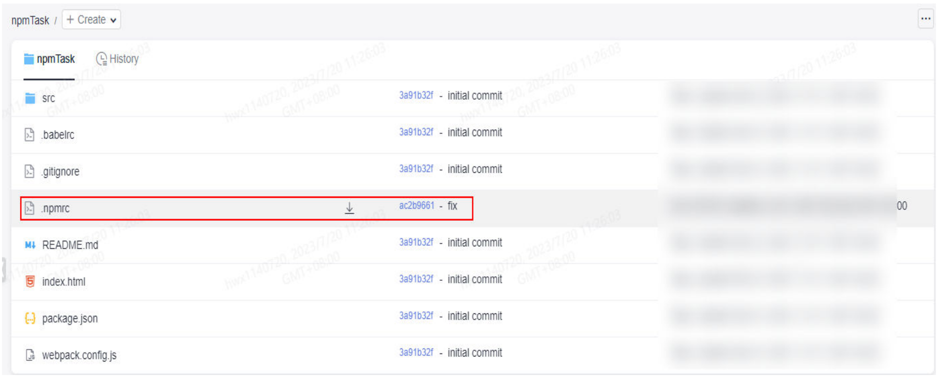
Set up this service Registry

```
npm config set registry https://mirrors[redacted]
npm config set @test:registry https://[redacted]
```

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

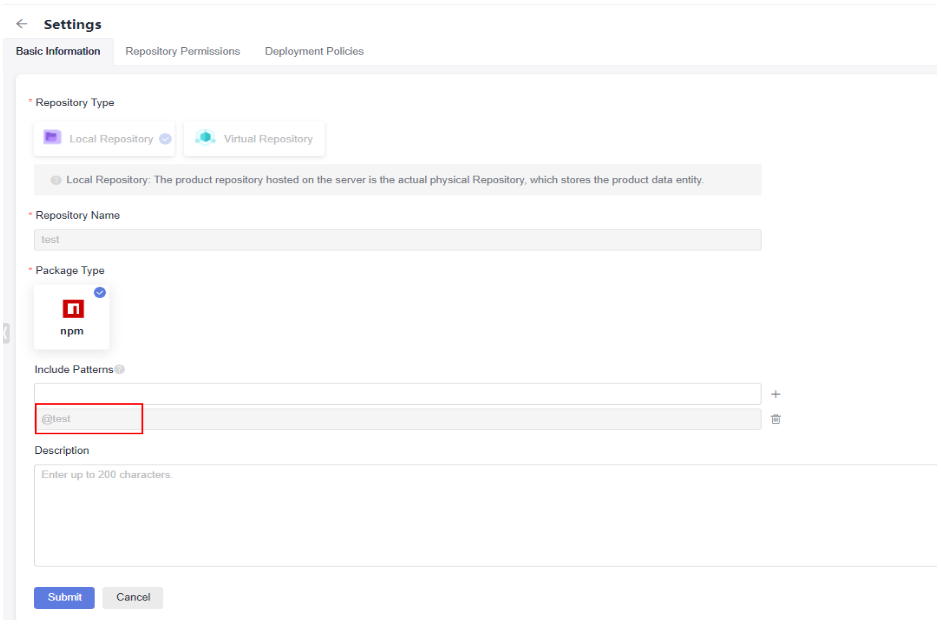
**Step 2** Configure a code repository.

1. Go to CodeArts Repo and create a **Node.js** repository. For details, see **Creating a Repository Using a Template**. This procedure uses the **Nodejs Webpack Demo** template.
2. Go to the code repository and upload the **.npmrc** file to the root directory of the **code repository**.



| | | |
|---|---|---|
| src | 3a91b32f - initial commit | [redacted] |
| .babelrc | 3a91b32f - initial commit | [redacted] |
| .gitignore | 3a91b32f - initial commit | [redacted] |
| .npmrc | ac2b9661 - fix | [redacted] |
| README.md | 3a91b32f - initial commit | [redacted] |
| index.html | 3a91b32f - initial commit | [redacted] |
| package.json | 3a91b32f - initial commit | [redacted] |
| webpack.config.js | 3a91b32f - initial commit | [redacted] |

3. Find the **package.json** file in the code repository and open it. Add the repository address recorded on the **Basic Information** on the **Settings** page to the **name** field in the file.



* Repository Type

Local Repository ✓ Virtual Repository

Local Repository: The product repository hosted on the server is the actual physical Repository, which stores the product data entity.

* Repository Name

test

* Package Type

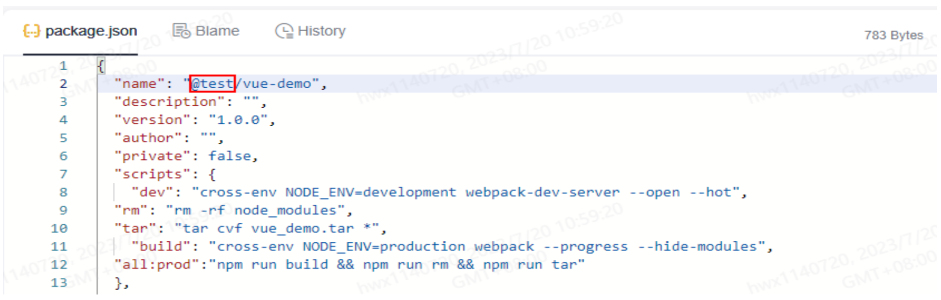
npm ✓

Include Patterns

@test

Description

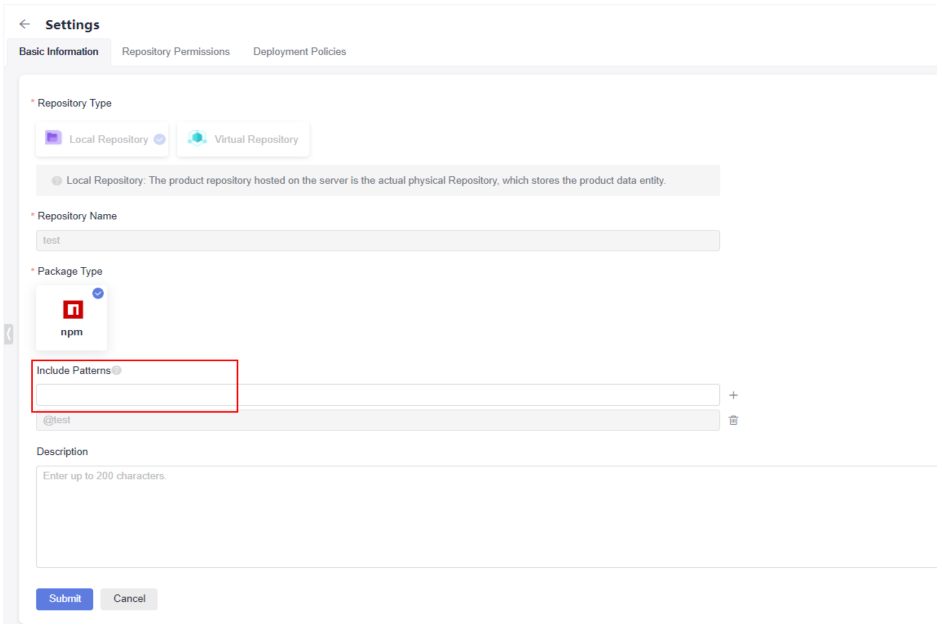
Enter up to 200 characters.



```
{
  "name": "@test/vue-demo",
  "description": "",
  "version": "1.0.0",
  "author": "",
  "private": false,
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --hot",
  "rm": "rm -rf node_modules",
  "tar": "tar cvf vue_demo.tar *",
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules",
  "all:prod":"npm run build && npm run rm && npm run tar"
  },
```

**NOTE**

If the **name** field cannot be modified, add the address to the **Include Patterns** field on the **Basic Information** on the **Settings** page.



* Repository Type

Local Repository ✓ Virtual Repository

Local Repository: The product repository hosted on the server is the actual physical Repository, which stores the product data entity.

* Repository Name

test

* Package Type

npm ✓

Include Patterns

@test

Description

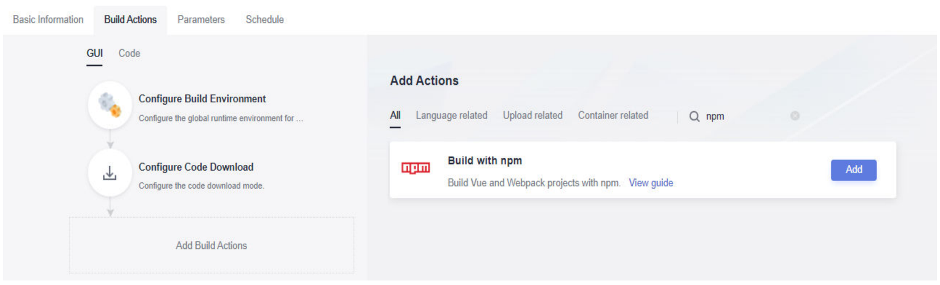
Enter up to 200 characters.

**Step 3** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

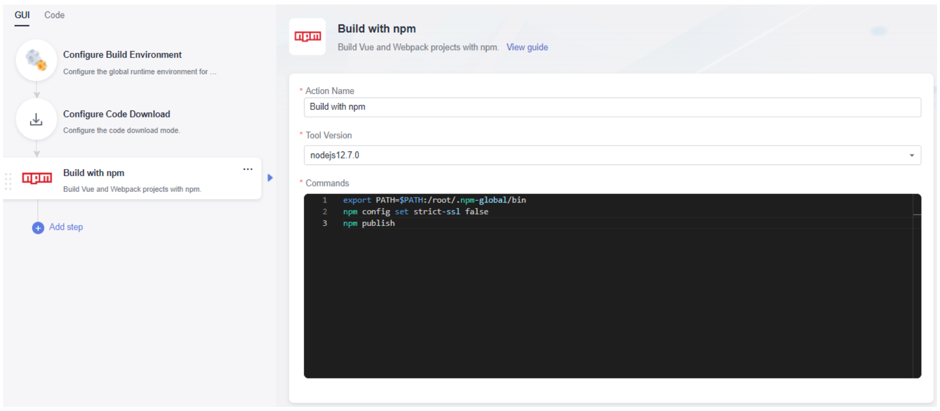
   Select **Blank Template** and click **Next**.

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



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

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


GUI
Code
Configure Build Environment
Configure the global runtime environment for ...
Configure Code Download
Configure the code download mode.
Build with npm
Build Vue and Webpack projects with npm.
Add step
Build with npm
Build Vue and Webpack projects with npm.
View guide

* Action Name: Build with npm

* Tool Version: nodejs12.7.0

* Commands

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

4. Click **Create and Run** to start the build task.

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

**----End**

## Obtaining a Dependency from a Self-hosted npm Repo

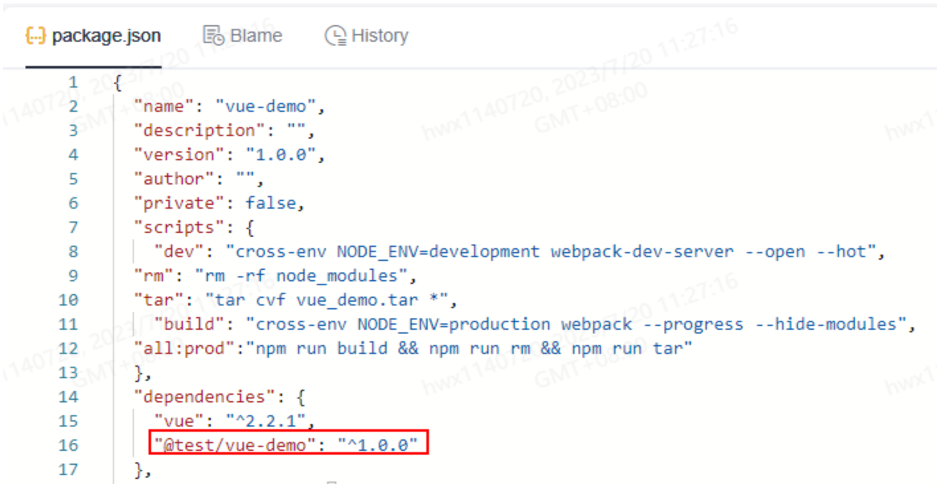
The following procedure uses the npm component released in **Releasing a Component to a Self-hosted npm Repo** as an example to describe how to obtain a dependency from a self-hosted npm repo.

**Step 1** Configure a code repository.

1. Go to CodeArts Repo and create a **Node.js** repository. For details, see **Creating a Repository Using a Template**. This procedure uses the **Nodejs Webpack Demo** template.
2. Obtain the **.npmrc** file (see **Releasing a Component to a Self-hosted npm Repo**) and upload it to the root directory of the code repository where the npm dependency is to be used.

3. Find and open the **package.json** file in the code repository, and configure the dependency to the **dependencies** field. In this document, the value is as follows:

```
"@test/vue-demo": "^1.0.0"
```



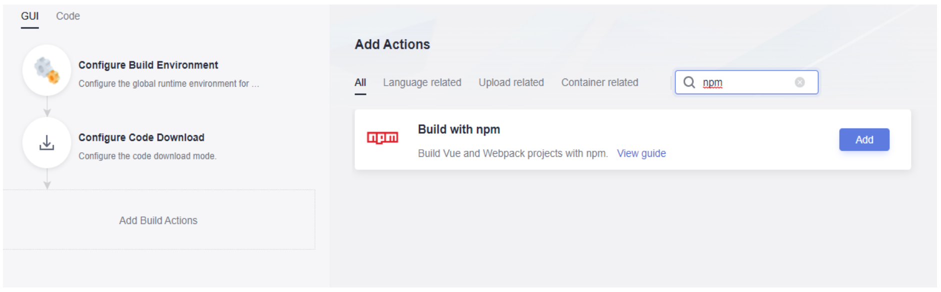
```
{
  "name": "vue-demo",
  "description": "",
  "version": "1.0.0",
  "author": "",
  "private": false,
  "scripts": {
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --hot",
  "rm": "rm -rf node_modules",
  "tar": "tar cvf vue_demo.tar *",
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules",
  "all:prod":"npm run build && npm run rm && npm run tar"
  },
  "dependencies": {
    "vue": "^2.2.1",
    "@test/vue-demo": "^1.0.0"
  },
```

**Step 2** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

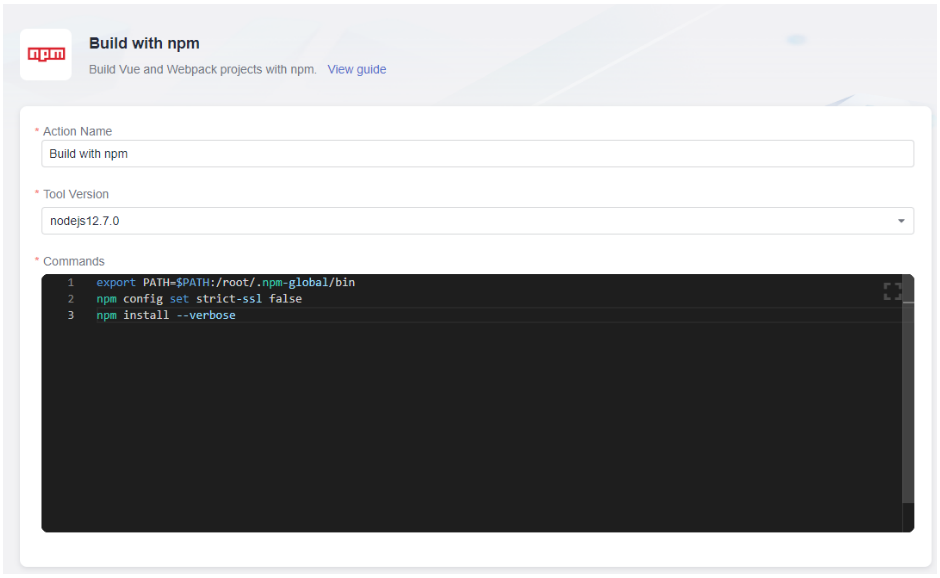
   Select **Blank Template** and click **Next**.

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


GUI
Code
Configure Build Environment
Configure the global runtime environment for ...
Configure Code Download
Configure the code download mode.
Add Build Actions
Add Actions
All
Language related
Upload related
Container related
npm
Build with npm
Build Vue and Webpack projects with npm.
View guide
Add

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

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



* Action Name
Build with npm

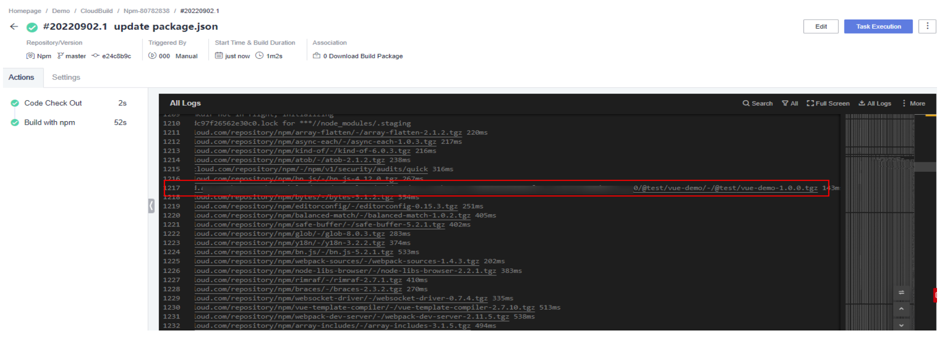
* Tool Version
nodejs12.7.0

* Commands

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

**Step 3** Click **Create and Run** to start the build task.

After the task is successfully executed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the self-hosted repo.



```
mkdir not in flight; initializing
ic97f26562e30c0.lock for ***//node_modules/.staging
loud.com/repository/npm/array-flatten/-/array-flatten-2.1.2.tgz 220ms
loud.com/repository/npm/async-each/-/async-each-1.0.3.tgz 217ms
loud.com/repository/npm/kind-of/-/kind-of-6.0.3.tgz 216ms
loud.com/repository/npm/atob/-/atob-2.1.2.tgz 238ms
:loud.com/repository/npm/-/npm/v1/security/audits/quick 316ms
loud.com/repository/npm/bn.js/-/bn.js-4.12.0.tgz 267ms
i.a[redacted]0/@test/vue-demo/-/@test/vue-demo-1.0.0.tgz 143ms
loud.com/repository/npm/bytes/-/bytes-3.1.2.tgz 354ms
loud.com/repository/npm/editorconfig/-/editorconfig-0.15.3.tgz 251ms
loud.com/repository/npm/balanced-match/-/balanced-match-1.0.2.tgz 405ms
loud.com/repository/npm/safe-buffer/-/safe-buffer-5.2.1.tgz 402ms
loud.com/repository/npm/glob/-/glob-8.0.3.tgz 283ms
loud.com/repository/npm/y18n/-/y18n-3.2.2.tgz 374ms
loud.com/repository/npm/bn.js/-/bn.js-5.2.1.tgz 533ms
loud.com/repository/npm/webpack-sources/-/webpack-sources-1.4.3.tgz 202ms
loud.com/repository/npm/node-libs-browser/-/node-libs-browser-2.2.1.tgz 383ms
loud.com/repository/npm/rimraf/-/rimraf-2.7.1.tgz 410ms
loud.com/repository/npm/braces/-/braces-2.3.2.tgz 270ms
loud.com/repository/npm/websocket-driver/-/websocket-driver-0.7.4.tgz 335ms
loud.com/repository/npm/vue-template-compiler/-/vue-template-compiler-2.7.10.tgz 513ms
loud.com/repository/npm/webpack-dev-server/-/webpack-dev-server-2.11.5.tgz 538ms
loud.com/repository/npm/array-includes/-/array-includes-3.1.5.tgz 494ms
```

**----End**

## npm Commands

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

- Delete an existing component from a self-hosted repo.
  ```
  npm unpublish @socpe/packageName@version
  ```
- Obtain tags.
  ```
  npm dist-tag list @scope/packageName
  ```
- Add a tag.
  ```
  npm dist-tag add @scope/packageName@version tagName --registry registryUrl --verbose
  ```
- Delete a tag.
  ```
  npm dist-tag rm @scope/packageName@version tagName --registry registryUrl --verbose
  ```

Command parameter description:

- **scope**: path of a self-hosted repo. For details about how to obtain the path, see **Releasing a Component to a Self-hosted npm Repo**.

- **packageName**: the part following **scope** in the **name** field of the **package.json** file.
- **version**: value of the **version** field in the **package.json** file.
- **registryUrl**: URL of the self-hosted repo referenced by **scope** in the configuration file.
- **tagName**: tag name.

The following uses the private component released in **Releasing a Component to a Self-hosted npm Repo** as an example:

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

The command for deleting this component is as follows:

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

# 5 Releasing/Obtaining a Go Component via a Build Task

This section describes how to release a component to a self-hosted Go repo via a build task and obtain a dependency from the repository for deployment.

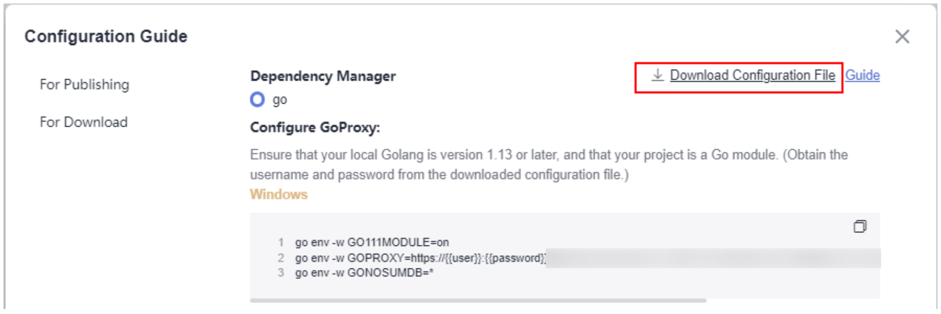
## Prerequisites

1. You already have a project. If no project is available, **create one**.
2. You have created a self-hosted Go repo.
3. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.

## Releasing a Component to a Self-hosted Go Repo

**Step 1** Download the configuration file of the self-hosted repo.

1. Log in to the CodeArts homepage and access the self-hosted repo for Go. Click **Set Me Up** on the right of the page.
2. In the displayed dialog box, click **Download Configuration File**.



**Dependency Manager**

go

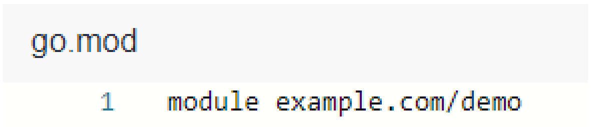
**Configure GoProxy:**

Ensure that your local Golang is version 1.13 or later, and that your project is a Go module. (Obtain the username and password from the downloaded configuration file.)

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://{{user}}:{{password}}[redacted]
go env -w GONOSUMDB=*
```

**Step 2** Configure a code repository.

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



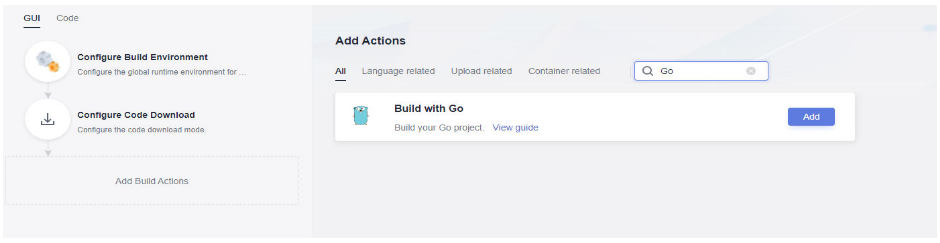
```
go.mod

module example.com/demo
```

**Step 3** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Blank Template** and click **Next**.

2. Add the **Build in Go** action.



3. Edit the **Build in Go** action.
   - Select the desired tool version. In this example, **go-1.13.1** is used.
   - Delete the existing commands, open the configuration file downloaded in **Step 1**, and copy the **commands for configuring Go environment variable in Linux** to the command box.
   - Copy the Go upload command segment in the configuration file to the command box, and replace the parameters in the commands by referring to **Go Module Packaging**. (In this example, the package version is **v1.0.0**.)

4. Click **Create and Run** to start the build task.

   When a message is displayed indicating **build successful**, go to the self-hosted repo and find the uploaded Go component.

**----End**

## Obtaining a Dependency from a Self-hosted Go Repo

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

**Step 1** Download the configuration file of the self-hosted repo by referring to **Releasing a Component to a Self-hosted Go Repo**.

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

**Step 3** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Blank Template** and click **Next**.

2. Add the **Build in Go** action.

3. Edit the **Build in Go** action.
   - Select the desired tool version. In this example, **go-1.13.1** is used.
   - Delete the existing commands, open the downloaded configuration file, and copy the **commands for configuring Go environment variables in Linux** to the command box.
   - Copy the **Go download commands** in the configuration file to the command box and replace the **<modulename>** parameter with the actual value. (In this example, the parameter is set to **example.com/ demo**).

**Step 4** Click **Create and Run** to start the build task.

When a message is displayed indicating **build successful**, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the self-hosted repo.

**----End**

## Go Module Packaging

This section describes how to build and upload Go components through Go module packaging.

Perform the following steps:

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

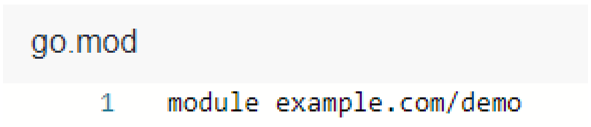
The component directory varies according to the package version. The version can be:

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

The following are examples of component directories for different versions:

- **Versions earlier than v2.0**

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



go.mod
```
module example.com/demo
```

a. Create a source folder in the working directory.

   The value of **module** is **example.com/demo** and that of **version** is **1.0.0**. The command is as follows:
   ```
   mkdir -p ~/example.com/demo@v1.0.0
   ```
b. Copy the code source to the source folder.

   The command is as follows (with the same parameter values as the previous command):
   ```
   cp -rf . ~/example.com/demo@v1.0.0/
   ```
c. Compress the component into a ZIP package.

   Run the following command to go to the upper-level directory of the root directory where the ZIP package is located:
   ```
   cd ~
   ```
   Then, use the **zip** command to compress the code into a component package. In this command, the **package root directory** is **example.com** and the **package name** is **v1.0.0.zip**. The command is as follows:
   ```
   zip -D -r v1.0.0.zip  example.com/
   ```
d. Upload the component ZIP package and the **go.mod** file to the self-hosted repo.

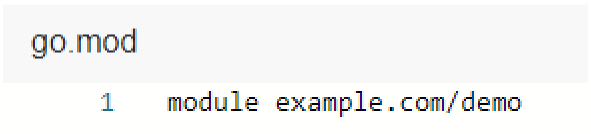
   Parameters **username**, **password**, and **repoUrl** can be obtained from the configuration file of the self-hosted repo.

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

   The command is as follows (replace **username**, **password**, and **repoUrl** with the actual values):
   ```
   curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/v1.0.0.zip -T
   v1.0.0.zip
   curl -u {{username}}:{{password}} -X PUT {{repoUrl}}/example.com/demo/@v/v1.0.0.mod -T
   example.com/demo@v1.0.0/go.mod
   ```

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

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



go.mod
```
module example.com/demo
```

a. Create a source folder in the working directory.

   The value of **module** is **example.com/demo/v2** and that of **version** is **2.0.0**. The command is as follows:
   ```
   mkdir -p ~/example.com/demo/v2@v2.0.0
   ```
b. Copy the code source to the source folder.

   The command is as follows (with the same parameter values as the previous command):
   ```
   cp -rf . ~/example.com/demo/v2@v2.0.0/
   ```
c. Compress the component into a ZIP package.

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

```
cd ~
```

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

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

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

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

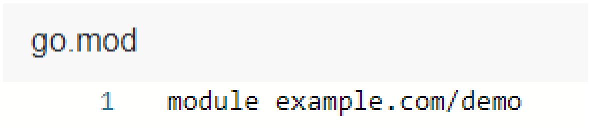
   - For the ZIP package, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.zip** and that of **localFile** is **v2.0.0.zip**.
   - For the **go.mod** file, the value of **filePath** is **example.com/demo/v2/@v/v2.0.0.mod** and that of **localFile** is **example.com/demo/v2@v2.0.0/go.mod**.

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

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

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

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



go.mod

```
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 component into a ZIP package.

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

```
cd ~
```

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

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

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

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

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

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

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

# 6 Releasing/Obtaining a PyPI Component via a Build Task

This section describes how to release a component to a self-hosted PyPI repo via a build task and obtain a dependency from the repository for deployment.

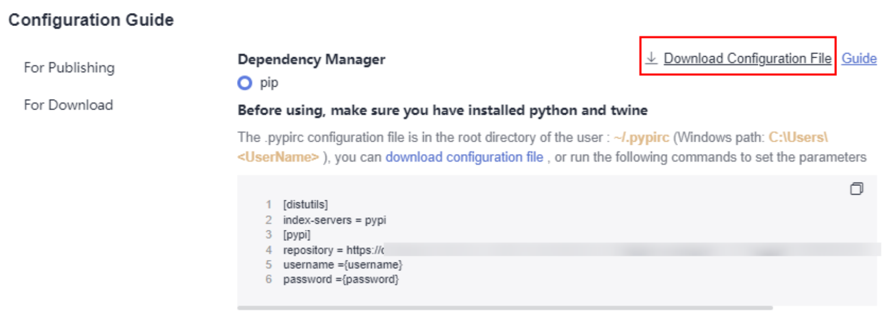
## Prerequisites

1. You already have a project. If no project is available, **create one**.
2. You have created a self-hosted PyPI repo.
3. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.

## Releasing a Component to a Self-hosted PyPI Repo

**Step 1** Download the configuration file of the self-hosted repo.

1. Log in to the CodeArts homepage and access the self-hosted repo for PyPI. Click **Set Me Up** on the right of the page.
2. In the displayed dialog box, find the **For Publishing** and click **Download Configuration File**.



Before using, make sure you have installed python and twine

The .pypirc configuration file is in the root directory of the user : ~/.pypirc (Windows path: C:\Users\<UserName> ), you can download configuration file , or run the following commands to set the parameters

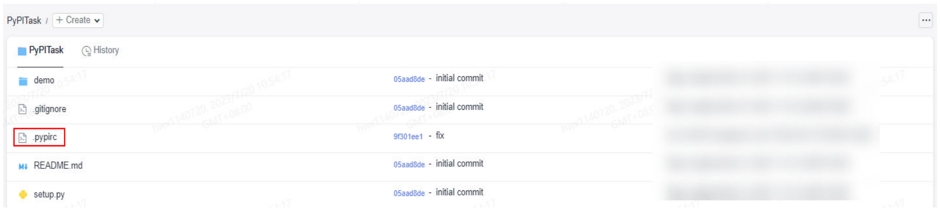
```
[distutils]
index-servers = pypi
[pypi]
repository = https://[redacted]
username ={username}
password ={password}
```

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

**Step 2** Configure a code repository.

1. Go to CodeArts Repo and create a Python repository. For details, see **Creating a Repository**. This procedure uses the **Python3 Demo** template.

2. Go to the code repository and upload the **.pypirc** file to the root directory of the **code repository**.

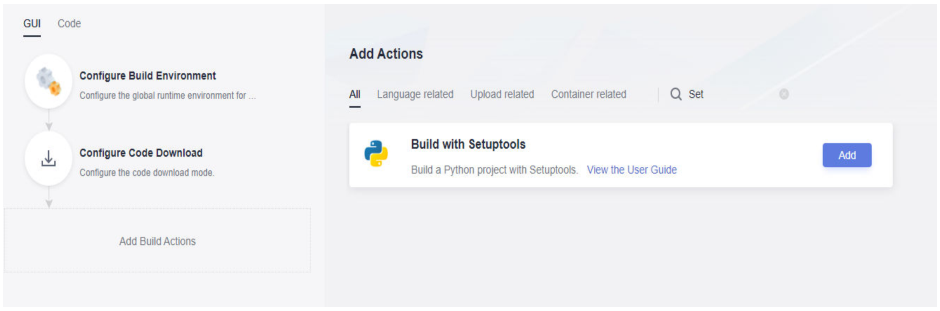


**Step 3** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

   Select **Blank Template** and click **Next**.

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


GUI
Code
Configure Build Environment
Configure the global runtime environment for ...
Configure Code Download
Configure the code download mode.
Add Build Actions
Add Actions
All
Language related
Upload related
Container related
Set
Build with Setuptools
Build a Python project with Setuptools. View the User Guide
Add

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

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

   – Delete the existing commands and run the following instead:

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

NOTE

If certificate verification fails during the upload, add the following command to the first line of the preceding command to skip certificate verification:

export CURL_CA_BUNDLE=""

4. Click **Create and Run** to start the build task.

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

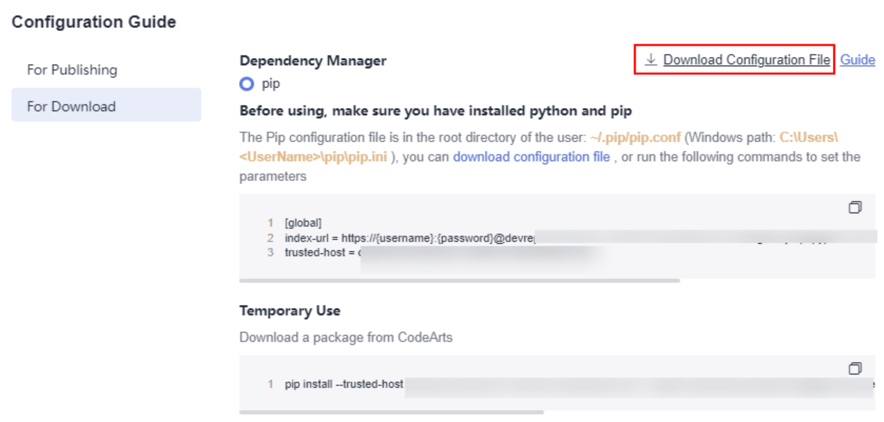
**----End**

## Obtaining a Dependency from a Self-hosted PyPI Repo

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

**Step 1** Download the configuration file of the self-hosted repo.

1. Go to the self-hosted PyPI repo. Click **Set Me Up** on the right of the page.
2. In the displayed dialog box, find the **For Download** and click **Download Configuration File**.



Configuration Guide

Dependency Manager

pip

Before using, make sure you have installed python and pip

The Pip configuration file is in the root directory of the user: ~/.pip/pip.conf (Windows path: C:\Users\<UserName>\pip\pip.ini ), you can download configuration file , or run the following commands to set the parameters

```
[global]
index-url = https://{username}:{password}@devre[redacted]
trusted-host = [redacted]
```

Temporary Use

Download a package from CodeArts

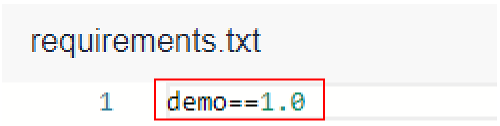
```
pip install --trusted-host [redacted]
```

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

**Step 2** Configure a code repository.

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

```
demo ==1.0
```



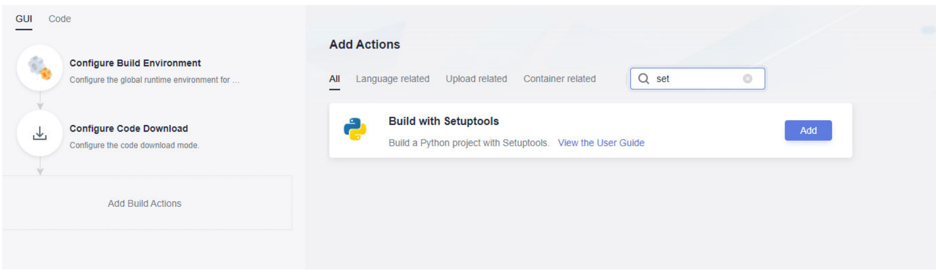
requirements.txt

```
demo==1.0
```

**Step 3** Configure and execute a build task.

1. On the code repository page, click **Create Build Task** in the upper right. The **Create Build Task** page is displayed.

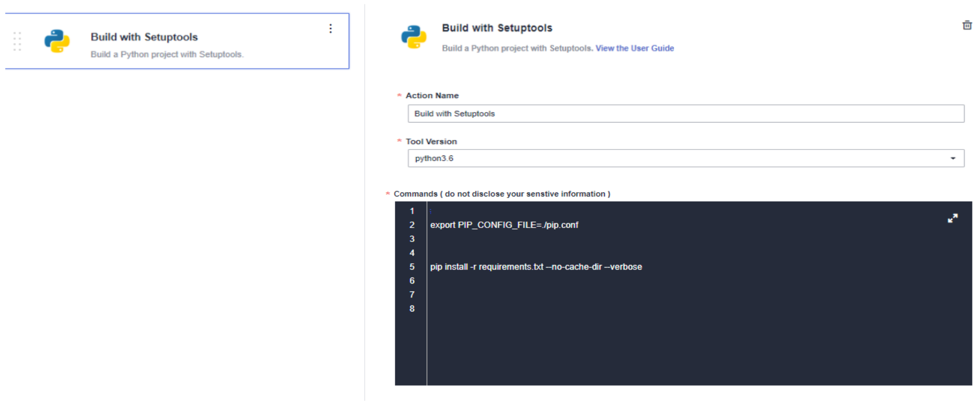
   Select **Blank Template** and click **Next**.
2. Add the **Build with Setuptools** action.



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

```
# Set the pip.conf file in the root directory of the current project as the configuration file.
 export PIP_CONFIG_FILE=./pip.conf
```

```
# Download the PyPI component.
 pip install -r requirements.txt --no-cache-dir
```


Build with Setuptools
Build a Python project with Setuptools.
Build with Setuptools
Build a Python project with Setuptools. View the User Guide
* Action Name
* Tool Version
* Commands ( do not disclose your sensitive information )

Action Name: Build with Setuptools

Tool Version: python3.6

```
export PIP_CONFIG_FILE=./pip.conf


pip install -r requirements.txt --no-cache-dir --verbose
```

**Step 4** Click **Create and Run** to start the build task.

After the task is successfully executed, view the task details. If information similar to the following is found in the log, the dependency has been downloaded from the self-hosted repo.

**----End**

# 7 Uploading/Obtaining an RPM Component Using Linux Commands

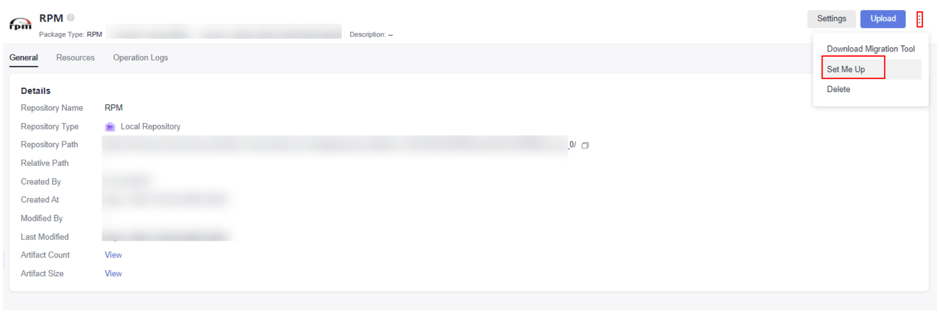
This section describes how to use Linux commands to upload a component to a self-hosted RPM repo and obtain a dependency from the repository.

## Prerequisites

1. An RPM component is available.
2. A Linux host that can connect to the public network is available.
3. You have created a self-hosted RPM repo.
4. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.

## Releasing a Component to a Self-hosted RPM Repo

**Step 1** Log in to the CodeArts homepage and access the self-hosted repo for RPM. Click **Set Me Up** on the right of the page.



Package Type: RPM [redacted] Description: --

Details

| | |
|---|---|
| Repository Name | RPM |
| Repository Type | Local Repository |
| Repository Path | [redacted]0/ |
| Relative Path | |
| Created By | [redacted] |
| Created At | [redacted] |
| Modified By | |
| Last Modified | [redacted] |
| Artifact Count | View |
| Artifact Size | View |

**Step 2** In the displayed dialog box, click **Download Configuration File**.

**Step 3** On the Linux host, run the following command to upload an RPM component:

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

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

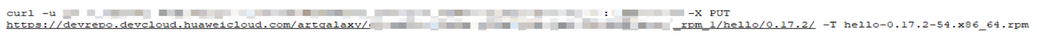
- *user*: character string before the colon (:) between **curl -u** and **-X**

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

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

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

The following figure shows the complete command.



```
curl -u [redacted] : [redacted] -X PUT
https://devrepo.devcloud.huaweicloud.com/artgalaxy/[redacted]_rpm_1/hello/0.17.2/ -T hello-0.17.2-54.x86_64.rpm
```

**Step 4** After the command is successfully executed, go to the self-hosted repo and find the uploaded RPM component.

**----End**

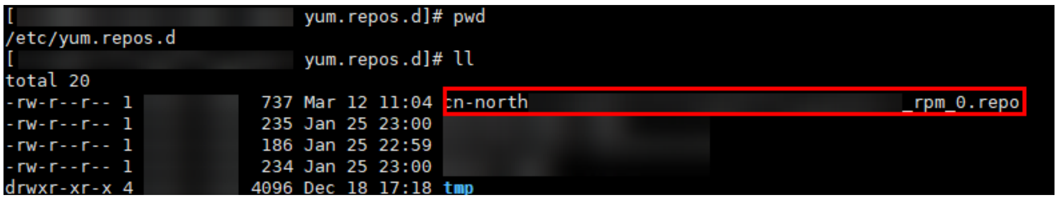
## Obtaining a Dependency from a Self-hosted RPM Repo

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

**Step 1** Download the configuration file of the self-hosted RPM repo by referring to **Releasing a Component to a Self-hosted RPM Repo**.

**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.



```
[[redacted] yum.repos.d]# pwd
/etc/yum.repos.d
[[redacted] yum.repos.d]# ll
total 20
-rw-r--r-- 1 [redacted]  737 Mar 12 11:04 cn-north[redacted]_rpm_0.repo
-rw-r--r-- 1             235 Jan 25 23:00 [redacted]
-rw-r--r-- 1             186 Jan 25 22:59
-rw-r--r-- 1             234 Jan 25 23:00
drwxr-xr-x 4            4096 Dec 18 17:18 tmp
```

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

```
yum install hello
```

**----End**

# 8 Uploading/Obtaining a Debian Component Using Linux Commands

This section describes how to use Linux commands to upload a component to a self-hosted Debian repo and obtain a dependency from the repository.

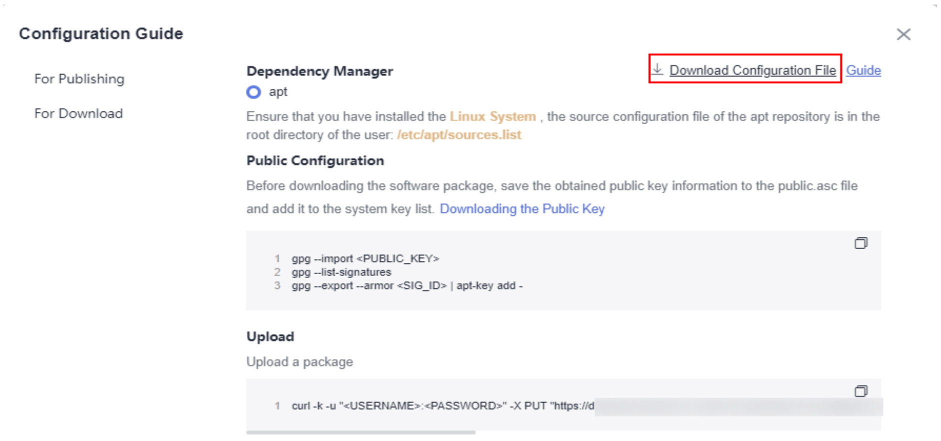
## Prerequisites

1. A Debian component is available.
2. A Linux host that can connect to the public network is available.
3. You have created a self-hosted Debian repo.
4. You have permissions for the current self-hosted repo. For details, see **Managing User Permissions**.

## Releasing a Component to a Self-hosted Debian Repo

**Step 1** Log in to the CodeArts homepage and access the self-hosted repo for Debian. Click **Set Me Up** on the right of the page.

**Step 2** In the displayed dialog box, click **Download Configuration File**.



Dependency Manager

apt

Ensure that you have installed the Linux System , the source configuration file of the apt repository is in the root directory of the user: /etc/apt/sources.list

Public Configuration

Before downloading the software package, save the obtained public key information to the public.asc file and add it to the system key list. Downloading the Public Key

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

Upload

Upload a package

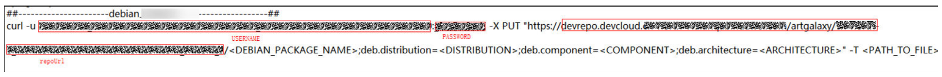
```
curl -k -u "<USERNAME>:<PASSWORD>" -X PUT "https://d[redacted]
```

**Step 3** On the Linux host, run the following command to upload a Debian component:

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

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

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



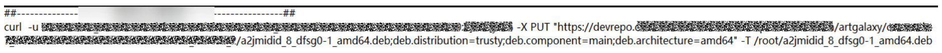
```
##-----------------------debian_[redacted]-----------------##
curl -u [redacted]:[redacted] -X PUT "https://devrepo.devcloud.[redacted]/artgalaxy/[redacted]/<DEBIAN_PACKAGE_NAME>;deb.distribution=<DISTRIBUTION>;deb.component=<COMPONENT>;deb.architecture=<ARCHITECTURE>" -T <PATH_TO_FILE>
```

**DEBIAN_PACKAGE_NAME**, **DISTRIBUTION**, **COMPONENT**, and **ARCHITECTURE** can be obtained from the Debian component.

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

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

The following figure shows the complete command.



```
##---------------[redacted]----------------##
curl  -u [redacted]:[redacted] -X PUT "https://devrepo.[redacted]/artgalaxy/[redacted]/a2jmidid_8_dfsg0-1_amd64.deb;deb.distribution=trusty;deb.component=main;deb.architecture=amd64" -T /root/a2jmidid_8_dfsg0-1_amd64.deb
```

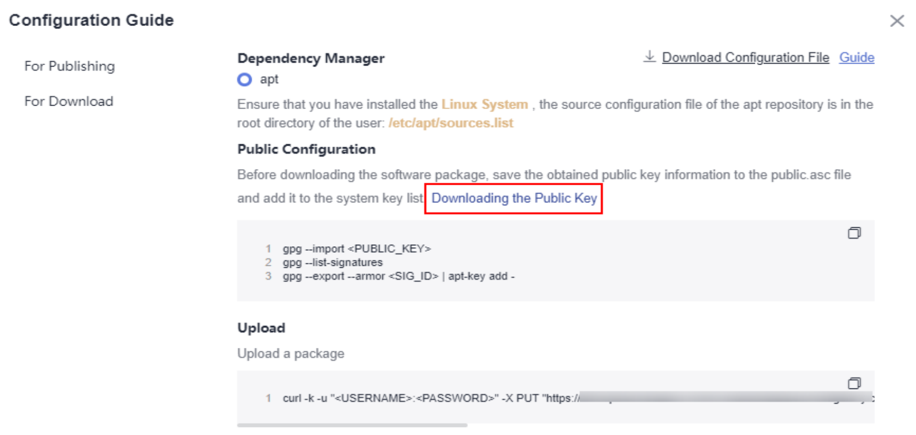
**Step 4** After the command is successfully executed, go to the self-hosted repo and find the uploaded Debian component.

**----End**

## Obtaining a Dependency from a Self-hosted Debian Repo

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

**Step 1** Download the **public key** file of the self-hosted Debian repo by referring to **Releasing a Component to a Self-hosted Debian Repo**.


Configuration Guide

For Publishing

For Download

**Dependency Manager**

Download Configuration File Guide

apt

Ensure that you have installed the Linux System , the source configuration file of the apt repository is in the root directory of the user: /etc/apt/sources.list

**Public Configuration**

Before downloading the software package, save the obtained public key information to the public.asc file and add it to the system key list. Downloading the Public Key

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

**Upload**

Upload a package

```
curl -k -u "<USERNAME>:<PASSWORD>" -X PUT "https://[redacted]
```

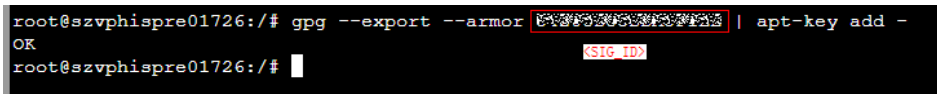
**Step 2** Import the gpg public key.

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

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

**Step 3** Add the public key to the list of keys used by apt to authenticate packages.

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

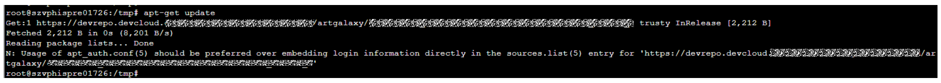

```
root@szvphispre01726:/# gpg --export --armor [redacted] | apt-key add -
OK
root@szvphispre01726:/#
```

**Step 4** Add the apt repository source.

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

**Step 5** After the repository source is added, run the following command to update the repository source:

```
apt-get update
```


```
root@szvphispre01726:/tmp# apt-get update
Get:1 https://devrepo.devcloud.[redacted]/artgalaxy/[redacted] trusty InRelease [2,212 B]
Fetched 2,212 B in 0s (8,201 B/s)
Reading package lists... Done
N: Usage of apt_auth.conf(5) should be preferred over embedding login information directly in the sources.list(5) entry for 'https://devrepo.devcloud[redacted]/artgalaxy/[redacted]'
root@szvphispre01726:/tmp#
```

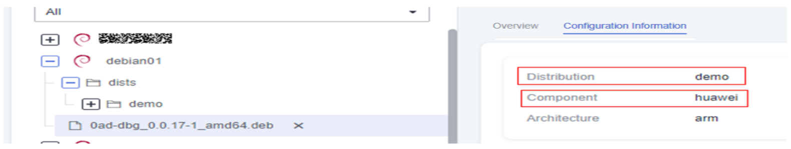
**Step 6** Run the following command to download the Debian package: Replace **a2jmidid** with the actual value of **PACKAGE**.

```
apt download a2jmidid
```

NOTE

Method for obtaining packages:

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



[redacted]
debian01
dists
demo
0ad-dbg_0.0.17-1_amd64.deb

| | |
|---|---|
| Distribution | demo |
| Component | huawei |
| Architecture | arm |

**----End**