CodeArts Artifact

Getting Started

Issue	02
Date	2023-04-23

HUAWEI TECHNOLOGIES CO., LTD.

Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Getting Started with a Release Repo..... 1
2 Getting Started with a Self-Hosted Repo.....4
3 Releasing/Obtaining a Self-Hosted Maven Component via a Build Task.....6
4 Releasing/Obtaining a Self-Hosted npm Component via a Build Task.....11
5 Releasing/Obtaining a Self-Hosted Go Component via a Build Task.....17
6 Releasing/Obtaining a Self-Hosted PyPI Component via a Build Task.....25
7 Uploading/Obtaining an RPM Component Using Linux Commands.....30
8 Uploading/Obtaining a Debian Component Using Linux Commands.....34

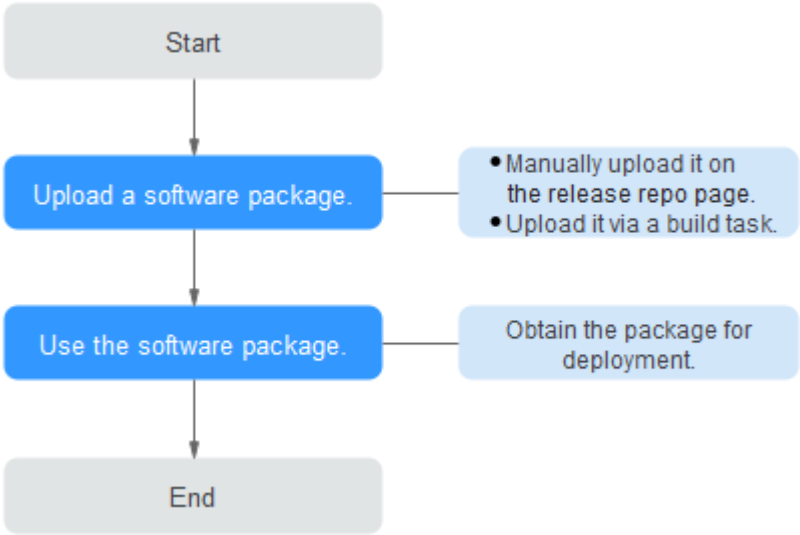
1 Getting Started with a Release Repo

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



Manually Uploading a Software Package on the Release Repo Page

Step 1 Log in to CodeArts, choose **Services** > **CodeArts 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

 Copyright © Huawei Technologies Co., Ltd.

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 **Cloud Repository Creation**. This procedure uses the **Java Web Demo** template.

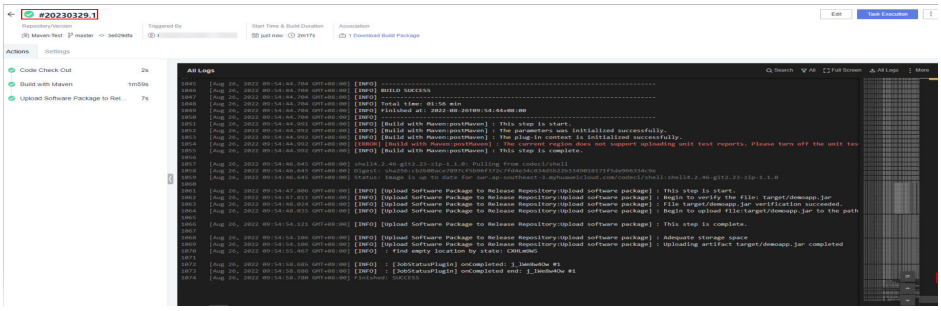
Step 2 Configure and execute a build task.

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

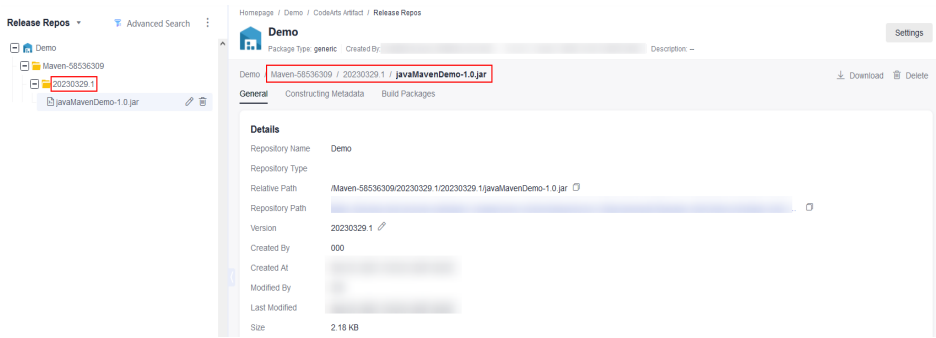
Figure 1-2 Build task



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



Copyright © Huawei Technologies Co., Ltd.

NOTE

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

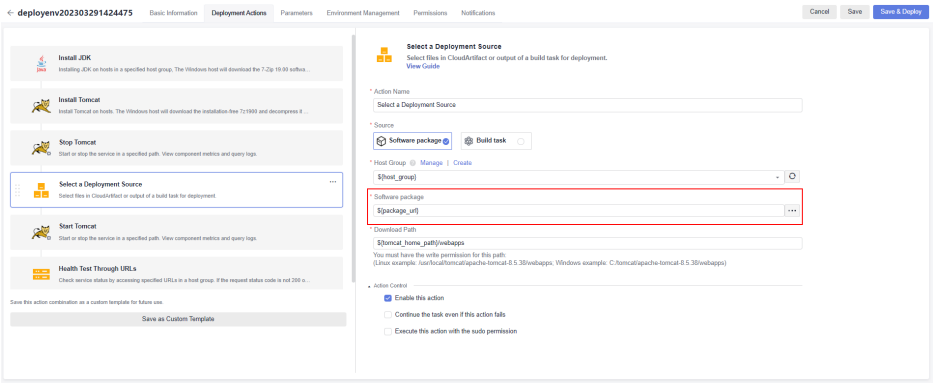
----End

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.

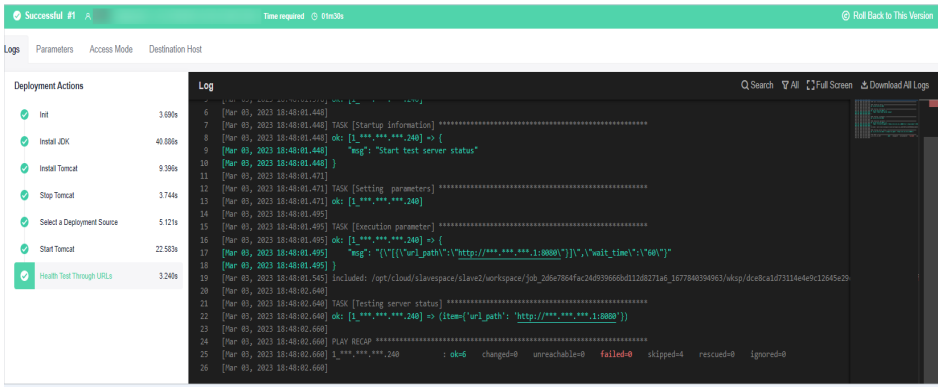
Step 1 **Create a deployment task** and configure it as required. In this example, configure it as follows:

- 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 2 On the **Environment Management** tab page, add a host group. For details, see **Creating a Host Group and Adding Trusted Hosts to the Host Group**.

Step 3 Click **Save & Deploy**. Execute the deployment task. When the task status becomes successful, the task has obtained the software package from the release repo and has deployed it on the target host.



----End

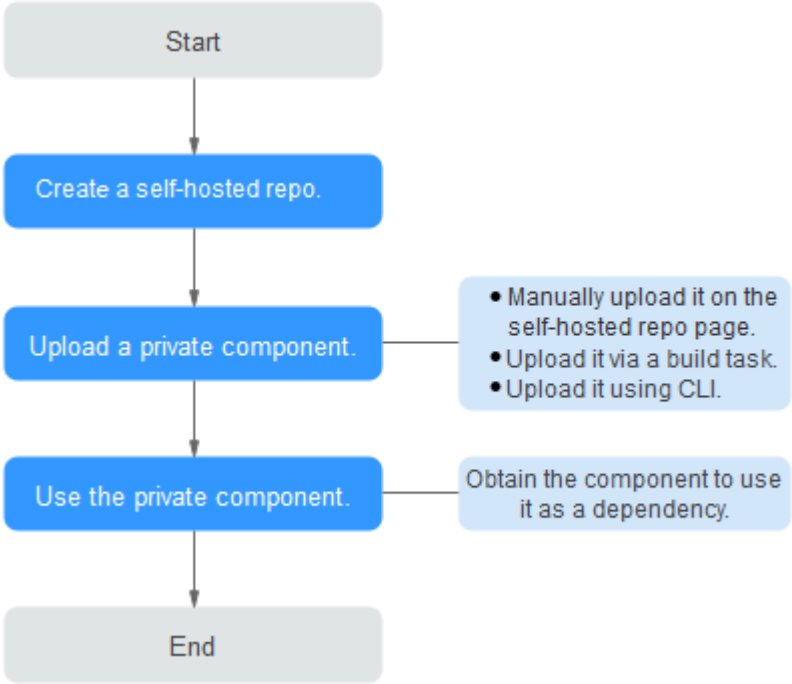
Copyright © Huawei Technologies Co., Ltd.

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



Creating a Self-Hosted Repo

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

Step 2 Click **Create Repository** on the left.

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

 Copyright © Huawei Technologies Co., Ltd.

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

----End

Uploading a Self-Hosted 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 Self-Hosted Component**.

----End

Uploading/Obtaining a Self-Hosted 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:

- **3 Releasing/Obtaining a Self-Hosted Maven Component via a Build Task**
- **4 Releasing/Obtaining a Self-Hosted npm Component via a Build Task**
- **5 Releasing/Obtaining a Self-Hosted Go Component via a Build Task**
- **6 Releasing/Obtaining a Self-Hosted 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 **7 Uploading/Obtaining an RPM Component Using Linux Commands**.

For details, see **8 Uploading/Obtaining a Debian Component Using Linux Commands**.

 Copyright © Huawei Technologies Co., Ltd.

3 Releasing/Obtaining a Self-Hosted 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 Maven repository 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 **Code** > **CodeArts Repo** on the top navigation bar.
2. Create a Maven repository. For details, see **Cloud Repository Creation**. This procedure uses the **Java Maven Demo** template.
3. Go to the code repository and view the component configuration in the **pom.xml** file.

 Copyright © Huawei Technologies Co., Ltd.

pom.xml

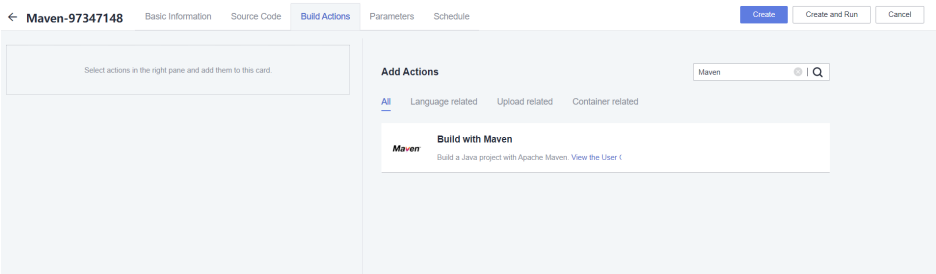
```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>com.huawei.demo</groupId>
4    <artifactId>javaMavenDemo</artifactId>
5    <packaging>jar</packaging>
6    <version>1.0</version>
7    <name>maven_demo</name>
8    <url>http://maven.apache.org</url>
9    <dependencies>
10     <dependency>
11     <groupId>junit</groupId>
12     <artifactId>junit</artifactId>
13     <version>3.8.1</version>
14     <scope>test</scope>
15     </dependency>
16   </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 Task** page is displayed.

   Select **Blank build template** and click **OK**.

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

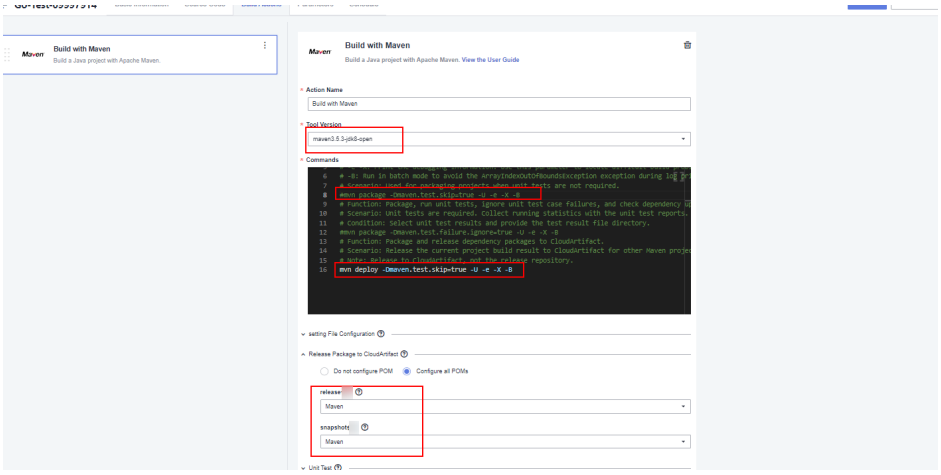


3. Edit the **Build with Maven** action.
   - Select the desired tool version. In this example, **maven3.5.3-jdk8-open** is used.
   - Find the following command and delete **#** in front of this command:
     ```
     #mvn deploy -Dmaven.test.skip=true -U -e -X -B
     ```
     Find the following command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

     NOTE

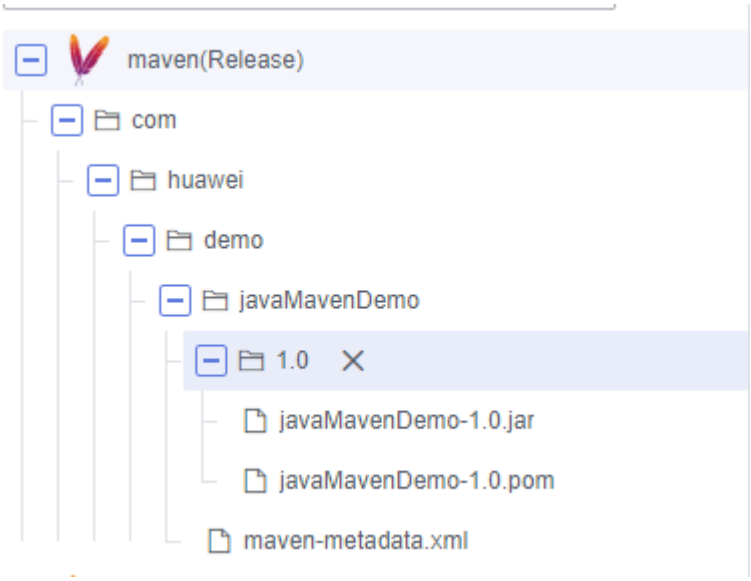
     If no option is available in the drop-down list, associate the Maven repository with the project of the build task by referring to **Managing the Association Between Repositories and Projects**.

 Copyright © Huawei Technologies Co., Ltd.



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 Maven repository 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.

 Copyright © Huawei Technologies Co., Ltd.

```
javaMavenDemo-1.0.pom
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <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">
4    <modelVersion>4.0.0</modelVersion>
5    <groupId>com.huawei.demo</groupId>
6    <artifactId>javaMavenDemo</artifactId>
7    <packaging>jar</packaging>
8    <version>1.0</version>
9    <name>maven_demo</name>
10   <url>http://maven.apache.org</url>
11   <dependencies>
12     <dependency>
13       <groupId>junit</groupId>
14       <artifactId>junit</artifactId>
15       <version>3.8.1</version>
16       <scope>test</scope>
17     </dependency>
18   </dependencies>
```

3. Go to CodeArts Repo. Create a Maven repository. For details, see **Cloud Repository Creation**. 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**).

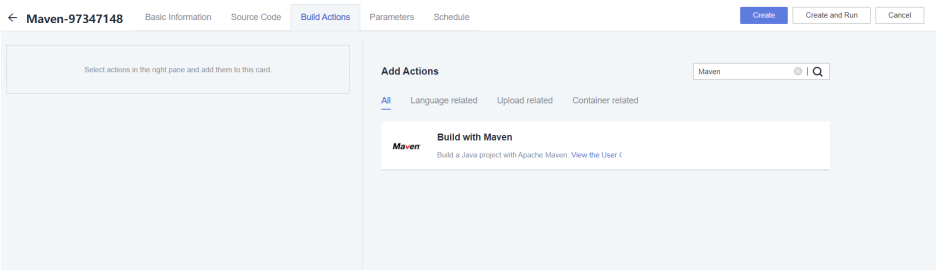
```
pom.xml                                  1.41 KB
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>com.huawei.demo</groupId>
4    <artifactId>javaMavenDemo</artifactId>
5    <packaging>jar</packaging>
6    <version>2.0</version>
7    <name>maven_demo</name>
8    <url>http://maven.apache.org</url>
9    <dependencies>
10     <dependency>
11     <groupId>junit</groupId>
12     <artifactId>junit</artifactId>
13     <version>3.8.1</version>
14     <scope>test</scope>
15     </dependency>
16     <dependency>
17     <groupId>com.huawei.demo</groupId>
18     <artifactId>javaMavenDemo</artifactId>
19     <version>1.0</version>
20     </dependency>
21   </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 Task** page is displayed.

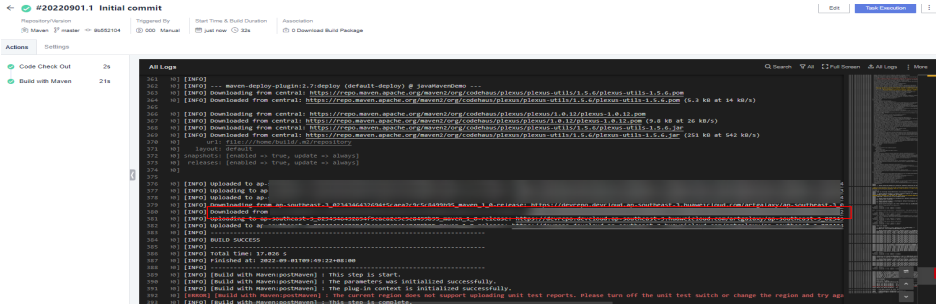
   Select **Blank build template** and click **OK**.
2. Add the **Build with Maven** action.

 Copyright © Huawei Technologies Co., Ltd.



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.



----End

Copyright © Huawei Technologies Co., Ltd.

4 Releasing/Obtaining a Self-Hosted 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.

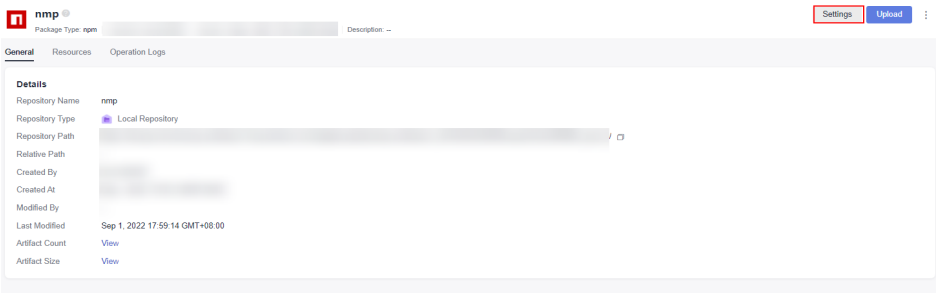
Prerequisites

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**.

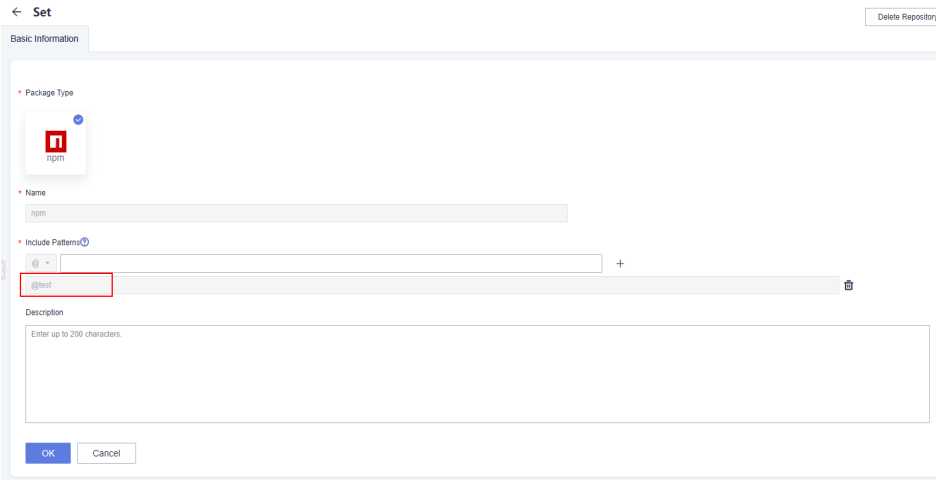
Releasing a Component to a Self-Hosted npm Repo

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

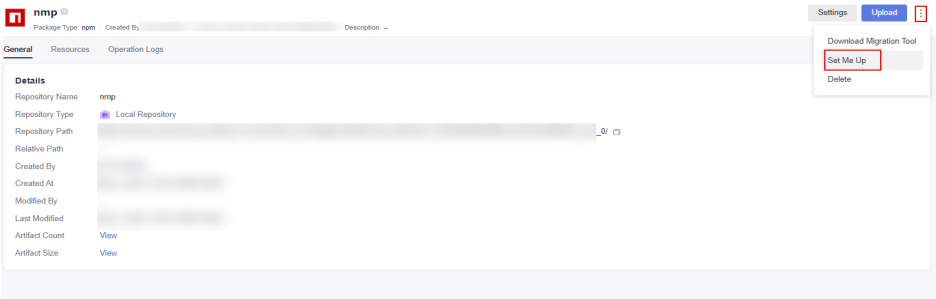
1. Log in to the CodeArts homepage and access the npm self-hosted repo for npm. Click **Settings** in the upper right corner and record the repository address.



 Copyright © Huawei Technologies Co., Ltd.



2. Click **Cancel** to return to the self-hosted repo page. Click ⋮ on the right of the page and choose **Set Me Up** from the drop-down list.



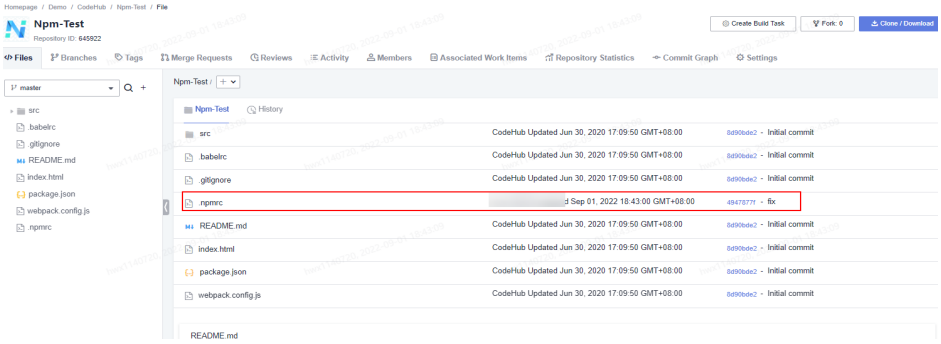
3. In the dialog box that is displayed, click **Download Configuration File**.

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

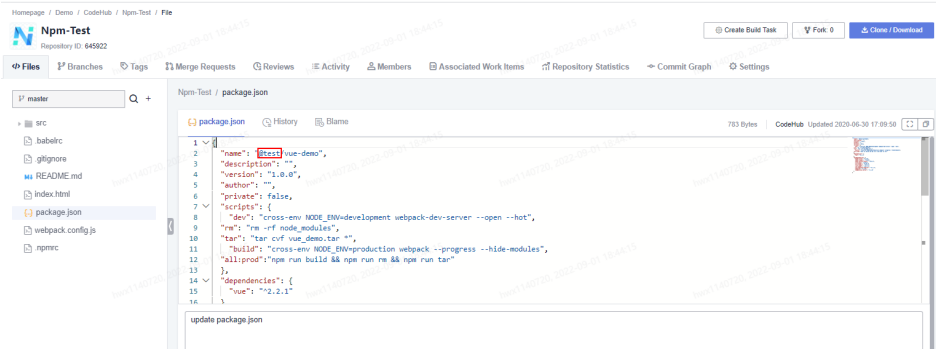
Step 2 Configure a code repository.

1. Go to CodeArts Repo and create a Node.js repository. For details, see **Cloud Repository Creation**. 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**.

 Copyright © Huawei Technologies Co., Ltd.



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



NOTE

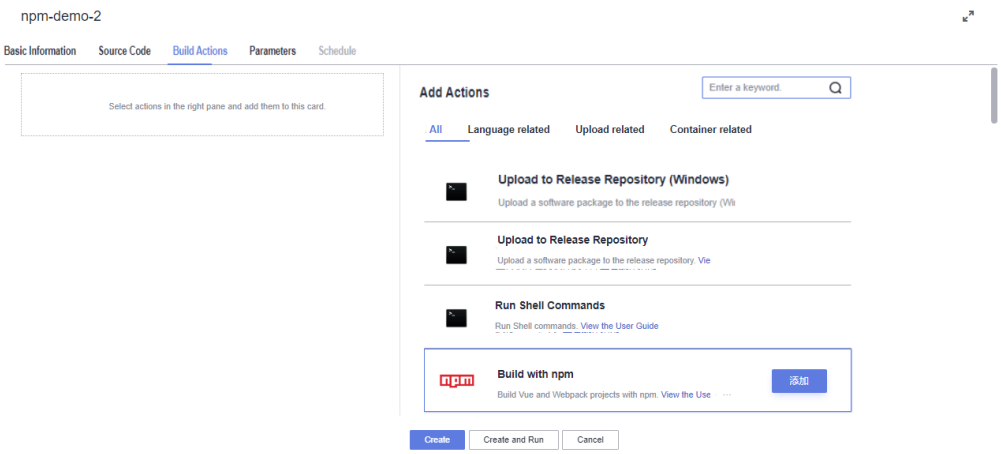
If the **name** field cannot be modified, add the address to the **Include Patterns** field on the **Edit Repository** page.

Step 3 Configure and execute a build task.

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

   Select **Blank build template** and click **OK**.

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

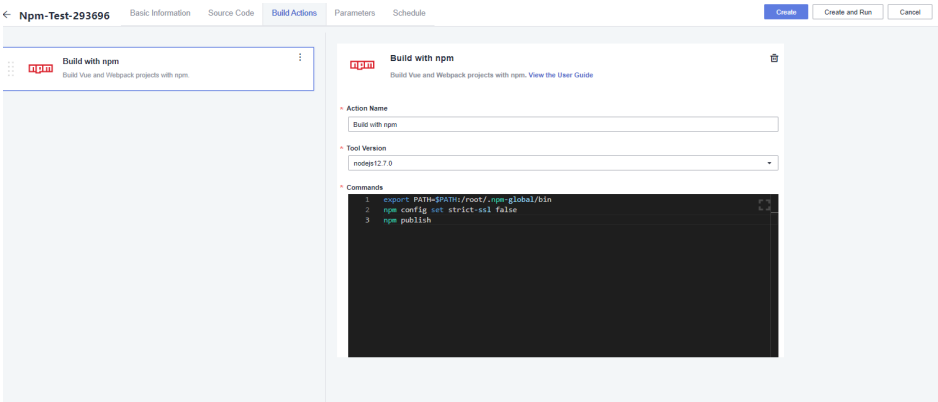


3. Edit the **Build with npm** action.
   - Select the desired tool version. In this example, **nodejs12.7.0** is used.

 Copyright © Huawei Technologies Co., Ltd.

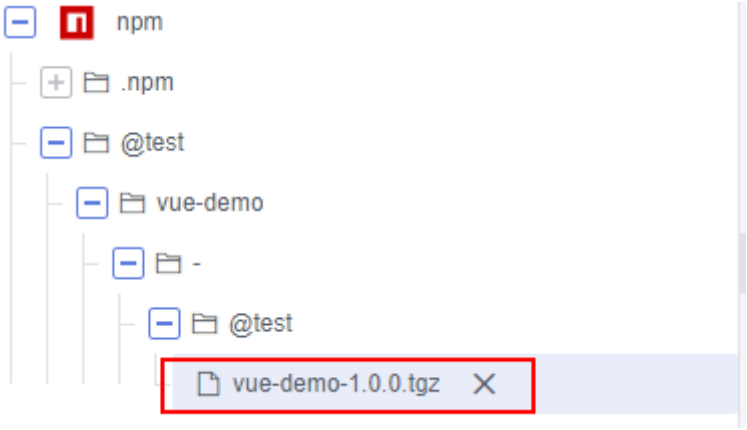
– Delete the existing commands and run the following instead:

```
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

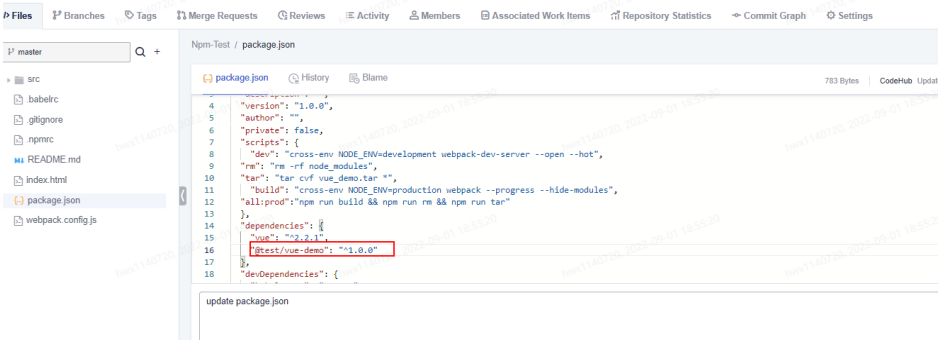
The following procedure uses the npm component released in **Releasing a Component to a Self-Hosted npm Repo** as an example to describe how to obtain a dependency from a self-hosted npm repo.

Step 1 Configure a code repository.

1. Go to CodeArts Repo and create a Node.js repository. For details, see **Cloud Repository Creation**. 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"
```

 Copyright © Huawei Technologies Co., Ltd.

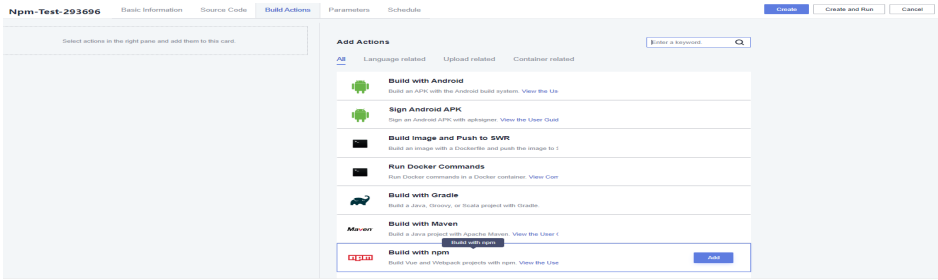


Step 2 Configure and execute a build task.

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

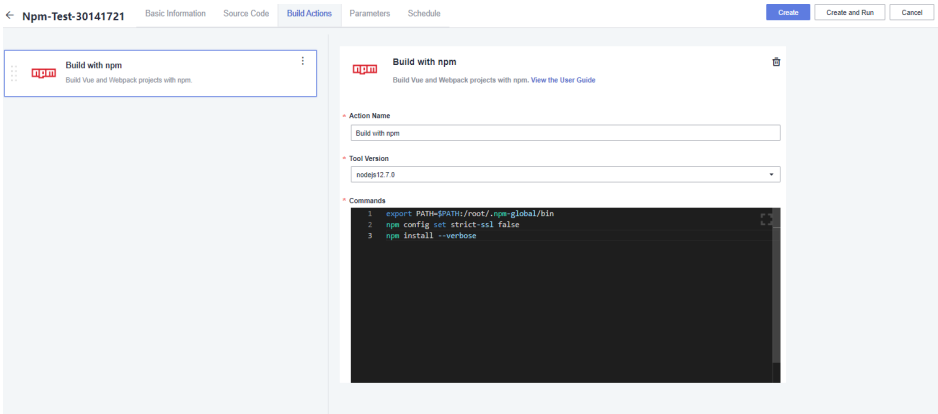
   Select **Blank build template** and click **OK**.

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



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

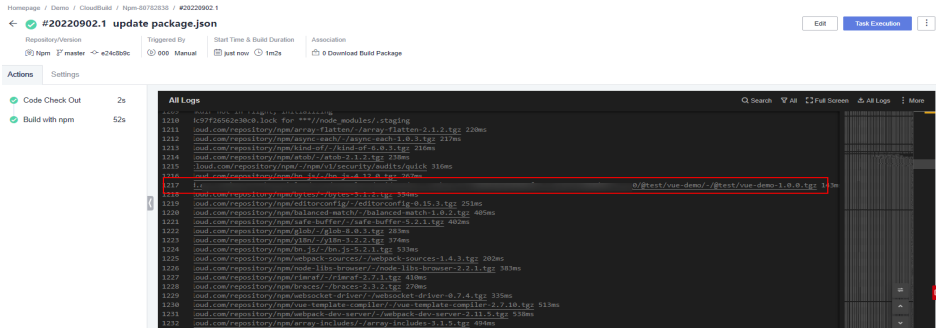
```
export PATH=$PATH:/root/.npm-global/bin
npm config set strict-ssl false
npm install --verbose
```



Step 3 Click **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.

 Copyright © Huawei Technologies Co., Ltd.



----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 self-hosted 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
```

 Copyright © Huawei Technologies Co., Ltd.

5 Releasing/Obtaining a Self-Hosted 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.

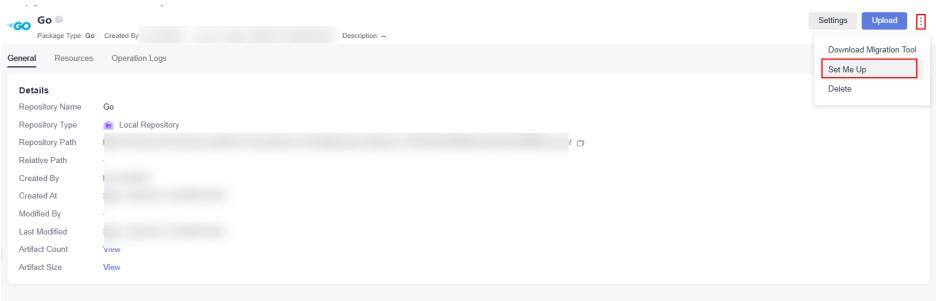
Prerequisites

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

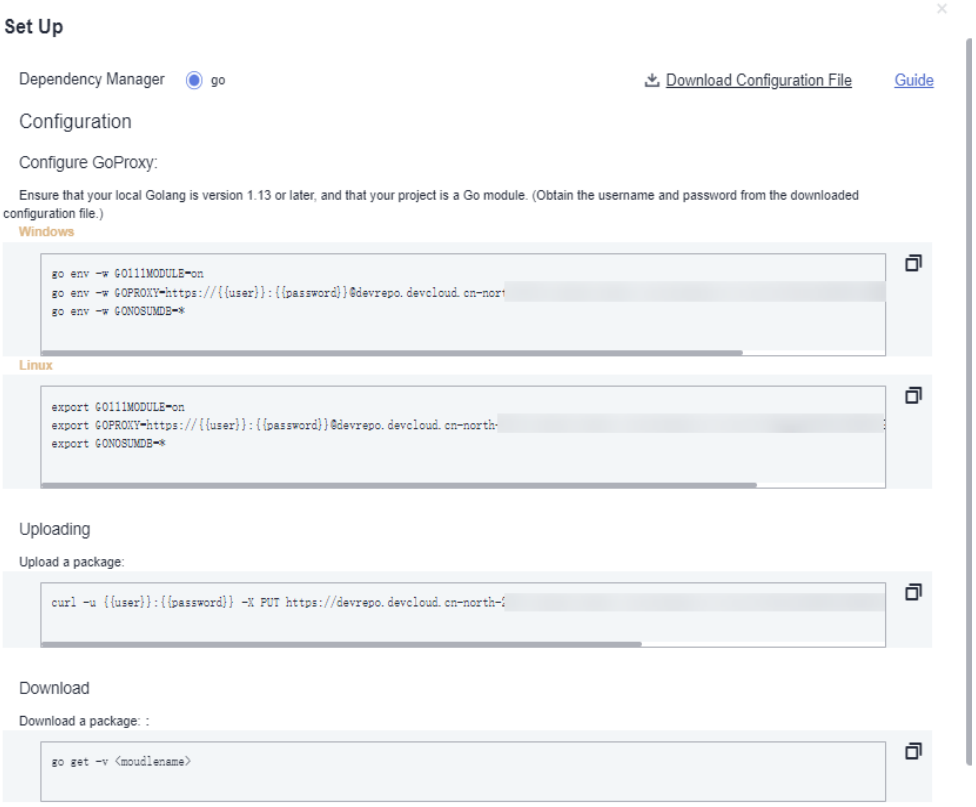
Step 1 Download the configuration file of the self-hosted repo.

1. Log in to the CodeArts homepage and access the self-hosted repo for Go. Click ⋮ on the right of the page and choose **Set Me Up** from the drop-down list.



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

 Copyright © Huawei Technologies Co., Ltd.



Step 2 Configure a code repository.

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

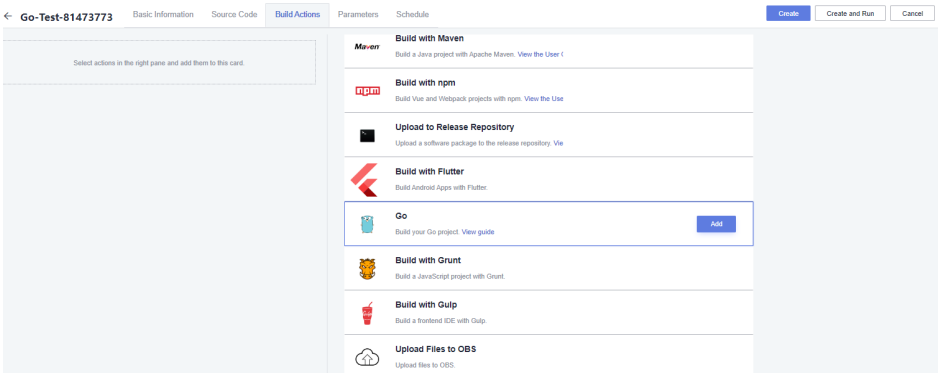
go.mod

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

Step 3 Configure and execute a build task.

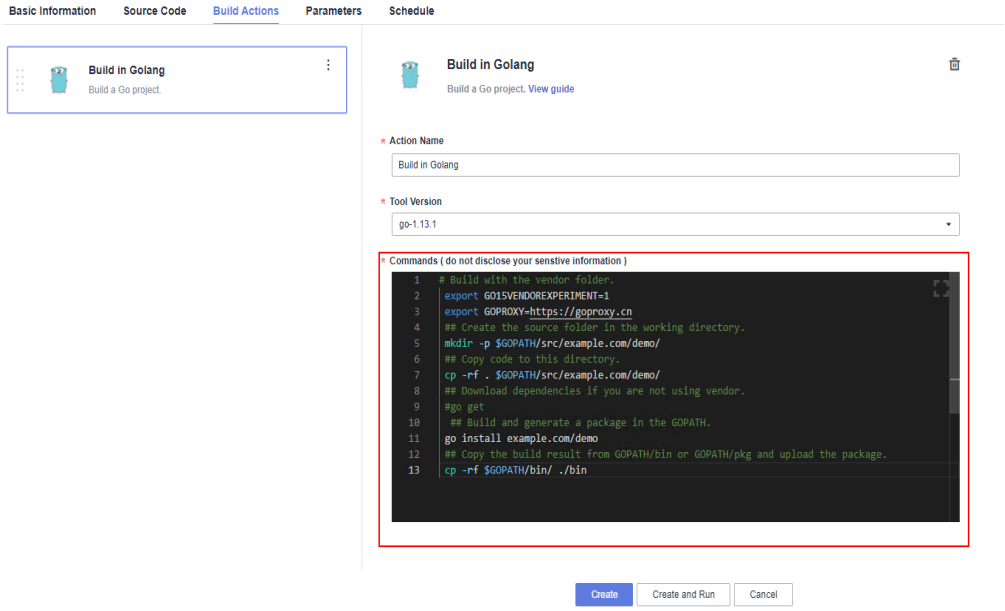
1. On the code repository page, click **Create Build Task** in the upper right. The **Create Task** page is displayed.

   Select **Blank build template** and click **OK**.
2. Add the **Build in Go** action.



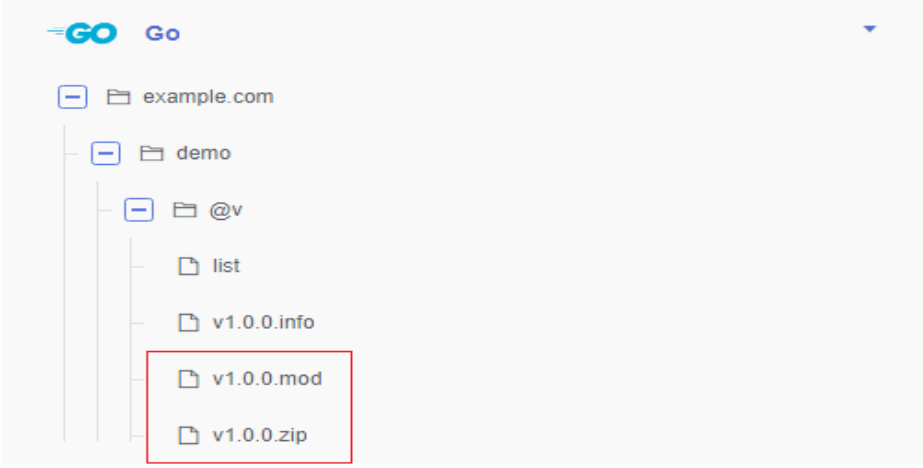
 Copyright © Huawei Technologies Co., Ltd.

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

Copyright © Huawei Technologies Co., Ltd.

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**.

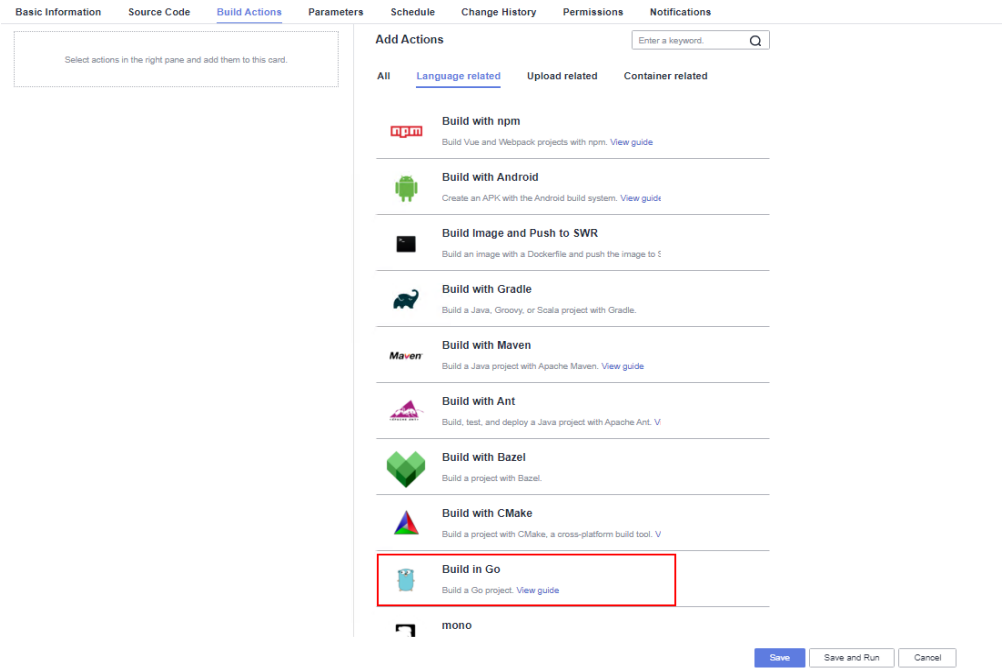
Step 2 Go to CodeArts Repo and create a Go repository. For details, see **Cloud Repository Creation**. This procedure uses the **Go web Demo** template.

Step 3 Configure and execute a build task.

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

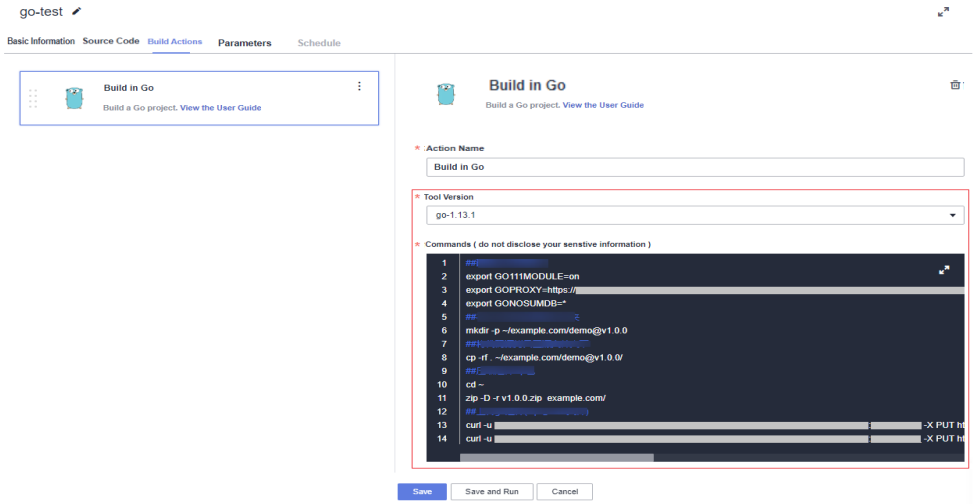
   Select **Blank build template** and click **OK**.

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



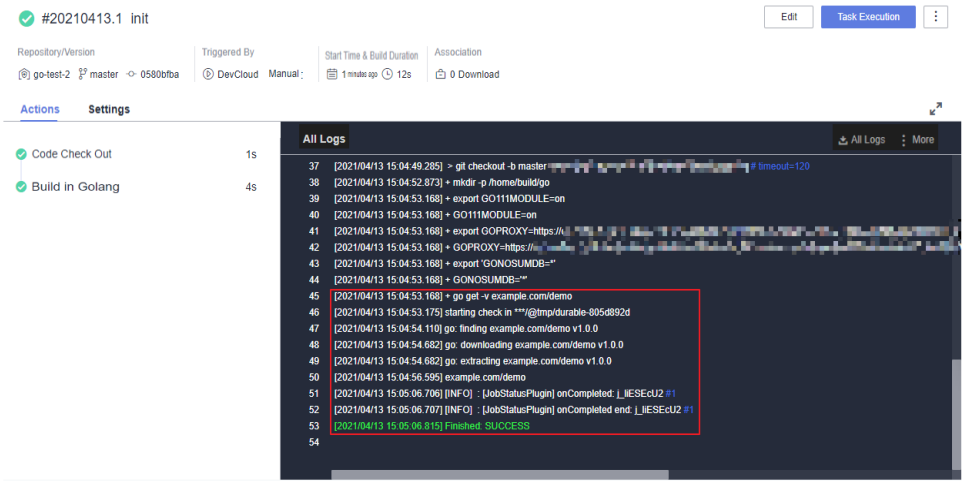
3. Edit the **Build in Go** action.
   - Select the desired tool version. In this example, **go-1.13.1** is used.
   - Delete the existing commands, open the downloaded configuration file, and copy the **commands for configuring Go environment variables in Linux** to the command box.
   - Copy the **Go 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**).

 Copyright © Huawei Technologies Co., Ltd.



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.

 Copyright © Huawei Technologies Co., Ltd.

```
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

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

  a. Create a source folder in the working directory.

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

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

  b. Copy the code source to the source folder.

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

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

  c. Compress the component into a ZIP package.

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

     ```
     cd ~
     ```

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

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

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

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

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

Copyright © Huawei Technologies Co., Ltd.

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

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

  a. Create a source folder in the working directory.

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

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

  b. Copy the code source to the source folder.

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

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

  c. Compress the component into a ZIP package.

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

     ```
     cd ~
     ```

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

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

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

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

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

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

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

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

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

  go.mod

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


Copyright © Huawei Technologies Co., Ltd.

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
```


Copyright © Huawei Technologies Co., Ltd.

6 Releasing/Obtaining a Self-Hosted 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.

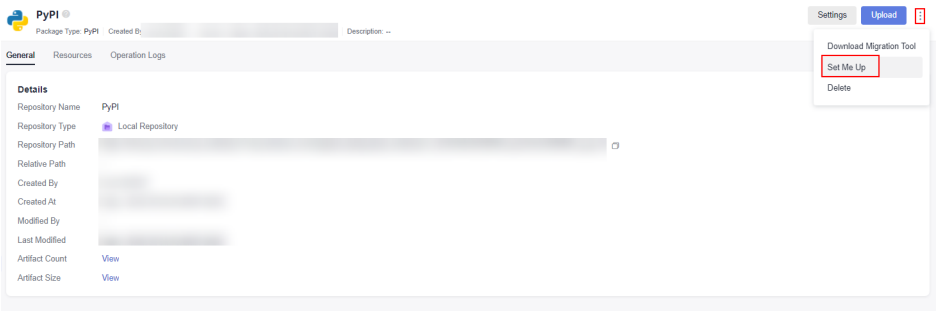
Prerequisites

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 Self-Hosted Component to a Self-Hosted PyPI Repo

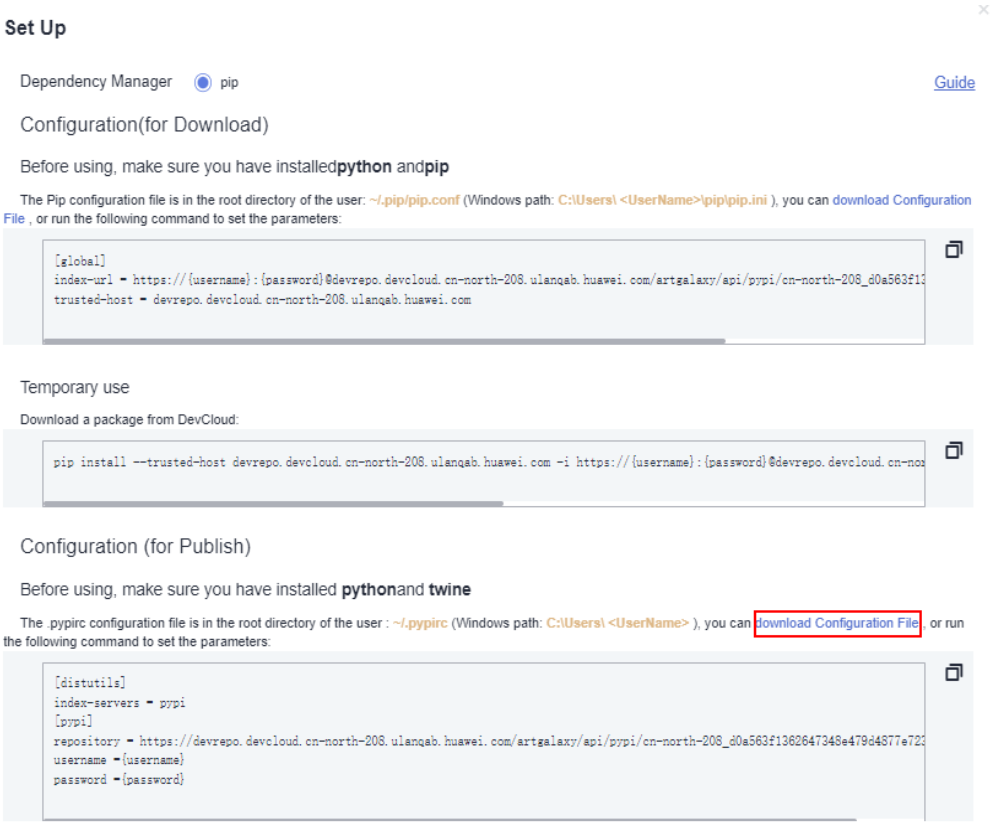
Step 1 Download the configuration file of the self-hosted repo.

1. Log in to the CodeArts homepage and access the self-hosted repo for PyPI. Click ⋮ on the right of the page and choose **Set Me Up** from the drop-down list.



2. Click **download Configure File** under **Configuration (for Publish)**.

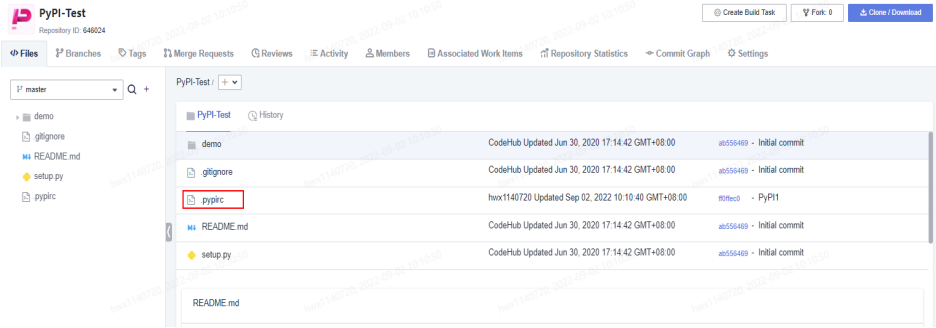
 Copyright © Huawei Technologies Co., Ltd.



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 **Uploading to a Code 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**.

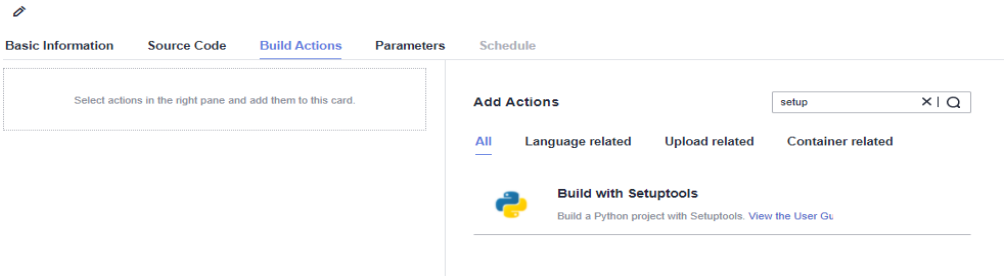


Step 3 Configure and execute a build task.

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

   Select **Blank build template** and click **OK**.
2. Add the **Build with Setuptools** action.

 Copyright © Huawei Technologies Co., Ltd.



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/*
```

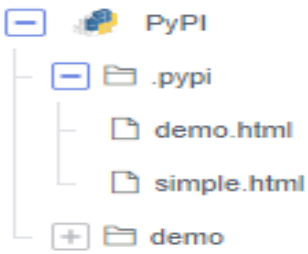
NOTE

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

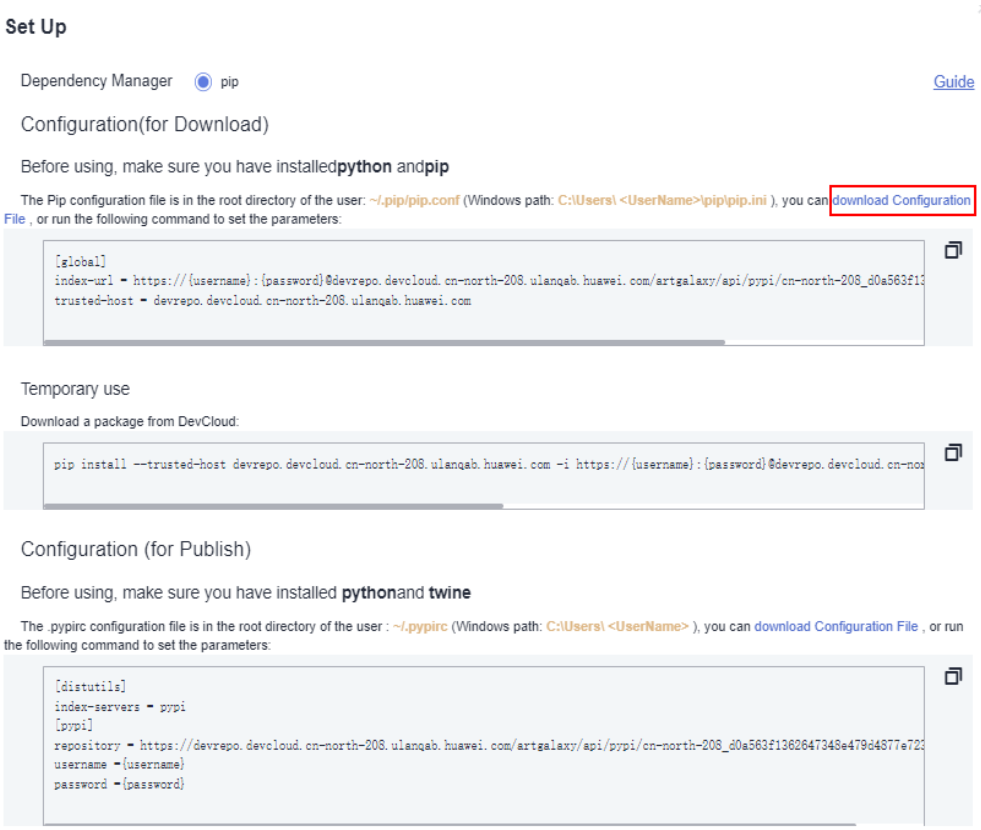
Obtaining a Dependency from a Self-Hosted PyPI Repo

The following procedure uses the PyPI component released in **Releasing a Self-Hosted 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 PyPI repository, and click **Set Up** on the right of the page.
2. Click **download Configure File** under **Configuration(for Download)**.

 Copyright © Huawei Technologies Co., Ltd.

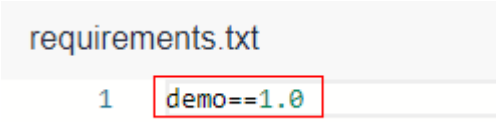


3. Save the downloaded **pip.ini** file as a **pip.conf** file.

Step 2 Configure a code repository.

1. Go to CodeArts Repo and create a Python repository. For details, see **Uploading to a Code 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
```

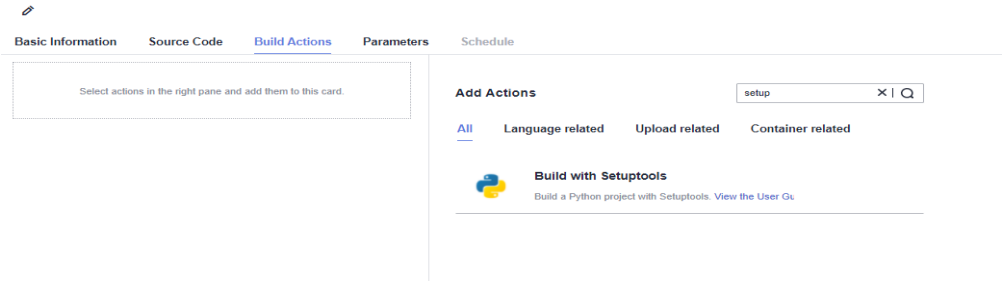


Step 3 Configure and execute a build task.

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

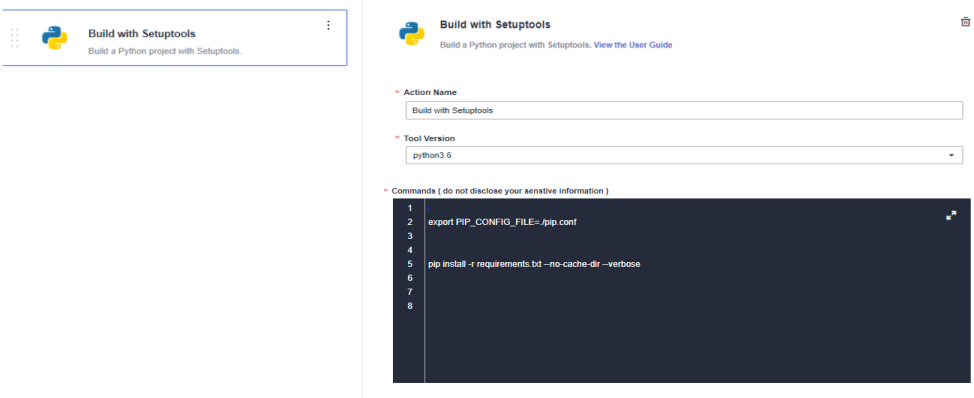
   Select **Blank build template** and click **OK**.
2. Add the **Build with Setuptools** action.

 Copyright © Huawei Technologies Co., Ltd.



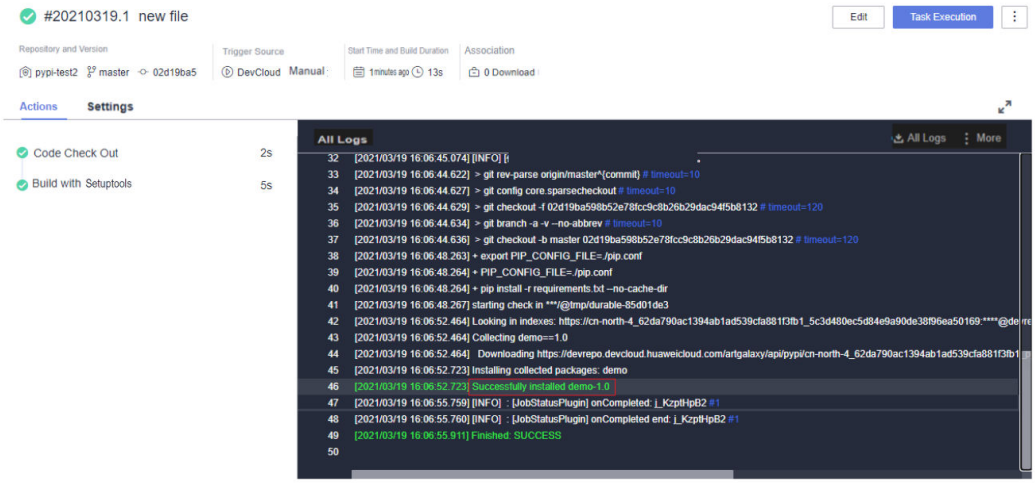
3. Edit the **Build with Setuptools** action.
   - Select the desired tool version. In this example, **python3.6** is used.
   - Delete the existing commands and run the following instead:

```
# 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
```



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

 Copyright © Huawei Technologies Co., Ltd.

7 Uploading/Obtaining an RPM Component Using Linux Commands

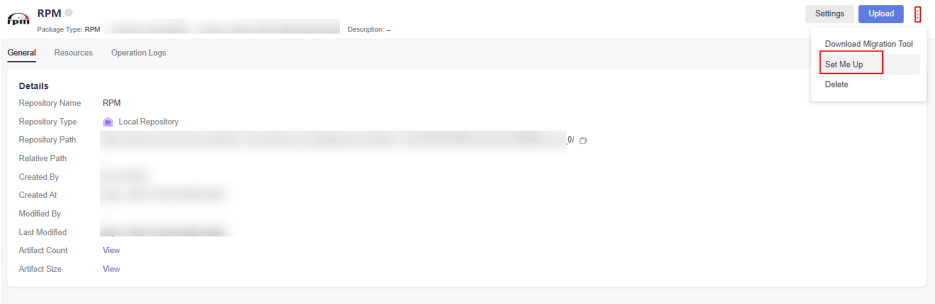
This section describes how to use Linux commands to upload a component to a self-hosted RPM repo and obtain a dependency from the repository.

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**.

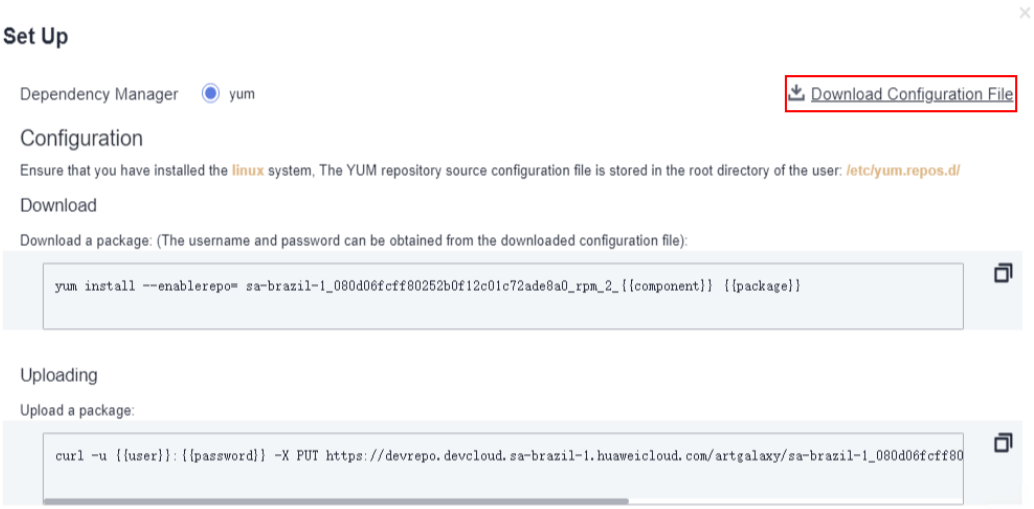
Releasing a Component to a Self-Hosted RPM Repo

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



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

 Copyright © Huawei Technologies Co., Ltd.



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}}**

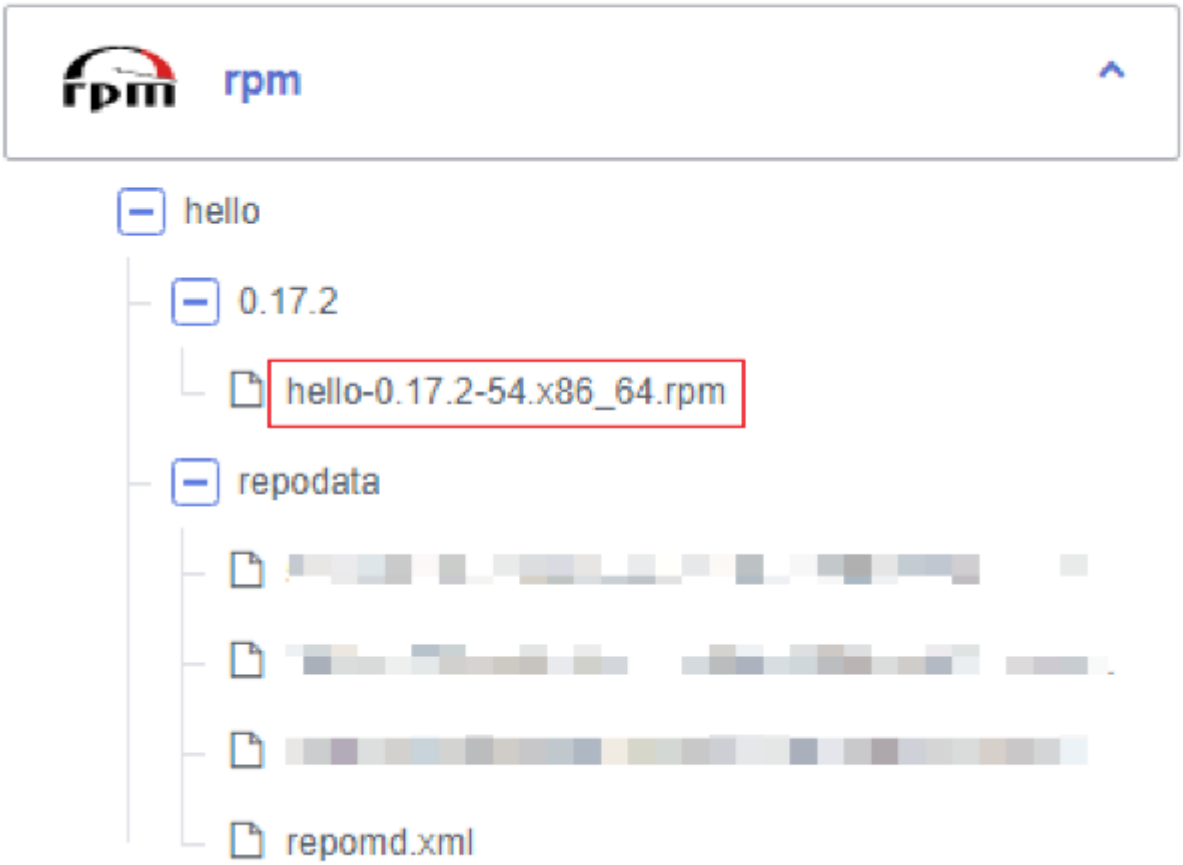
component, **version**, and **localFile** can be obtained from the RPM component. The **hello-0.17.2-54.x86_64.rpm** component is used as an example.

- *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.

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

 Copyright © Huawei Technologies Co., Ltd.



----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.

```
[                    yum.repos.d]# pwd
/etc/yum.repos.d
[                    yum.repos.d]# ll
total 20
-rw-r--r-- 1          737 Mar 12 11:04 cn-north                    _rpm_0.repo
-rw-r--r-- 1          235 Jan 25 23:00
-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
```

 Copyright © Huawei Technologies Co., Ltd.

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

```
yum install hello
```

----End

Copyright © Huawei Technologies Co., Ltd.

8 Uploading/Obtaining a Debian Component Using Linux Commands

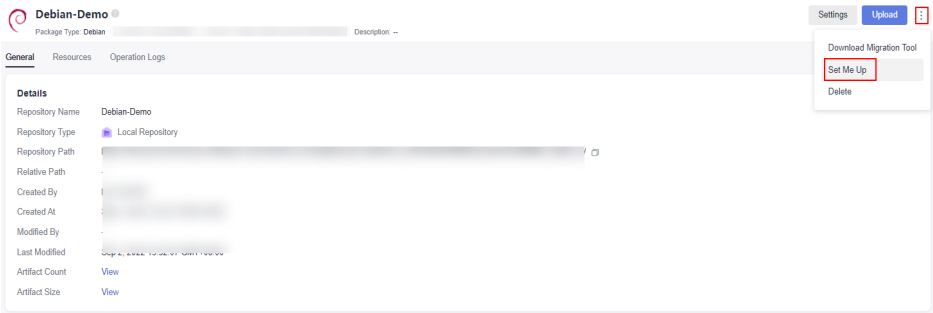
This section describes how to use Linux commands to upload a component to a self-hosted Debian repo and obtain a dependency from the repository.

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**.

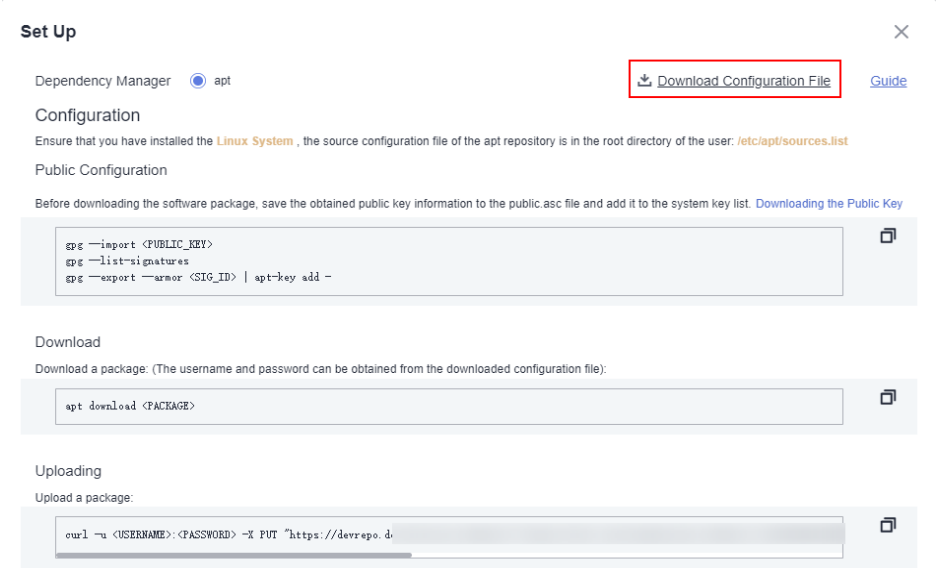
Releasing a Component to a Self-Hosted Debian Repo

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



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

 Copyright © Huawei Technologies Co., Ltd.

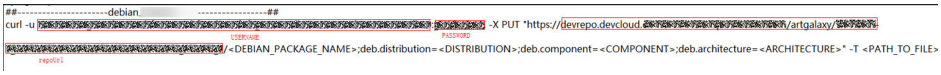


Step 3 On the Linux host, run the following command to upload a Debian component:

```
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.

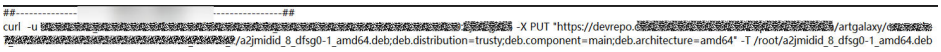


DEBIAN_PACKAGE_NAME, **DISTRIBUTION**, **COMPONENT**, and **ARCHITECTURE** can be obtained from the Debian component.

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

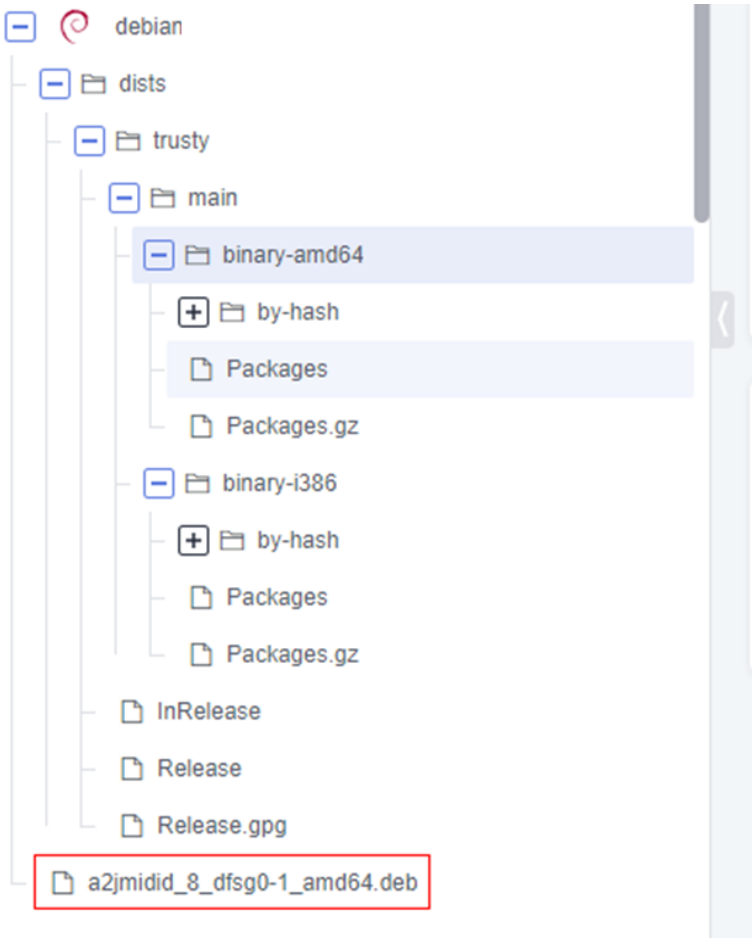
- **DEBIAN_PACKAGE_NAME**: software package name, for example, **a2jmidid_8_dfsg0-1_amd64.deb**.
- **DISTRIBUTION**: release version, for example, **trusty**.
- **COMPONENT**: component name, for example, **main**.
- **ARCHITECTURE**: system architecture, for example, **amd64**.
- **PATH_TO_FILE**: local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

The following figure shows the complete command.



 Copyright © Huawei Technologies Co., Ltd.

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



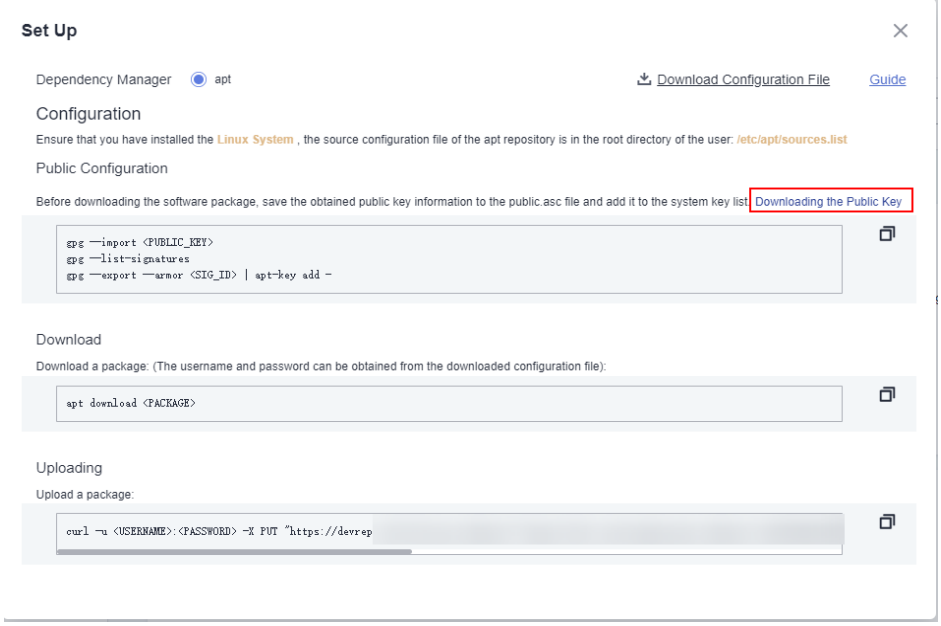
----End

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**.

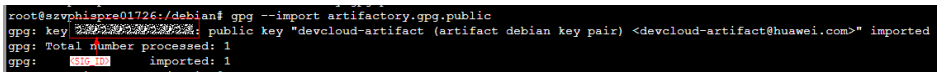
 Copyright © Huawei Technologies Co., Ltd.



Step 2 Import the gpg public key.

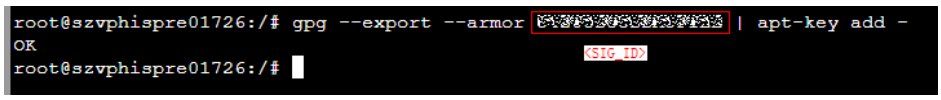
```
gpg --import <PUBLIC_KEY_PATH>
```

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 -
```



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
```

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

```
apt download a2jmidid
```

 Copyright © Huawei Technologies Co., Ltd.

NOTE

Method for obtaining packages:

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

----End

Copyright © Huawei Technologies Co., Ltd.