CodeArts Build

User Guide

Issue	01
Date	2023-11-30

HUAWEI TECHNOLOGIES CO., LTD.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:
https://www.huawei.com/en/psirt/vul-response-process
For vulnerability information, enterprise customers can visit the following web page:
https://securitybulletin.huawei.com/enterprise/en/security-advisory

Contents

1 Before You Start

Build is the process of converting source code into a target file and packaging configuration and resource files.

CodeArts Build provides an easy-to-use, cloud-based build platform that supports multiple programming languages, helping you achieve continuous delivery, with shorter delivery period and higher delivery efficiency. With CodeArts Build, you can create, configure, and run build tasks with a few clicks. CodeArts Build also supports automation of code retrieval, build, and packaging, as well as real-time status monitoring.

For more product information, see **Service Overview**.

Before using CodeArts Build, learn about the **roles, permissions**, and **process** of the service.

2 Roles & Permissions

The following table describes the default user role types and build task operation permissions in CodeArts Build.

Table 2-1 Default role permission matrix

Role	Edit	Delete	View	Run	Clone	Disable	Assign Permissions
Task creator	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)
Project creator	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)	√ (*)
Project manager	√	√	√	√	√	√	√
Developer	√	√	√	√	√	√	×
Test manager	×	×	√	×	×	×	×
Tester	×	×	×	×	×	×	×
Participant	×	×	×	×	×	×	×
Viewer	×	×	√	×	×	×	×

NOTE

- A check mark (√) indicates that the user has the permission by default, and a cross mark (×) indicates that the user does not have the permission by default.
- Roles who have the permission to assign permissions can modify the permission matrix, but the permissions marked with an asterisk (*) cannot be modified.
- Project creators, project managers, and developers can create build tasks.

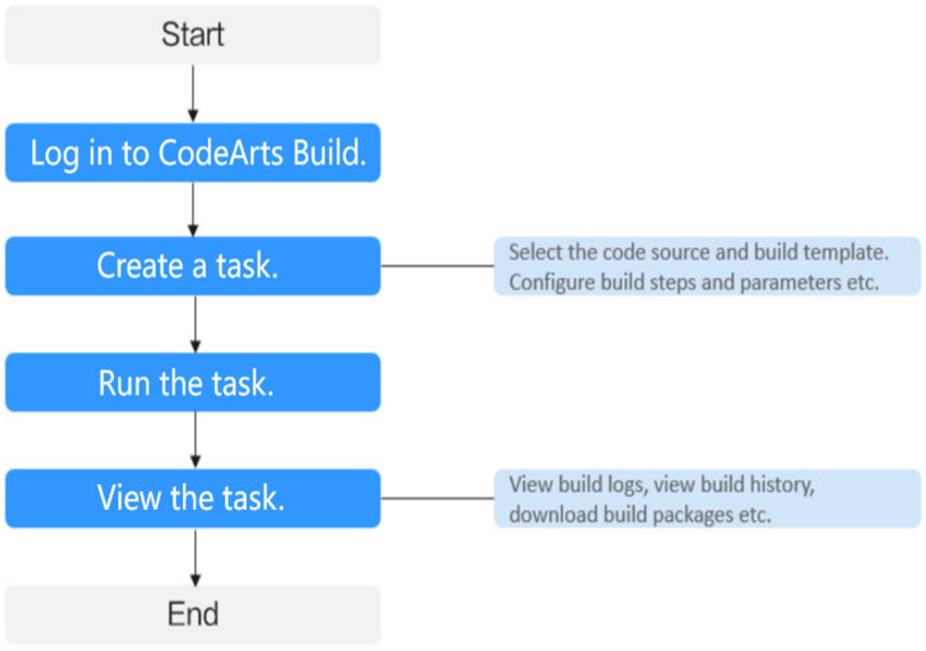
If the current role permissions do not meet your needs, configure the permissions by referring to **Configuring Role Permissions**.

3 Process

CodeArts Build provides an easy-to-use cloud-based build platform that supports multiple programming languages, helping you achieve continuous delivery with shorter delivery period and higher delivery efficiency. With CodeArts Build, you can create, configure, and run build tasks with a few clicks. CodeArts Build also supports automation of code retrieval, build, and packaging, as well as real-time status monitoring.

Introduction

This topic describes the basic build process.


Start
Log in to CodeArts Build.
Create a task.
Select the code source and build template. Configure build steps and parameters etc.
Run the task.
View the task.
View build logs, view build history, download build packages etc.
End

The process is described in the following table.

Operation	Description
Log in to the CodeArts Build homepage	Access the homepage of CodeArts Build.
Create a task	Create a build task and configure the following information: ● **Code source**: Select Repo, GitHub, Git, or Pipeline. ● **Build template**: CodeArts Build comes with default templates for mainstream build standards such as Maven, Ant, Gradle, and CMake. You can also customize your build environment by creating images or using public images to meet special build requirements. ● **Build actions**: CodeArts Build has various preset actions. You can customize the combination of actions.
Run a task	After the task is configured, run the task. For details, see **Running a Build Task**.
View the build task	After the task execution is complete, you can view the details and execution results of the task. For details, see **Viewing the Build Task**.

4 Logging In to the CodeArts Build Homepage

Prerequisites

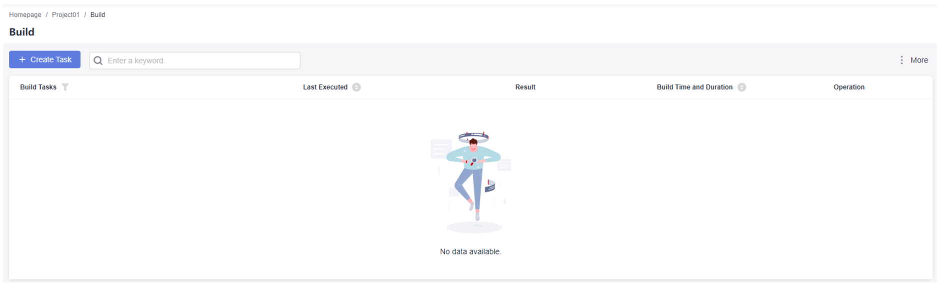
- You have **registered a HUAWEI ID and enabled Huawei Cloud services**.

Procedure

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

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

Step 3 Click **Access Service** to go to the service homepage.



- Click in the upper left corner of the page and select a region.
- Click **More** to access the following functions:
  - **Custom Templates**
  - **Custom Build Environments**
  - **Files**
  - **Recycle Bin**
  - **Pools**

----End

5 Creating a Build Task

Prerequisites

- A project is available. If no project is available, **create one**.
- A code repository has been created in the project. If no code repository is available, **create one**.

Configuring Basic Information

1. **Log in to the CodeArts Build homepage**.
2. Click **Create Task**. On the displayed page, configure the basic information of the build task.

Table 5-1 Basic information

Parameter	Description
Task Name	Enter the name of the task.
Project	Select the project that the task belongs to.
Code Source	● Repo: By default, CodeArts Build pulls code from CodeArts Repo. Select a source code repository and branch. ● Git: For code hosted on other services, you can use a Git connection to pull the code. For details, see **Using Git for Build**. ● Pipeline: If the code source is from a pipeline, the code can be executed only by the pipeline driver and cannot be executed alone.
Description	Describe the task.

Configuring a Build Template

1. Click **Next**. The **Build Template** page is displayed.
2. Select a suitable build template and click **Next**. You can also select the **Blank Template**.

If the existing templates do not meet your needs, **customize templates**.

Configuring Build Actions

1. Click **Next**. The **Build Action** tab page is displayed, showing the default action combination of the selected template.
2. Click + to add build actions as required.

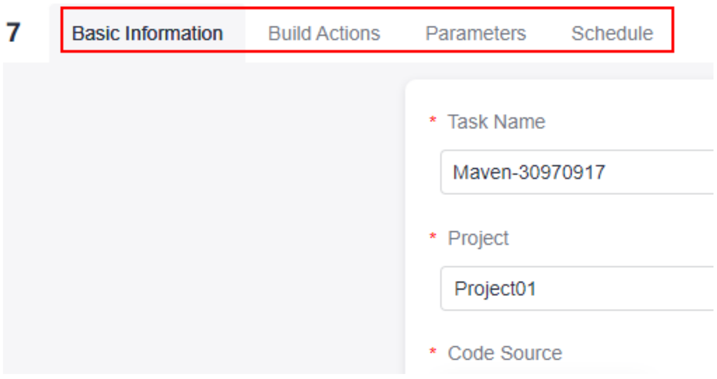
   For details, see **Configuring Build Actions**.

   NOTE

   You can also **use SoftWare Repository for Container (SWR) public images** to build a custom environment.

Configuring Other Information

Configure information on other tab pages of the navigation bar.



* Task Name
Maven-30970917
* Project
Project01
* Code Source

- On the **Basic Information** tab page, configure the task name, project, code source, and task description.
- On the **Build Actions** tab page, configure build actions.
- On the **Parameters** tab page, customize parameters for running the build task.
- On the **Schedule** tab page, configure the scheduled execution and continuous integration triggering policy.

6 Configuring Build Actions

6.1 Introduction

CodeArts Build provides **graphical build** and **code-based build**.

Graphical Build

CodeArts Build provides various build actions and allows you to orchestrate them as required. If the preset build tool version cannot meet your requirements, customize a build environment and package it into a Docker image. Push the image to SoftWare Repository for Container (SWR) for future use. For details, see **Creating Images and Pushing to SWR** and **Using SWR Public Images**.

Code-based Build

Code-based build only uses **Repo as the code source**.

You can use YAML files to configure build scripts. To be specific, you can use YAML syntax to write a **build.yml** file based on the build environment, parameters, commands, and actions required during the build process. You can also add the **build.yml** file to a code repository together with the built code. The system uses the **build.yml** file as the build script to execute the build task, making the build process traceable, recoverable, secure, and reliable. Code-based build has the following advantages:

- The script file clearly describes the build process, including build parameters, commands, steps, and post-build operations, to make the build process trustworthy.
- The **build.yml** configuration corresponding to the current commit is used for each build to ensure that the build can be restored and traced. You do not need to worry that the previous task cannot be executed repeatedly due to build configuration modification.
- If the build script needs to be modified for a new feature, you can create a branch to modify the **build.yml** file for testing without worrying about affecting other branches.

This build method supports the configuration of **a single task** or **multiple tasks**.

6.2 Graphical Build

6.2.1 Configuring Build Environment

Configure a global runtime environment for the build task.

NOTE

There are x86 servers and Arm servers. For software running on different chip architectures, select the corresponding hosts. Your software will run better on a server using the same architecture. Kunpeng servers are Arm-based.

macOS executors

- Currently, you can run build jobs on macOS executors. All macOS versions are supported.
- If you select a macOS executor, only the following build actions are available: **Run Shell Commands**, **Uploading Software Packages to Release Repos**, and **Downloading Package from Release Repos**.

Configuration Description

Configure the build environment.

The parameters are described in the following table.

Parameter	Description
Host type	x86/Kunpeng (Arm) server

Parameter	Description
Executor	Compute resource used to execute build tasks. In CodeArts Build, VMs are used. Executors can be built-in or custom executors. ● Built-in executors: Provided by CodeArts Build with out-of-the-box availability. ● Custom executors: Compute resources provided by users. They can be hosted in CodeArts Build after registration. CodeArts Build schedules these executors to execute build tasks. You can select a built-in or custom executor. A custom executor is the agent executor added to the agent pool. For details about how to customize an executor, see **Agent Pools**. **NOTE** Built-in executors are only available in x86 servers.

6.2.2 Configuring Code Download

Configure the code download mode. You can use the specified code repository tag or commit ID to build the code. In addition, you can enable the automatic update of submodules and Git LFS.

Configuration Description

Configure the code download.

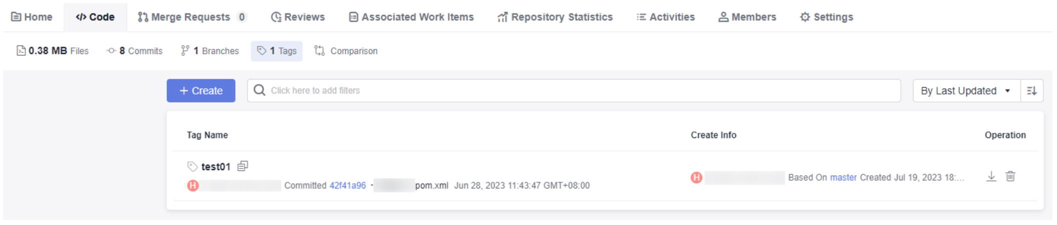
The parameters are described in the following table.

Parameter	Description
Use a specified code repository tag or commit ID	Do not specify a tag or a commit ID, **specify a tag**, **or specify a commit ID**.

Parameter	Description
Auto Update	Submodule is a concept of Git and is used to solve the problem that a code repository contains and uses the code repository of other projects. For details, see **Submodules (Git Submodule)**. ● Enabled: If the code repository contains submodules, the system automatically pulls the code from the submodule repository during a build. ● Disabled: The system does not automatically pull the code of the submodule repository.
Enable Git LFS	Determine whether to enable Git LFS as required. By default, large files such as audio, video, and images are not pulled. After Git LFS is enabled, all files are pulled.

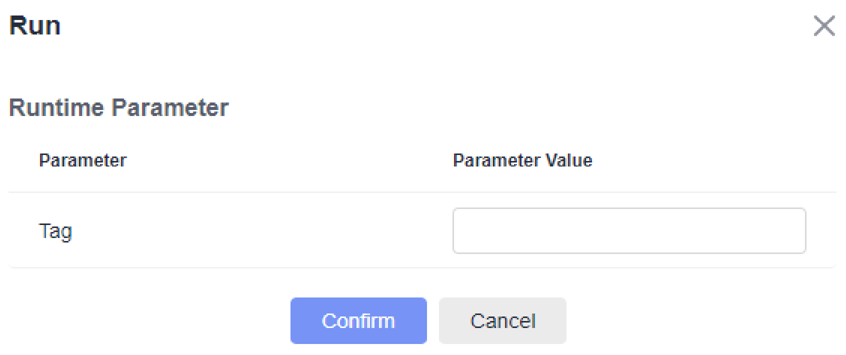
Build by Tag

A tag is associated with a code repository. If you select Repo as the code source, you can create a tag by referring to **Managing Tags**.



Tag Name	Create Info	Operation
test01 [redacted] Committed 42f41a96 · [redacted] pom.xml Jun 28, 2023 11:43:47 GMT+08:00	[redacted] Based On master Created Jul 19, 2023 18:...	

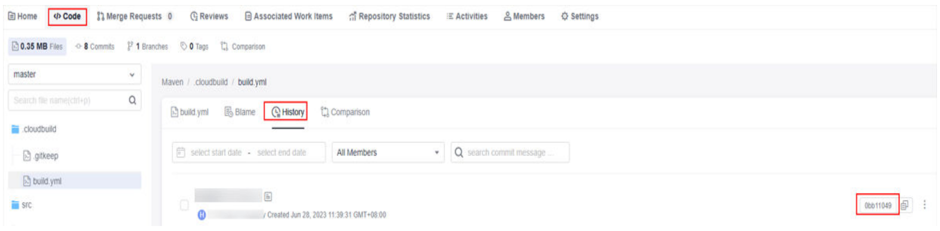
1. When creating a build task, select **Tag** to use the code of a previous version.
2. During task execution, a dialog box is displayed. Enter the tag name and click **Confirm** to run the task.



Parameter	Parameter Value
Tag	

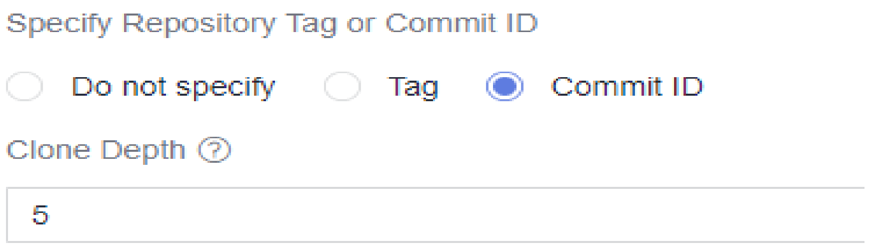
Build by Commit ID

A commit ID is the number generated when the code is committed. If you select Repo as the code source, the commit ID is displayed in the code repository as shown in the following figure.



In a build task, you can specify the commit ID to use the code of a previous version for building.

1. Select **Commit ID**, enter the clone depth, and save the task.



Specify Repository Tag or Commit ID

Do not specify   Tag   [✓ Commit ID]

Clone Depth

5

NOTE

The clone depth is the number of revisions of the repository that will be cloned. A larger value indicates a longer time for checking out the code. The clone depth must be a positive integer. The recommended maximum depth is 25.

For example, if **Clone Depth** is set to 5, you can set **Commit ID** to any of the previous five commits.

2. Enter the commit ID and click **Confirm** to start the task.

6.2.3 Build with Maven

6.2.3.1 Operation Guide

Built-in tools such as Maven and JDK are provided. Select a tool version based on the build scenarios.

Maven is used to build a Java project, which has the following functions:

- You can run **mvn package**, **mvn deploy**, or other shell commands for your build.
- You can use public repositories not provided by CodeArts for your build.
- You can add other private repositories.
- Deployment configurations can be automatically added to the **pom.xml** file. You can run **mvn deploy** to release private dependencies to self-hosted repos.
- You can view reports of JUnit unit testing after build.

Configuration Description

Add **Build with Maven** when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version. **NOTE** If the preset tool version cannot meet your requirements, you can **customize a Docker image**, add dependencies and tools required by the project, package the required environment into a Docker image, and push the image to SoftWare Repository for Container (SWR). For details, see **Creating Images and Pushing to SWR** and **Using SWR Public Images**.
Commands	Configure Maven commands. You can also use default commands.
setting File Configuration	The **setting** file is automatically generated with repositories. The optimal repository access mode is automatically configured based on the user's IP address, which may be in regions in or outside China. You are advised to retain the default settings. You can also add a repository that cannot be found in CodeArts Mirrors, Self-hosted Repos, or Huawei SDK repositories. For details, see **Configuring a Repository**.
Release to Self-hosted Repos	By default, CodeArts Build uses the self-hosted repos as the download source of private dependency. The configuration is required for uploading build products to the self-hosted repos and store the build products as dependencies for other projects. For details, see **Configuring the Release to Self-hosted Repos**.
Unit test	To process unit test results, set the parameters. For details, see **Configuring a Unit Test**.
Cache	Opt to use the cache to improve the build speed. If you set **Use Dependency Cache** to **Yes**, the downloaded dependency package is cached during each build. In this way, the dependency package does not need to be pulled repeatedly during subsequent builds, which effectively improves the build speed. **NOTE** After the dependency package built by Maven is stored in the cache, the cache directory is updated only when a new dependency package is introduced to the project built by the tenant. The existing dependency package cache file cannot be updated.

6.2.3.2 Configuring a Repository

Configuration Description

This section describes how to configure repositories not provided by CodeArts for builds. In the **Build with Maven** action, there are public and private repositories based on their sources, networks, and permissions.

- Public Repositories
  - CodeArts Mirror: By default, CodeArts Mirror is configured. This repository can be used in build tasks without any modification.
  - Custom public repositories: A public repository not provided by CodeArts can be used only after being **configured** in the **Build with Maven** action. (A public repository is accessible in the Internet.)
- Private Repositories
  - Self-hosted Repos: By default, self-hosted repos of CodeArts Artifact (if subscribed) are configured. These repositories can be used in build tasks without any modification.
  - Custom private repository: A private repository not provided by CodeArts can be used only after being **configured** in the **Build with Maven** action. (A private repository is accessible only to authorized accounts.)

Configuring a Custom Public Repository

1. In the **Build with Maven** action, expand **setting File Configuration**.
2. Add a public repository, enter the repository address, and select **Release** and **Snapshot** as required.
   - **Release**: If this option is selected, the build process attempts to download the release version dependency from the repository.
   - **Snapshot**: If this option is selected, the build process attempts to download the snapshot version dependency from the repository.

   NOTE

   Select either **Release** or **Snapshot**, or both.

Configuring a Custom Private Repository

1. **Create a Nexus repository service endpoint**, such as **test01**.
2. In the **Build with Maven** action, expand **setting File Configuration**.

   Add a private repository, select the service endpoint created in step 1, and select **Release** and **Snapshot** as required.

6.2.3.3 Configuring the Release to Self-hosted Repos

Configuration Description

By default, CodeArts Build uses the self-hosted repos as the download source of private dependency. The configuration is required for uploading build products to the self-hosted repos and store the build products as dependencies for other projects.

- **Release repo** is used to archive software packages for deployment or other purposes.
- **Self-hosted repo** is used to store tool packages that other projects depend on.

  Self-hosted Maven repositories are classified into release and snapshot repositories.

  – For private dependency packages released for debugging, add the **-SNAPSHOT** suffix to the dependency version (for example, **1.0.0-SNAPSHOT**). During each release, the dependency is automatically released to the snapshot repository. The version does not need to be updated each time the dependency is released. You can add the **-U** parameter to the build command to obtain the latest version.

  – For officially released private dependency packages, do not add the **-SNAPSHOT** suffix to the dependency version (for example, **1.0.0**). During each release, the dependency is automatically released to the release repository. The version must be updated each time the dependency is released. Otherwise, the latest dependency package cannot be obtained during the build.

NOTE

Pay attention to their differences. If you upload a dependency to a release repo, it cannot be downloaded during building.

Procedure

Step 1 Create a self-hosted repo. (Skip this step if the repository is available.)

Step 2 Use the Maven template to create a code repository.

Step 3 Click the name of the code repository. On the **Files** page that is displayed, configure the self-hosted repo coordinate information (groupId, artifactId, and version) in the **pom.xml** file.

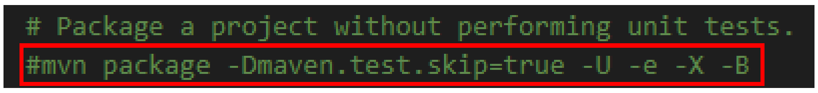
Modify the self-hosted project to be built. The coordinates specified in the **pom.xml** file are as follows:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>come.test.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 4 In the build with Maven action, configure **Build with Maven**, expand **Release to Self-hosted Repos**, and select **Configure all POMs**.

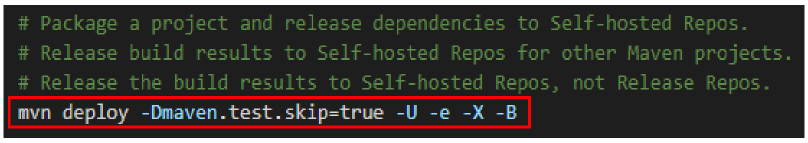
- **Do not configure POM**: Self-hosted repos are not required.
- **Configure all POMs**: Deployment configurations are added to all **pom.xml** files of the project. The **mvn deploy** command is used to upload the built dependency package to the self-hosted repo.

Step 5 In the command window, use **#** to comment out the **mvn package -Dmaven.test.skip=true -U -e -X -B** command.



```
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B
```

Step 6 Delete **#** before the **#mvn deploy -Dmaven.test.skip=true -U -e -X -B** command.



```
# 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
```

Step 7 Run the build task. After the execution is successful, the dependency package is released to the self-hosted repo.

Step 8 In the navigation pane, choose **Artifact** > **Self-hosted Repos**. On the displayed page, search for and view the uploaded dependency.

After the upload is successful, add the following coordinates to other projects for reference.

```
<dependency>
    <groupId>com.test.demo</groupId>
    <artifactId>javaMavenDemo</artifactId>
    <version>1.0</version>
</dependency>
```

----End

6.2.3.4 Configuring a Unit Test

Configuration Description

To process the unit test results, configure the unit test function provided by the **build with Maven build**. Compile the unit test code in the project and ensure that the following conditions are met:

- The storage location of unit test code must comply with the default unit test case directory specifications and naming specifications of Maven. You can specify the case location in the configuration.

  For example, if the unit test cases are stored in **src/test/java/{{package}}/**, the unit test is automatically executed during a Maven build task.
- The project cannot contain the configuration code of ignoring unit test cases.

  Click the name of the code repository. The **Files** page of CodeArts Repo is displayed. Verify that the following code does not exist in the **pom.xml** file of the project:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.18.1</version>
    <configuration>
        <skipTests>true</skipTests>
    </configuration>
</plugin>
```

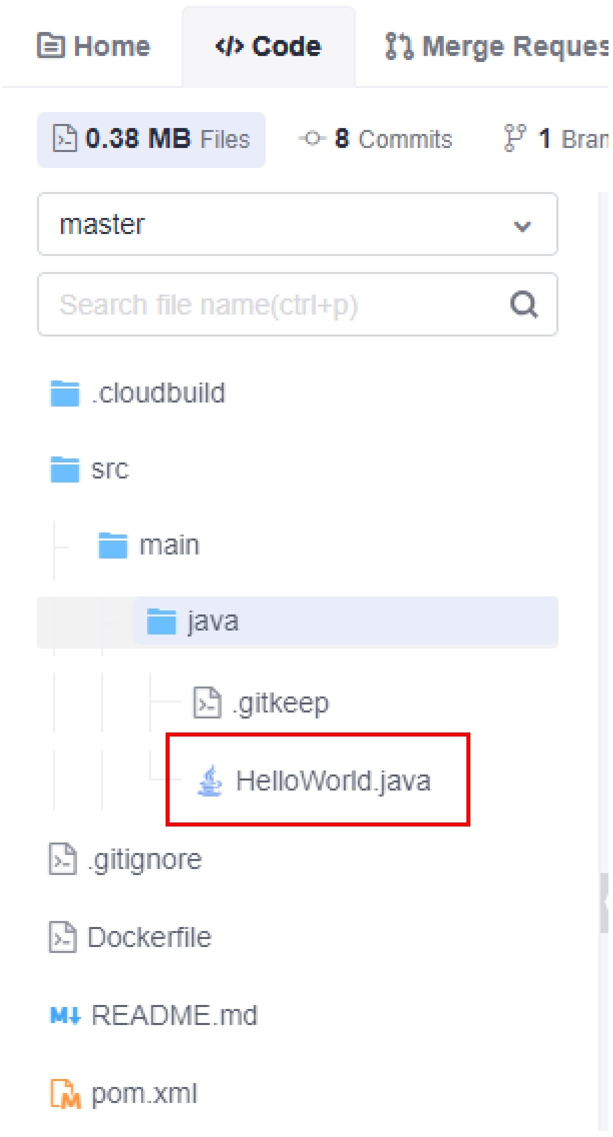
- Click the code repository name. On the **File** page of CodeArts Repo, import the JUnit dependency to the **pom.xml** file. The following shows the code example.

```
<dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.7</version>
 </dependency>
```

Procedure

Step 1 Create a code repository and upload the code to the code repository.

Step 2 Create a unit test class in the **src** directory, as shown in the following figure.



The code of the demo project is as follows:

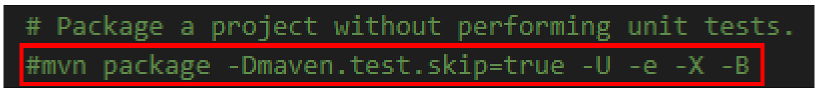
```
package test;
```

```
public class Demo {
    public String test(Integer i) {
        switch (i) {
            case 1:
                return "1";
            case 2:
                return "2";
            default:
                return "0";
        }
}
}
```

The unit test code is shown in the following section. **@Test** indicates the test method annotation.

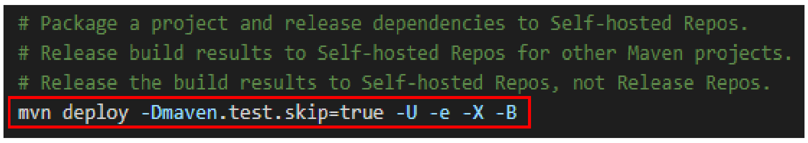
```
package test;

import org.junit.Test;

public class DemoTest {
    private Demo demo=new Demo();
    @Test
    public void  test(){
        assert demo.test(1).equals("1");
        assert demo.test(2).equals("2");
        assert demo.test(3).equals("0");
    }
}
```

Step 3 In the command window displayed in the **build with Maven**, use **#** to comment out the **mvn package -Dmaven.test.skip=true -U -e -X -B** command.



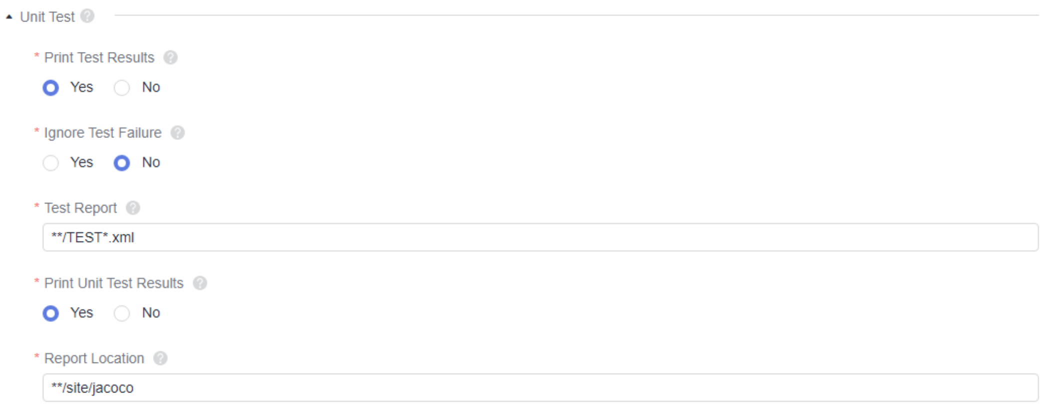
```
# Package a project without performing unit tests.
#mvn package -Dmaven.test.skip=true -U -e -X -B
```

Step 4 Delete **#** before the **#mvn deploy -Dmaven.test.skip=true -U -e -X -B** command.



```
# 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
```

Step 5 Expand **Unit Test**.



Unit Test

* Print Test Results
[✓ Yes] No

* Ignore Test Failure
Yes [✓ No]

* Test Report
**/TEST*.xml

* Print Unit Test Results
[✓ Yes] No

* Report Location
**/site/jacoco

- Select **Yes** for **Print Test Results**.
- Configure **Ignore Test Failure** as required.

  - If **Yes** is selected, the build task is successful when the test case fails.
  - If **No** is selected, the build task fails when the test case fails.
- Configure the path of the unit test result file.

  The test report needs to collect the unit test result to generate a visual report. Therefore, specify the path of the unit test result file.
  - In most cases, retain the default path ****/TEST*.xml**.
  - To improve the accuracy of the result, you can specify a precise report path, for example, **target/surefire-reports/TEST*.xml**.
- Configure **Print Unit Test Results** as required. For details about the configuration method, see **Generating a Unit Test Report Using JaCoCo**.
- Configure the report location.

  A relative path in the project whose files will all be uploaded. Example: target/site/jacoco

Step 6 Run the build task.

After the task is successfully executed, you can view the test report on the **Test** tab page of the task execution details page. If you set **Print Unit Test Results** to **Yes**, a test report is generated. You can click **Download Test Coverage Report** to download the report.

----End

Generating a Unit Test Report Using JaCoCo

- Configuration method for a single-module project

  The jacoco-maven-plugin add-on has been added to the project to generate the unit coverage report. That is, the following configuration has been added to the pom.xml file:

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.5</version>
    <executions>
        <execution>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
 </plugin>
```

  By default, the JaCoCo report target is in the verify phase. You need to define the report target as the test phase. When **mvn test** is executed, the unit test report is generated in the **target/site/jacoco** directory of the code.
- Configuration method for a multi-module project

  Assume that the code structure of a multi-module project is as follows to describe how to configure and generate a unit test report.

```
├── module1
│   └── pom.xml
├── module2
│   └── pom.xml
├── module3
│   └── pom.xml
├── pom.xml
```

a. Add a module for aggregation under the project. The name is **report**. The code structure after the aggregation module is added is as follows:

```
├── module1
│   └── pom.xml
├── module2
│   └── pom.xml
├── module3
│   └── pom.xml
├── report
│   └── pom.xml
├── pom.xml
```

b. Add the jacoco-maven-plugin add-on to the pom.xml file in the root directory of the project.

```
<!--Configure unit test coverage-->
 <plugin>
     <groupId>org.jacoco</groupId>
     <artifactId>jacoco-maven-plugin</artifactId>
     <version>0.8.3</version>
     <executions>
         <execution>
             <goals>
                 <goal>prepare-agent</goal>
             </goals>
         </execution>
     </executions>
 </plugin>
```

c. Configure the pom.xml file of the aggregation module.

Introduce all dependent modules in dependency mode and use **report-aggregate** to define the JaCoCo aggregation target.

```
<dependencies>
     <dependency>
         <groupId>${project.groupId}</groupId>
         <artifactId>module1</artifactId>
         <version>${project.version}</version>
     </dependency>
     <dependency>
         <groupId>${project.groupId}</groupId>
         <artifactId>module2</artifactId>
         <version>${project.version}</version>
     </dependency>
     <dependency>
         <groupId>${project.groupId}</groupId>
         <artifactId>module3</artifactId>
         <version>${project.version}</version>
     </dependency>
 </dependencies>

 <build>
     <plugins>
         <plugin>
             <groupId>org.jacoco</groupId>
             <artifactId>jacoco-maven-plugin</artifactId>
             <version>0.8.3</version>
             <executions>
                 <execution>
                     <id>report-aggregate</id>
                     <phase>test</phase>
                     <goals>
                         <goal>report-aggregate</goal>
```

```
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

After the configuration, run **mvn test** in the root directory of the project. After the command is successfully executed, the coverage report of each module is generated in the **report/target/site/jacoco-aggregate** directory. You can also customize the output path of the report in outputDirectory.

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.3</version>
    <executions>
        <execution>
            <id>report-aggregate</id>
            <phase>test</phase>
            <goals>
                <goal>report-aggregate</goal>
            </goals>
            <configuration>
                <outputDirectory>target/site/jacoco</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

6.2.4 Build with Android

The Android build system compiles application resources and source code, and then packages them into APKs that can be deployed, signed, and distributed.

Custom Installation

sdkmanager command (sdkmanager packages [options]): installs the required Android build environment. For example, **sdkmanager "platform-tools" "platforms;android-28" --sdk_root=./** indicates that sdkmanager is used to download platform-tools and platforms;android-28 to the root directory of the current code.

Configuration Description

Add **Build with Android**, when **configuring build actions**.

The parameters are described in the following table.

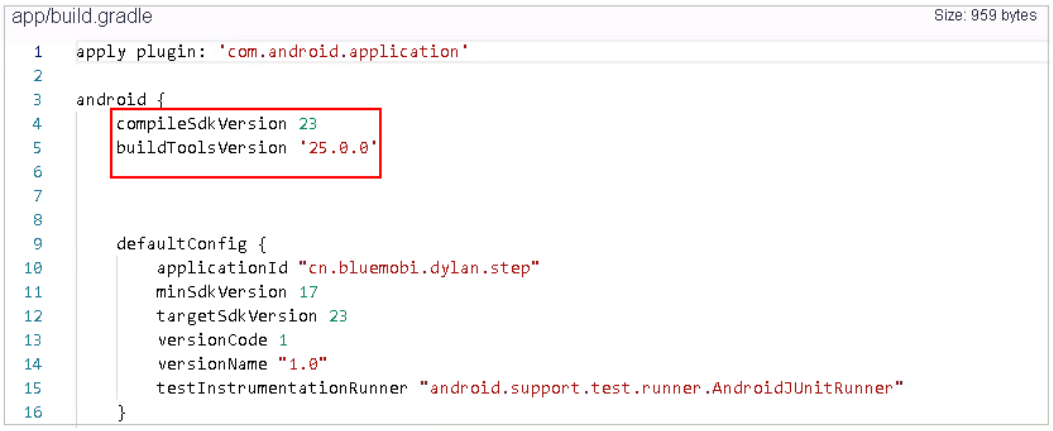
Parameter	Description
Action Name	Name of a build action. It can be customized.
Gradle	Select a Gradle version.
JDK	Select a JDK version.
NDK	Select an NDK version as required. You can also select **No**.

Parameter	Description
Commands	Configure the Gradle commands. You can also use default commands.

Android Version Description

- SDK: used to specify compileSdkVersion.
- Build Tools: used to specify buildToolsVersion.

You can find the two versions in the **build.gradle** file or the global configuration file (user-defined) of the project.


app/build.gradle Size: 959 bytes

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 23
    buildToolsVersion '25.0.0'



    defaultConfig {
        applicationId "cn.bluemobi.dylan.step"
        minSdkVersion 17
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
```

NOTE

- Select compileSdkVersion or buildToolsVersion based on project requirements.
- The Gradle wrapper build mode is also supported. If the provided Gradle version does not meet your requirements, you can run the **gradlew** command for build using the wrapper. The required Gradle version will be automatically downloaded. Example of the build command: **./gradlew clean build**.

6.2.5 Sign Android APK

With the **Sign Android APK** action, use apksigner to sign the Android APK.

Configuration Description

1. Add **Sign Android APK** after **Build with Android**, when **configuring build actions**.

   The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
APK Location	Location of the APK file to be signed generated after Android building. Regular expressions are supported. For example, **build/bin/*.apk** can be used to match the built APK package.

Parameter	Description
Keystore File	Keystore file used for signature, which is generated by referring to **Generating Keystore Signature Files**. Select a keystore file from the list of files already uploaded after you click **Manage Files**.
Keystore Password	Keystore password.
Alias	Alias of the keystore file.
Key Password	Password of the key.
apksigner Command	Custom signature parameter. The default value is **--verbose**.

2. Check whether the signing is successful.

   After the configuration is complete, run the build task. After the task is executed successfully, view the build log. If "result: Signed" is displayed in the Android APK signature log, the signing is successful.

6.2.6 Build with npm

Build Vue and Webpack projects with npm.

Configuration Description

Add **Build with npm**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the npm commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.7 Build with Gradle

Build a Java, Groovy, or Scala project with Gradle.

Configuration Description

Add **Build with Gradle**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Gradle	Select a Gradle version.
JDK	Select a JDK version.
Commands	Configure the Gradle commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.8 Build with Yarn

Build a JavaScript project with Yarn.

Add **Build with Yarn**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the Yarn commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.9 Build with Gulp

Build a frontend IDE with Gulp.

Add **Build with Gulp**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the Gulp commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.10 Build with Grunt

Build a JavaScript project with Grunt.

Add **Build with Grunt**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the Grunt commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.11 Build with Mono

Build a project with MSBuild and .NET on Mono Linux for x86 and Arm.

Add **Build with Mono**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the Mono commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.12 Build in PHP

Install PHP and Composer for PHP libraries.

Add **Build in PHP**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the PHP commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.13 Build with Setuptools

Build a Python project with setuptools.

Prerequisites

When using setuptools to pack the code, ensure that the **setup.py** file exists in the root directory of the code. For details on how to write the file, see the **official instructions of Python**.

Configuration Description

Add **Build with Setuptools**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the build commands. ● You can use the default commands to pack the file into an **.egg** file. ● For Python 2.7 or later, it is advised to use **python setup.py sdist bdist_wheel** to pack the source code package and **.whl** installation package for pip installation.

6.2.14 Build with PyInstaller

Build a Python project with PyInstaller.

Configuration Description

Add **Build with PyInstaller**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Configure the build packaging command. The default command is to package the project into an executable file. For details about the PyInstaller command, visit the PyInstaller website for official documentation.

6.2.15 Run Shell Commands

Add **Run Shell Commands**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Enter the command as required.

6.2.16 Build with GNU Arm

Design, develop, and use an Arm simulator with the GNU Arm embedded toolchain.

Configuration Description

Add **Build with GNU Arm**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select an Arm tool version.

Parameter	Description
Commands	Configure the GNU Arm commands. You can also use default commands. ● If Makefile is not in the root directory of the code, run the **cd** command to access the correct directory and then run the **make** command. ● If you do not want to run the **make** command, you can refer to the build commands provided by the following images: – gnuarm201405 image Use the **arm-none-linux-gnueabi-gcc** command, for example, **arm-none-linux-gnueabi-gcc -o main main.c**. – gnuarm-linux-gcc-4.4.3 image Use the **arm-linux-gcc** command, for example, **arm-linux-gcc -o main main.c**. – gnuarm-7-2018-q2-update image Use the **arm-none-eabi-gcc** command, for example, **arm-none-eabi-gcc --specs=nosys.specs -o main main.c**. **NOTE** ● For details about how to write the GNU makefile in Linux, see the **official website**. ● Makefile contains only line comment tags (#). If you want to use or output the number sign (#), escape the number sign, for example, using \#.

6.2.17 Build with CMake

Build a cross-platform project with CMake.

Configuration Description

Add **Build with CMake**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select the CMake build tool version.
Commands	Configure the CMake commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.18 Build with Ant

Apache Ant is a tool used to compile, test, and deploy Java projects.

Prerequisites

The project is in the Ant structure using the Java language, and a correct build description file **build.xml** is available.

Configuration Description

Add **Build with Ant**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	The recommended version is used by default. You can select the Ant and JDK image versions that match your compilation environment.
Commands	Configure the Ant commands. You can also use default commands. If there are special build requirements, you can enter your custom build script in the text box.

6.2.19 Build with Go

Build a Go project.

Prerequisites

The project is developed using the Go language, and the build description file exists in the code.

Configuration Description

Add **Build with Go**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version. The recommended version has been selected by default. You can select the Go version that matches your build environment.

Parameter	Description
Commands	Configure the Go project build command. You can also use default commands. If there are special build requirements, enter a customized build script in the text box.

6.2.20 Build Android Quick App

Use the **npm config set xxx** command.

Configuration Description

Add **Build Android Quick App**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a build tool version.

Commands	Configure commands. The following is an example of using the debug signature for packing.
	To sign a quick app, perform the following steps:
	1. Run the **openssl** command to generate the signature files **private.pem** and **certificate.pem**. Example:
	`openssl req -newkey rsa:2048 -nodes -keyout private.pem -x509 -days 3650 -out certificate.pem`
	Create the **release** directory in the **sign** directory of the project and copy the private key file **private.pem** and certificate file **certificate.pem** to the directory.
	2. Before releasing the program package, add the release signature and run the following command in the root directory of the project:
	`npm run release`
	The generated application directory is **/dist/.release.rpk**.
	3. To temporarily use the debug signature, run the following command:
	`npm run release -- --debug`
	NOTE
	The debug signature is open and not necessarily secure. Therefore, do not use the debug signature to sign an application to be officially released.

6.2.21 Creating Images and Pushing to SWR

CodeArts Build provides a large number of default build actions and templates. If necessary dependency packages and tools are missing, you can create an image from a Dockerfile and push it to the specified repository in SWR.

This document uses Maven build as an example.

Prerequisites

- You have **created an organization** in SWR. For details about organization restrictions, see **Notes and Constraints**.
- You have created a code repository by using system template Java Maven Demo. For details, see **Creating a Repository Using a Template**.
- You have **customized a build environment** and upload the Dockerfile and other files required for image creation to the root directory of the code repository.

Configuration Description

Add **Build Image and Push to SWR** after **Build with Maven**, when **configuring build actions**.

In the **Build with Maven** action, retain default values of the parameters. In the **Build Image and Push to SWR** action, set the parameters as described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select the tool version. You can also use the default version.
Image Repository	By default, CodeArts Build provides the SWR repository address of each region. You do not need to change the address. **NOTE** Images can be pushed to custom image repositories.
Authorized User	Current user. Ensure that you have the permissions to edit or manage all images in the organization. For details, see **User Permissions**.
Organization	Enter the organization name of the SWR repository created in **Prerequisites**.
Image Name	Name of the created image, which can be customized.
Image Tag	Specify the image tag, which can be customized. You can use *Image name*:*Tag* to uniquely specify an image.
Working Directory	The context path parameter in the docker build command is the relative path of the root directory of the CodeArts Repo code repository. Context path: When Docker builds an image, the docker build command packs all content in the path and sends it to the container engine to help build the image.

Parameter	Description
Dockerfile Path	Path of the Dockerfile. Set this parameter to a path relative to the working directory. For example, if the working directory is a root directory and the Dockerfile is in the root directory, set this parameter to **./Dockerfile**.
Add Build Metadata to Image	Add the build information to the image. After the image is created, run the docker inspect command to view the image metadata.

6.2.22 Using SWR Public Images

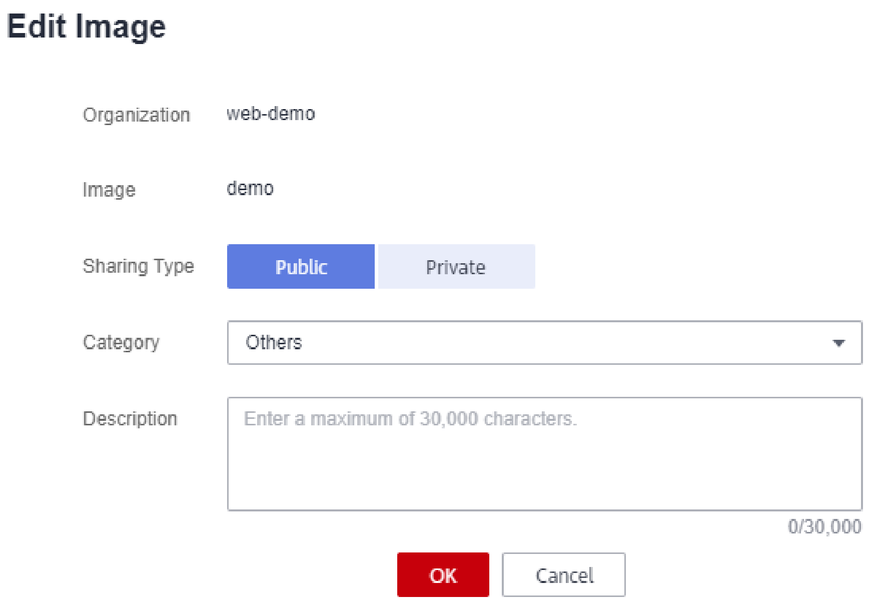
Prerequisites

You have **created an image and push it to SWR**.

Procedure

Step 1 Images in SWR cannot be pulled during building. Therefore, you need to set the image type to **Public**.

1. Log in to **SWR**.
2. In the navigation pane, choose **My Images**, click the image name to go to the image details page, and click **Edit** in the upper right corner.
3. In the dialog box, set the type to be public.


Edit Image

Organization web-demo

Image demo

Sharing Type [✓ Public] Private

Category Others

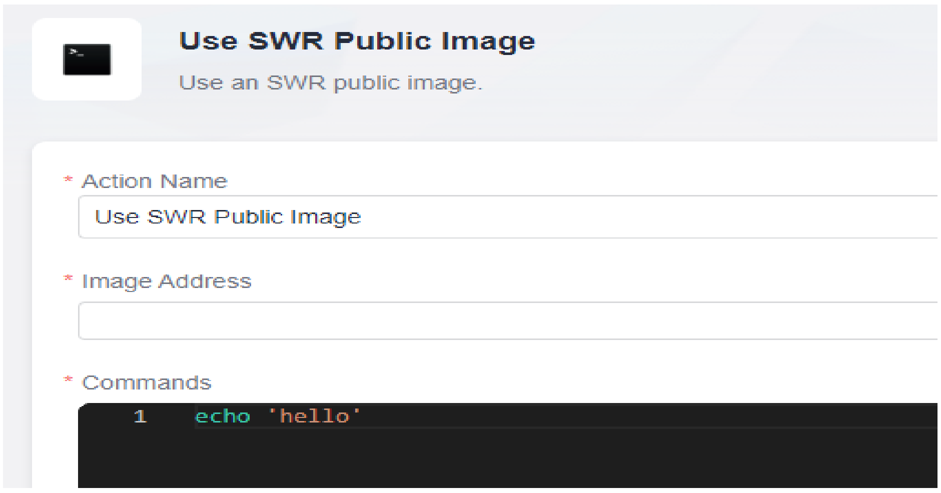
Description Enter a maximum of 30,000 characters.

0/30,000

4. To obtain the complete image path, click ⧉ to copy the image download command. The part following **docker pull** is the image path.

Step 2 Add **Use SWR Public Image**, when **configuring build actions**.

Step 3 Paste the image address obtained in step 1 to the text box.



```
echo 'hello'
```

NOTE

When pasting the download command to the image address text box, delete **docker pull**.

Step 4 Enter the build commands in the command window and run the build task commands to complete the build.

----End

6.2.23 Uploading Software Packages to Release Repos

To upload generated software packages to release repos, add **Upload to Release Repos** when configuring build actions.

NOTE

When you choose Windows executors, add action **Upload Software Package to Release Repos (Windows)**.

- Only one or more files can be uploaded. Folders cannot be uploaded and directories cannot be automatically created.

  For example, the **a** directory contains the **aa** file and **b** directory that contains the **bb** file, and the build package directory is set to **a/****.

  When the **a** directory is scanned, both **aa** and **bb** will be uploaded to the same directory, and the system will not create a **b** directory in release repos.
- To upload a folder, package it before adding the **Upload to Release Repos** action. You can package the folder by running the packaging command or adding the **Run Shell Commands** action.
- For details about the restrictions on the uploaded software packages, see Constraints of CodeArts Artifact.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.

Parameter	Description
Package Location	Directory for storing the build result. A regular expression is supported. Example: ****/target/*.?ar** uploads all JAR and WAR packages built with Maven.
Version	Directory for storing the build package in a release repo. Leave this blank to use the build ID for the directory (recommended).
Package Name	New name for files uploaded to the release repo. Leave this parameter blank to use the original file names (recommended).

Parameter Settings

- **Build Package Directory**

  The build package directory supports regular expression matching. ****** means that the system recursively traverses the current directory. ***** indicates zero or multiple characters. **?** indicates one character.

  The system file separator is a slash **/**, and the path is case-insensitive.

  Examples:

  – *.class

    Matches files whose names end with **.class** in the current directory.

  – **/*.class

    Recursively matches all files whose names end with **.class** in the current directory.

  – test/a??.java

    Matches Java files whose names start with **a** followed by two characters in the **test** directory.

  – **/test/**/XYZ*

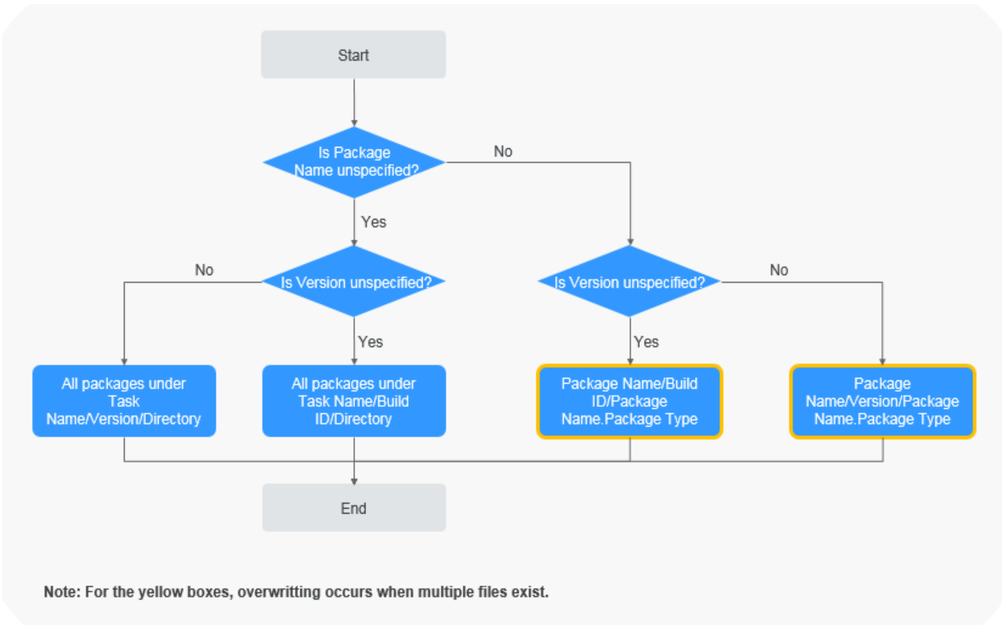
    Recursively matches all files whose parent directory is **test** and whose names start with **XYZ**, for example, **abc/test/def/ghi/XYZ123**.

- **Release Version and Package Name**

  Leave **Package Name** unspecified so that all files matching the build package directory can be uploaded.

  After the package name is set, overwriting may occur if multiple files are matched. If the package name needs to be set and multiple files need to be uploaded, add the uploading action for multiple times.

  The figure below illustrates the impact of an unspecified release version and package name on uploads.


Start
Is Package Name unspecified?
No
Yes
Is Version unspecified?
No
Yes
Is Version unspecified?
No
Yes
All packages under Task Name/Version/Directory
All packages under Task Name/Build ID/Directory
Package Name/Build ID/Package Name.Package Type
Package Name/Version/Package Name.Package Type
End
Note: For the yellow boxes, overwritting occurs when multiple files exist.

6.2.24 Uploading Files to OBS

For details about the restrictions on using OBS, see **Restrictions and Limitations**.

Configuration Description

Add **Upload Files to OBS**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Authorized Account	● **Current**: Upload files to an OBS bucket of the current account. ● **Other**: Upload files to an OBS bucket of a specific account by using an IAM account.
Build Directory	Directory for storing build results. If no file name is specified for OBS storage, use wildcard characters to upload multiple files. Example: ****/target/*.?ar** uploads all JAR and WAR packages built with Maven.
Bucket Name	Name of the target OBS bucket. Cross-region upload is not supported.
OBS Directory	Directory for storing build results on OBS (for example, **application/version/**). You can leave this parameter blank or enter **./** to store build results to the OBS root directory.
File Name	New name (excluding the directory) for the built file after OBS storage. Leave it blank to upload multiple files with their old names, or specify a name to upload a single file, for example, **application.jar**.

Parameter	Description
Headers	Add one or more custom response headers during file upload. The headers will be included in the response to download objects or query the object metadata. For example, you can set the key to **x-frame-options** and value to **false** to prevent web pages stored in OBS from being embedded into third-party web pages.

6.2.25 Running Docker Commands

Add **Run Docker Commands**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Commands	Click **Add** to add a command, and configure the command as required. View the **Docker Docs**.

6.2.26 Downloading Package from Release Repos

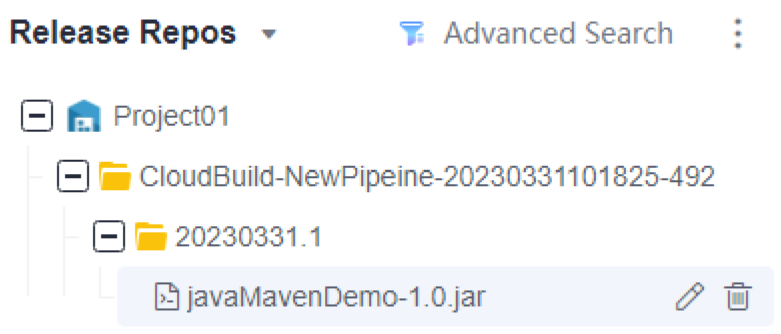
By configuring the action **Download from Release Repos**, you can download the packages or other files in the release repos to the root directory of a build task so that these packages or files can be used in subsequent build actions.

Obtaining the Download Address

Step 1 Log in to CodeArts.

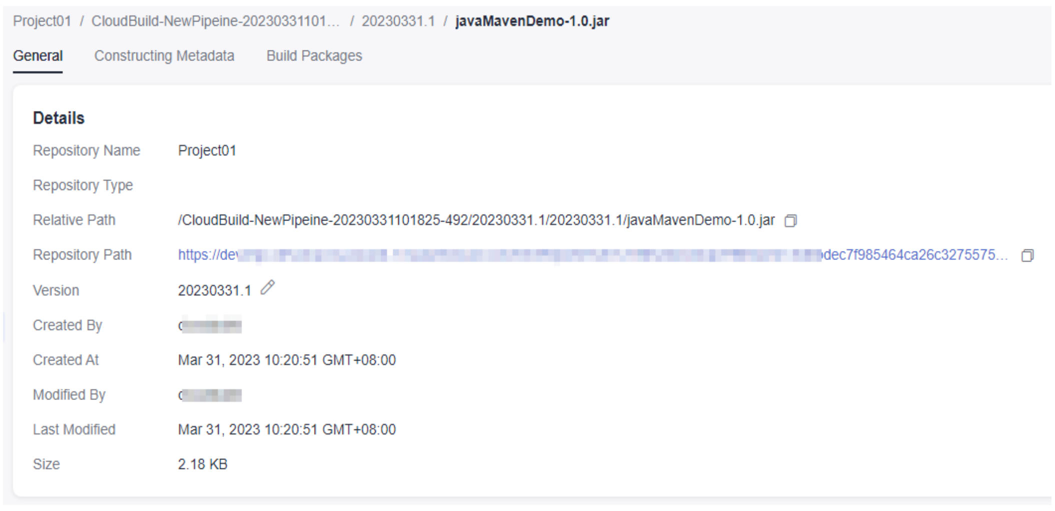
Step 2 Search for the target project and click the project name. In the navigation pane, choose **Artifact** > **Release Repos**.

Step 3 On the **Release Repos** page, search for the repository package to download.



Step 4 Click the name of the package. The package details page is displayed.

The repository path is the address for downloading the package. Click ⧉ to copy the address.



Project01 / CloudBuild-NewPipeine-20230331101... / 20230331.1 / **javaMavenDemo-1.0.jar**

General | Constructing Metadata | Build Packages

Details

Repository Name	Project01
Repository Type	
Relative Path	/CloudBuild-NewPipeine-20230331101825-492/20230331.1/20230331.1/javaMavenDemo-1.0.jar
Repository Path	https://de[redacted]dec7f985464ca26c3275575...
Version	20230331.1
Created By	c[redacted]
Created At	Mar 31, 2023 10:20:51 GMT+08:00
Modified By	c[redacted]
Last Modified	Mar 31, 2023 10:20:51 GMT+08:00
Size	2.18 KB

----End

Configuring the Download Action

Add **Download Package from Release Repos** when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
Package Address	Paste the package download address copied in **step 4** to the text box.

6.2.27 Downloading File from File Manager

File management stores Android APK signature files and the **settings.xml** files of Maven build, and manages these files (For example, you can create, edit, and delete these files, and modify users' permissions on them). For details about how to upload files, see **File Management**. Add the **Download File from File Manager** action to download files from **Files** to the working directory for use.

Configuration Description

Add **Download File from File Manager**, when **configuring build actions**.

The parameters are described in the following table.

Parameter	Description
Action Name	Name of a build action. It can be customized.
Tool Version	Select a tool version.
File Name	● Select an uploaded file from the drop-down list. ● Click **Upload** to upload a local file to File Manager. ● Click **Manage Files** to manage files on the **Files** page.

6.3 Code-based Build

6.3.1 Configuring a Task

6.3.1.1 Introducing the YAML File Structure

YAML File Example

In this example, the Maven build template is executed using the YAML file.

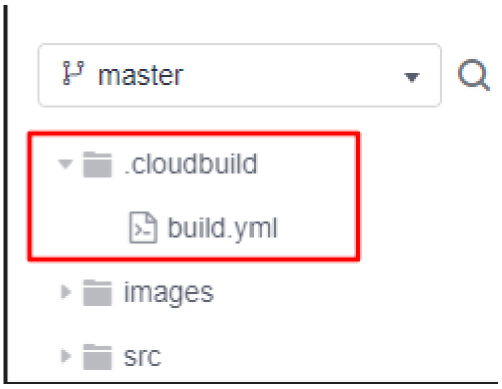
```
version: 2.0 # The value must be 2.0.
params: # Build parameters, which can be referenced during a build.
  - name: paramA
    value: valueA
  - name: paramB
    value: valueB
env: # This parameter is optional but has the highest priority. The host specifications and type (if any)
defined here will be used instead of those you configured on the basic information page for a task.
  resource:
    type: docker # The agent pool type can be: Docker, Linux, macOS or custom ones.
    arch: X86 # The host type of the build environment can be: x86 or Arm.
    class: 8 vCPUs | 16 GB # The specification can be: 2 vCPUs | 8 GB, 4 vCPUs | 8 GB, 8 vCPUs | 16 GB, 16
vCPUs | 32 GB, or 16 vCPUs | 64 GB. This parameter is not required when the agent pool type is set to a
custom one.
    pool: Mydocker # Agent pool name. This parameter is required when the agent pool type is set to a
custom one.
steps:
  PRE_BUILD:
    - checkout:
        name: Download Code # This field is optional.
       inputs: # Action parameters
          scm: codehub # Code source: Repo only
          url: xxxxxxxxx # SSH address for the URL to pull code
          branch: ${codeBranch} # Pulled code branch, which can be parameterized.
    - sh:
        inputs:
          command: echo ${paramA}
  BUILD:
    - maven: # Action keyword. Only specified keywords are supported.
    name: maven build # Optional
    image: xxx # You can customize the image path. For details, see the following description.
    inputs:
      command: mvn clean package
    - upload_artifact:
    inputs:
          path: "**/target/*.?ar"
```

The **.yml** file consists of four parts:

- **Version**: In the example file, **version** is set to **2.0**. The version number is mandatory and unique.
- Build environment **(env)**: The sample file defines the resource pool type, build environment host type, and host specifications.
- Build parameters (**params**): Optional. **paramA** and **paramB** are defined in the sample file and can be referenced during the build process. Build parameters created during task configuration are preferentially used.
- Build actions **(steps)**: In the example file, there are three phases under the steps level:
  – **PRE_BUILD**: used to prepare for building, for example, downloading code and executing shell.
  – **BUILD**: used to build mainstream projects such as Maven, npm, Go, Python, Ant, CMake, Mono, sbt, Android and Bazel. After the build is complete, you can define post-build operations, such as creating images and uploading them to SWR, uploading files to OBS, downloading files, uploading binary packages to the specified repository, downloading binary packages, and running Docker commands.

NOTE

- After the **build.yml** file is defined, strictly comply with the file storage path rules and place the **build.yml** file in the **.cloudbuild** path.



- The image can be in either of the following formats:
  - **cloudbuild@maven3.5.3-jdk8-open**, which starts with **cloudbuild**, uses **@** as the separator, and is followed by the default image provided by CodeArts Build.
  - Complete SWR image path, for example, **swr.example.example.com/codeci_test/demo:141d26c455abd6d7***xxxxxxxxxxxxxxxxxxxx*.

Build Procedure

In steps, there are two phases: **PRE_BUILD** and **BUILD**. Each phase can define a series of build steps. For details, see the following table.

PRE_BUILD	Description	Operation Guide
- checkout	Download code	**Using YAML to Configure Code Download**

PRE_BUILD	Description	Operation Guide
- sh	Run Shell commands	**Using YAML to Configure and Execute Shell Commands**

BUILD	Description	Operation Guide
- maven	Build with Maven	**Using YAML to Configure a Maven Build**
- npm	Build with npm	**Using YAML to Configure an npm Build**
- go	Use Go for build	**Using YAML to Configure a Build with Go**
- python	Use Python for build	**Using YAML to Configure a Python Build**
- gradle	Build with Gradle	**Using YAML to Configure a Gradle Build**
- ant	Build with Ant	**Using YAML to Configure an Ant Build**
- cmake	Build with CMake	**Using YAML to Configure a CMake Build**
- mono	Build with Mono	**Using YAML to Configure a Mono Build**
- sbt	Build with sbt	**Using YAML to Configure an sbt Build**
- android	Build with Android	**Using YAML to Configure an Android Build**
- bazel	Build with Bazel	**Using YAML to Configure a Bazel Build**
- build_image	Create container images and upload them to SWR	**Using YAML to Create an Image and Upload It to SWR**
- upload_obs	Upload a file to OBS	**Using YAML to Upload Files to OBS**
- download_file	Download a file	**Using YAML to Download Files**
- upload_artifact	Upload the binary package to the repository	**Using YAML to Upload Binary Packages to a Repository**

BUILD	Description	Operation Guide
- download_artifact	Download the binary package	**Using YAML to Download Binary Packages**
- docker	Run Docker commands	**Using YAML to Run Docker Commands**

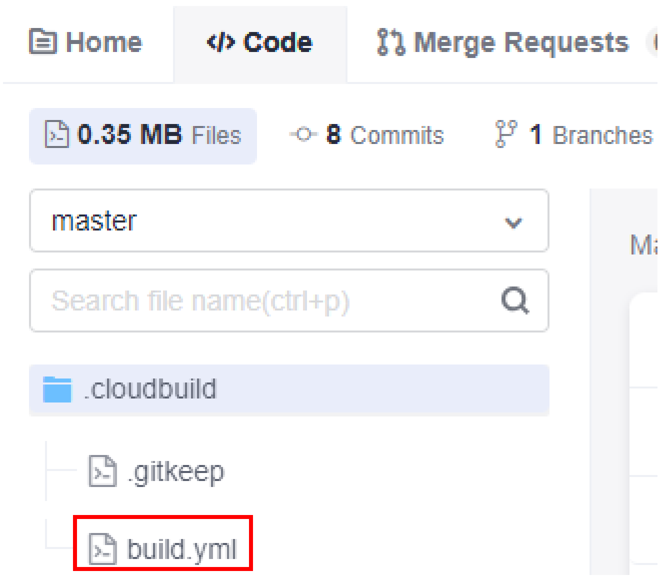
6.3.1.2 Using YAML for Build

Prerequisites

- A project is available. If no project is available, **create one**.
- A code repository has been created in the project. If no code repository is available, **create one**.
- In the code repository, create the **.cloudbuild** directory and store the YAML file in the directory. For details about how to write the YAML file and its specifications, see **Introducing the YAML File Structure**.

  NOTE

  If the YAML file is not stored in the **.cloudbuild** directory, you can use **CB_BUILD_YAML_PATH** to specify the path of the YAML file in the code repository.



Selecting a Code Source

1. **Log in to the CodeArts Build homepage**.
2. Click **Create Task**. The **Basic Information** page is displayed.
3. Select **Repo** as the source code source and configure the source code repository and branch to be used.

Configuring and Executing the YAML Build Task

1. Click **Next**. The **Build Template** page is displayed.
2. Select **Blank Template** and click **Next**.

   Or select a recommended template. This does not affect the YAML build.
3. Go to the **Build Actions** tab page. In the upper left corner, select the **Code** tab.

   You can modify the YAML file here. The system automatically reads the YAML file in the code repository and branch configured when you **select a code source**.
4. After the configuration, click **Create** in the upper right corner.
5. Click **Create and Run**. Changes to the YAML file take effect and the YAML build task file is executed. After the build script is submitted, the original **build.yml** file is overwritten.

6.3.1.3 Using YAML to Configure Code Download

The following configurations are for your reference.

```
version: 2.0 # The value must be 2.0.
steps:
  PRE_BUILD:
    - checkout:
        name: checkout
        inputs:
          scm: codehub # Code source: Repo and open source
          url: xxxxxxxxx # SSH address for the URL to pull code
          branch: ${codeBranch} # Mandatory at any time and can be parameterized.
          commit: ${commitId}
          lfs: true
          submodule: true
          depth: 100
          tag: ${tag}
          path: test
```

The parameters are described in the following table.

Parameter	Type	Description	Mandatory	Default Value
scm	string	Code source. Currently, only CodeArts Repo is supported. If this parameter is not configured in the YAML file, the code repository information configured in build task is used.	No	codehub
url	string	SSH address for pulling code	Yes	None
branch	string	Pulled code branch, which can be parameterized.	Yes	None
commit	string	Commit ID obtained during builds can be parameterized.	No	None

Parameter	Type	Description	Mandatory	Default Value
tag	string	Tag pulled during tag builds: It can be parameterized. If a commit ID and a tag exist at the same time, the build based on a commit ID is executed first.	No	None
depth	int	Shallow clone depth. When a commit ID is specified for builds, **depth** must be greater than or equal to the depth of the commit ID.	No	1
submodule	bool	Whether to pull the submodule. The options are **true** (pull) and **false** (not pull).	No	false
lfs	bool	Whether to enable Git LFS: If this parameter is set to **true**, Git LFS pull is executed.	No	false
path	string	Subpath for cloning: The code is downloaded to the subpath.	No	None

6.3.1.4 Using YAML to Configure Manifest File Download from Repositories

In scenarios such as Android and Harmony, hundreds or even thousands of code repositories need to be integrated at the same time during one build. The integration and download efficiency of multiple code repositories is critical.

CodeArts Build has integrated the CodeArts Repo download tool. You only need to perform simple configurations to download multiple code repositories. Currently, Repo and Gerrit are supported.

The following configurations are for your reference.

```
version: 2.0 # The value must be 2.0.
steps:
  PRE_BUILD:
  - manifest_checkout:
      name: "manifest"
      inputs:
        manifest_url: "https://example.example.example.example.example.com/xx/manifest.git"
        manifest_branch: "master"
        manifest_file: "default.xml"
        path: "dir/dir02"
        repo_url: "https://example.example.example.example.example.com/xx/git-repo.git"
        repo_branch: "master"
        username: "someone"
        password: "${PASSWD}"
```

The parameters are described in the following table.

Parameter	Type	Description	Mandatory	Default Value
name	string	Step name.	No	manifest_checkout
manifest_url	string	Specifies the manifest repository address, including the repository of XML files.	Yes	None
manifest_branch	string	Specifies a manifest branch or revision.	No	HEAD
manifest_file	string	Manifest file path.	No	default.xml
path	string	Download path of all sub-repositories of the customized manifest file, which is the relative path of the working path. The path cannot start with a slash (/) and cannot contain any period (.).	No	The default value is the working path.
repo_url	string	Repo repository address.	No	**https://gerrit.googlesource.com/git-repo**
repo_branch	string	Repo repository branch.	No	stable
username	string	Username for downloading the repository.	No. This parameter is mandatory when a non-public repository is downloaded.	None

Parameter	Type	Description	Mandatory	Default Value
password	string	HTTPS password used for downloading the repository.	No. This parameter is mandatory when a non-public repository is downloaded.	None

NOTE

1. The repositories defined in manifest_file must be of the same source code source.
2. **manifest_url** and **manifest_file** must use the same code source. For a non-public repository, **username** and **password** must have the download permission.
3. The repo repository corresponding to **repo_url** must have the download permission (the repository is open-source, or the repository is private but configured with an account and password).
4. If the values of the preceding optional parameters are empty, the default values are used.
5. When a non-public repository is used, you are advised to configure the username and password using the constructed private parameters. For details, see **Parameter Settings**.

6.3.1.5 Using YAML to Configure and Execute Shell Commands

```
version: 2.0 # The value must be 2.0.
steps:
  PRE_BUILD:
    - sh:
        inputs:
          command: echo ${a}
```

The parameters are described in the following table.

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.6 Using YAML to Configure a Maven Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - maven:
          image: cloudbuild@maven3.5.3-jdk8-open # You can customize the image path. For details, see the
following description.
```

```
    inputs:
      settings:
        public_repos:
          - https://mirrors.example.com/maven
      cache: true # Indicates whether to enable the cache.
      unit_test:
        coverage: true
        ignore_errors: false
        report_path: "**/TEST*.xml"
        enable: true
        coverage_report_path: "**/site/jacoco"
      command: mvn package -Dmaven.test.failure.ignore=true -U -e -X -B
```

NOTE

The image can be in either of the following formats:

1. **cloudbuild@maven3.5.3-jdk8-open**, which starts with **cloudbuild**, uses **@** as the separator, and is followed by the default image provided by CodeArts Build.

2. Complete SWR image path, for example, **swr.example.example.com/codeci_test/demo:141d26c455abd6d7***xxxxxxxxxxxxxxxxxxxx*.

Configuration Description

Parameter	Type	Description	Mandatory	Default Value
settings	map	Setting for Maven builds.	No	None
cache	bool	Whether to enable cache.	No	false
command	string	Execute commands.	Yes	None
unit_test	map	Unit test.	No	None

Parameters for unit_test are described in the following table.

Parameter	Type	Description	Mandatory	Default Value
enable	bool	Whether to process test data.	No	true
ignore_errors	bool	Whether to ignore unit test errors.	No	true
report_path	String	Unit test data path.	Yes	None
converage	bool	Whether to process coverage data.	No	false

Parameter	Type	Description	Mandatory	Default Value
coverage_report_path	string	Coverage data path.	No	None

6.3.1.7 Using YAML to Configure an npm Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - npm:
        inputs:
          command: |
              export PATH=$PATH:~/.npm-global/bin
              npm config set registry https://repo.example.com/repository/npm/
              npm config set disturl https://repo.example.com/nodejs
              npm config set sass_binary_site https://repo.example.com/node-sass/
              npm config set phantomjs_cdnurl https://repo.example.com/phantomjs
              npm config set chromedriver_cdnurl https://repo.example.com/chromedriver
              npm config set operadriver_cdnurl https://repo.example.com/operadriver
              npm config set electron_mirror https://repo.example.com/electron/
              npm config set python_mirror https://repo.example.com/python
              npm config set prefix '~/.npm-global'
              npm install --verbose
              npm run build
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.8 Using YAML to Configure a Yarn Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - yarn:
       inputs:
          command: |-
    #If the Node.js version is earlier than 18, the settings can be as follows:
            npm config set cache-folder /yarncache
            npm config set registry http://mirrors.tools.huawei.com/npm/
            npm config set disturl http://mirrors.tools.huawei.com/nodejs
            npm config set sass_binary_site http://mirrors.tools.huawei.com/node-sass/
            npm config set phantomjs_cdnurl http://mirrors.tools.huawei.com/phantomjs
            npm config set chromedriver_cdnurl http://mirrors.tools.huawei.com/chromedriver
            npm config set operadriver_cdnurl http://mirrors.tools.huawei.com/operadriver
            npm config set electron_mirror http://mirrors.tools.huawei.com/electron/
            npm config set python_mirror http://mirrors.tools.huawei.com/python

            #If the Node.js version is 18 or later, the settings can be as follows:
             #npm config set registry http://mirrors.tools.huawei.com/npm/
             npm config set prefix '~/.npm-global'
            export PATH=$PATH:~/.npm-global/bin
            #yarn add node-sass-import --verbose
            yarn install --verbose
```

```
      yarn run build
      tar -zcvf demo.tar.gz ./**
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.9 Using YAML to Configure a Build with Go

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - go:
        inputs:
          command: |
            export GO15VENDOREXPERIMENT=1
            export GOPROXY=https://goproxy.cn
            mkdir -p $GOPATH/src/example.com/demo/
            cp -rf . $GOPATH/src/example.com/demo/
            go install example.com/demo
            cp -rf $GOPATH/bin/ ./bin
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.10 Using YAML to Configure a Build with Gulp

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - gulp:
        inputs:
          command: |-
            export PATH=$PATH:~/.npm-global/bin
            npm config set registry http://mirrors.tools.huawei.com/npm/
            npm config set prefix '~/.npm-global'
            #If node-sass needs to be installed:
            #npm config set sass_binary_site https://repo.huaweicloud.com/node-sass/
            #npm install node-sass
            #Load dependencies.
            npm install -verbose
            gulp
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.11 Using YAML to Configure a Build with Grunt

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - grunt:
          inputs:
            command: |-
              npm config set registry http://7.223.219.40/npm/
              #npm cache clean -f
              #npm audit fix --force
              npm install --verbose
              grunt
              npm run build
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.12 Using YAML to Configure a PHP Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - php:
        inputs:
          command: |-
            composer config -g secure-http false
            composer config -g repo.packagist composer http://mirrors.tools.huawei.com/php/
            composer install
            tar -zcvf php-composer.tgz *
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.13 Using YAML to Configure a Build with GNU Arm

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - gnu_arm:
        inputs:
          command: make
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.14 Using YAML to Configure a Build with Setuptools

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - python:
        name: Build with Setuptools
        image: cloudbuild@python3.6
        inputs:
          command: |
            pip config set global.index-url https://pypi.org/simple
            pip config set global.trusted-host repo.xxcloud.com
            python setup.py bdist_egg
```

Parameter	Type	Description	Mandatory	Default Value
name	/	Name of a build action. It can be customized.	No	None
image	/	Image version. **cloudbuild@** is a fixed part, followed by the supported Python version. You can view the tool versions supported for **Build with Setuptools** in the graphical build mode.	No	cloudbuild@python3.6
command	string	Execute commands and enter required code.	Yes	None

6.3.1.15 Using YAML to Configure a Build with PyInstaller

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - python:
        name: Build with PyInstaller
        image: cloudbuild@python3.6
        inputs:
          command: |
            pip config set global.index-url https://pypi.org/simple
            pip config set global.trusted-host repo.xxcloud.com
            # Create a single executable file in the dist directory with -F.
            # For command details, see https://pyinstaller.readthedocs.io/en/stable/usage.html.
            pyinstaller -F  *.py
```

Parameter	Type	Description	Mandatory	Default Value
name	/	Name of a build action. It can be customized.	No	None
image	/	Image version. **cloudbuild@** is a fixed part, followed by the supported Python version. You can view the tool versions supported in **Build with PyInstaller** in the graphical build mode.	No	cloudbuild@python3.6

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands and enter required code.	Yes	None

6.3.1.16 Using YAML to Configure a Python Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - python:
        inputs:
          command: |
              pip config set global.index-url https://pypi.org/simple
              pip config set global.trusted-host repo.xxcloud.com
              python setup.py bdist_egg
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.17 Using YAML to Configure a Gradle Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - gradle:
        inputs:
            gradle: 4.8
            jdk: 1.8
            command: |
                # Gradle Wrapper provided by CodeArts and cache are used for acceleration.
                cp /cache/android/wrapper/gradle-wrapper.jar ./gradle/wrapper/gradle-wrapper.jar
                # Build an unsigned APK.
                /bin/bash ./gradlew build --init-script  ./.codeci/.gradle/init_template.gradle -
Dorg.gradle.daemon=false -Dorg.gradle.internal.http.connectionTimeout=800000
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None
gradle	string	Gradle version.	Yes	None
jdk	string	JDK version.	Yes	None

6.3.1.18 Using YAML to Configure an Ant Build

```
version: 2.0 # The value must be 2.0.
steps:
```

```
BUILD:
  - ant:
      inputs:
        command:  ant -f build.xml
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.19 Using YAML to Configure a CMake Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - cmake:
        inputs:
          command: |
            # Create the build directory and switch to the build directory.
            mkdir build && cd build
            # Generate makefiles for the Unix platform and perform the build.
            cmake -G 'Unix Makefiles' ../ && make -j
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.20 Using YAML to Configure a Mono Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - mono:
        inputs:
          command: |
            nuget sources Disable -Name 'nuget.org'
            nuget sources add -Name 'xxcloud' -Source 'https://repo.xxcloud.com/repository/nuget/v3/
index.json'
            nuget restore
            msbuild /p:OutputPath=../buildResult/Release/bin
            zip -rq ./archive.zip ./buildResult/Release/bin/*
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.21 Using YAML to Configure a Build with Flutter

```
version: 2.0 # The value must be 2.0.
steps:
```

```
BUILD:
  - flutter:
     inputs:
        flutter: cn-north-1
        jdk: '3333'
        ndk: '23.1.7779620'
        command: ./instrumented.apk
```

Parameter	Type	Description	Mandatory	Default Value
flutter	string	Region name.	Yes	None
jdk	string	JDK file name.	Yes	None
ndk	string	NDK file name.	Yes	None
command	string	Execute commands.	Yes	None

6.3.1.22 Using YAML to Configure an sbt Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - sbt:
        inputs:
            command: |
                sbt package
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.23 Using YAML to Configure an Android Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - android:
        inputs:
          gradle: 4.8
          jdk: 1.8
          ndk: 17
          command: |
            cat ~/.gradle/init.gradle
            cat ~/.gradle/gradle.properties
            cat ~/.gradle/init_template.gradle
            rm -rf ~/.gradle/init.gradle
            rm -rf /home/build/.gradle/init.gradle
            # Gradle Wrapper provided by CodeArts and cache are used for acceleration.
            cp /cache/android/wrapper/gradle-wrapper.jar ./gradle/wrapper/gradle-wrapper.jar
            # Build an unsigned APK.
            /bin/bash ./gradlew assembleDebug -Dorg.gradle.daemon=false -d --stacktrace
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None
gradle	string	Gradle version.	Yes	None
jdk	string	JDK version.	Yes	None
ndk	string	NDK version.	Yes	None

6.3.1.24 Using YAML to Sign Android APK

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - android_sign:
        inputs:
           file_path: build/bin/*.apk
           keystore_file: androidapk.jks
           keystore_password: xxxxxx
           alias: keyalias
           key_password: xxxxxx
           apksigner_commond: --verbose
```

Parameter	Type	Description	Mandatory	Default Value
file_path	string	APK directory.	Yes	None
keystore_file	string	Keystore file name.	Yes	None
alias	string	Alias.	Yes	None
apksigner_commond	string	apksigner CLI.	Yes	None

6.3.1.25 Using YAML to Inject an APM Probe into an Android App

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - apm_probe:
       inputs:
          region: cn-north-1
          app_id: '3333'
          path: app/build/outputs/apk/app-debug.apk
          output_path: ./instrumented.apk
```

Parameter	Type	Description	Mandatory	Default Value
region	string	Region name.	Yes	None

Parameter	Type	Description	Mandatory	Default Value
app_id	string	App ID.	Yes	None
path	string	APK location before injection.	Yes	None
path	string	APK location after injection.	Yes	None

6.3.1.26 Using YAML to Build an Android Quick App

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - quick_app:
        inputs:
          command: |-
            npm config set registry http://7.223.219.40/npm/
             # Load dependencies.
            npm install --verbose
            # Build an app with the default settings.
            npm run build
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.27 Using YAML to Configure a Bazel Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - bazel:
        inputs:
          command: |
              cd java-maven
              bazel build //:java-maven_deploy.jar
              mkdir build_out
              cp -r  bazel-bin/*  build_out/
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.28 Using YAML to Configure a Build with Grails

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - grails:
        inputs:
          command: grails war
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.29 Using YAML to Build an Android App with Ionic

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - ionic_android_app:
        inputs:
          gradle: '4.8'
          jdk: '3333'
          ndk: '17'
          command: ./instrumented.apk
```

Parameter	Type	Description	Mandatory	Default Value
gradle	string	Gradle version.	Yes	None
jdk	string	JDK file name.	Yes	None
ndk	string	NDK file name.	Yes	None
command	string	Execute commands.	Yes	None

6.3.1.30 Using YAML to Configure a Fortran Build

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - fortran:
        inputs:
          command: |-
            gfortran -c -fpic helloworld.f90
            gfortran -shared -o helloworld.so helloworld.o
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.31 Using YAML to Configure a Build with MSBuild

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - msbuild:
        inputs:
          command: |-
            nuget sources Disable -Name 'nuget.org'
            nuget sources Disable -Name 'myget'
            nuget sources Disable -Name 'huawei'
            nuget sources add -Name 'huaweicloud' -Source 'http://mirrors-devcloud.rnd.huawei.com/
repository/nuget/v3/index.json'
            nuget restore
            msbuild /p:OutputPath=../buildResult/Release/bin
            powershell -Command Compress-Archive -Path ./buildResult/Release/bin/* -DestinationPath ./
archive.zip
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Execute commands.	Yes	None

6.3.1.32 Using YAML to Create an Image and Upload It to SWR

Before uploading an image to SWR, learn about the **Notes and Constraints** of SWR.

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - build_image:
        name: buildImage
        inputs:
          regions: ["x-x-x", "x-x-xxx"]
          organization: codeci_test
          image_name: demo
          image_tag: ${GIT_COMMIT}
          dockerfile_path: dockerfile/Dockerfile
          # set_meta_data: true
```

Parameter	Type	Description	Mandatory	Default Value
regions	list	Select the regional SWR to be uploaded. By default, the file is uploaded to SWR in the region where the current task is located.	No	None
organization	string	Upload to the SWR organization.	Yes	None
image_name	string	Image name.	No	demo
image_tag	string	Image tag.	No	v1.1

Parameter	Type	Description	Mandatory	Default Value
context_path	string	Docker context path.	No	.
dockerfile_path	string	Path of the dockerfile relative to context_path.	No	./ Dockerfile
set_meta_data	bool	Whether to add build metadata to the image.	No	false

6.3.1.33 Using YAML to Specify SWR Public Images

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - swr:
        image: cloudbuild@ddd
        inputs:
          command: echo 'hello'
```

Parameter	Type	Description	Mandatory	Default Value
image	string	Image address.	No	None
command	string	Execute commands.	Yes	None

6.3.1.34 Using YAML to Upload Files to OBS

For details about the restrictions on using OBS, see **Restrictions and Limitations**.

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - upload_obs:
          inputs:
            artifact_path: "**/target/*.?ar"
            bucket_name: codecitest-obs
            obs_directory: test
          #   artifact_dest_name: ""
          #   upload_directory: true
          #   headers:
          #     x-frame-options: true
          #     test: test
          #     commit: ${commitId}
```

Parameter	Type	Description	Mandatory	Default Value
artifact_path	string	Path of the product to be uploaded. Regular expressions are supported.	No	bin/*

Parameter	Type	Description	Mandatory	Default Value
bucket_name	string	Specifies the name of the OBS bucket to which the file is uploaded.	Yes	None
obs_directory	string	Path of the OBS folder to be uploaded. By default, the file is uploaded to the root directory of the bucket.	No	./
artifact_dest_name	string	Name of the file uploaded to OBS. Set this parameter when the product needs to be renamed.	No	None
upload_directory	bool	Whether to upload a folder. If this parameter is set to **false**, all matched products are uploaded to **obs_directory** in tile mode.	No	false
headers	map	Uploaded header information	No	None

6.3.1.35 Using YAML to Download Files

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - download_file:
          inputs:
            name: android22.jks
```

Parameter	Type	Description	Mandatory	Default Value
name	string	File name.	Yes	None

6.3.1.36 Using YAML to Upload Binary Packages to a Repository

For details about the restrictions on the uploaded software packages, see **Constraints** of CodeArts Artifact.

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - upload_artifact:
          inputs:
            path: "**/target/*.?ar"
            version: 2.1
            name: packageName
```

Parameter	Type	Description	Mandatory	Default Value
path	string	Directory for storing the build result. A regular expression is supported. Example: **/target/*.?ar uploads all JAR and WAR packages built with Maven.	Yes	None
version	string	Directory for storing the build package in a release repo. Leave this blank to use the build ID for the directory (recommended).	No	None
name	string	New name for files uploaded to the release repo. Leave this parameter blank to use the original file names (recommended).	No	None

6.3.1.37 Using YAML to Download Binary Packages

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - download_artifact:
        inputs:
          url: xxxxxxxxxxxxx
```

Parameter	Type	Description	Mandatory	Default Value
url	string	Download URL (the download address of the binary package in release repos).	Yes	None

6.3.1.38 Using YAML to Run Docker Commands

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - docker:
        inputs:
          command: |
            docker pull swr.xx-xxxxx-x.myxxcloud.com/codeci/dockerindocker:dockerindocker18.09-1.3.2
```

Parameter	Type	Description	Mandatory	Default Value
command	string	Each command takes up one line. Supported docker commands: **build**, **tag**, **push**, **pull**, **login**, **logout**, and **save**.	Yes	None

6.3.2 Configuring Tasks

Background

A build task is the minimum unit and applies to simple service scenarios. However, build tasks may not meet complex requirements. For example:

- A multi-repository project needs to be built on multiple machines, and the build projects depend on each other.
- You want to split build task in a more modular and fine-grained manner and build them in the dependency sequence.

In the preceding complex build scenarios, BuildFlow can be used to assemble multiple dependent build tasks in directed acyclic graph (DAG) mode. BuildFlow will concurrently build tasks based on the dependencies.

BuildFlow Overview

The following is a BuildFlow example.

```
version: 2.0 # The value must be 2.0.
params:
  - name: buildFlowParam
    value: buildFlowValue
buildflow:
 strategy: lazy # Defines the running policy of BuildFlow. The value can be lazy or eager.
jobs: # Build task
   - job: Job3
     depends_on: # Define the job dependency. In the instance, Job3 depends on Job1 and Job2.
         - Job1
         - Job2
     build_ref: .cloudbuild/build3.yml # Define the YAML build script to run during a job build.
   - job: Job1
     build_ref: .cloudbuild/build1.yml
   - job: Job2
     build_ref: .cloudbuild/build2.yml
```

The BuildFlow contains the following key elements:

- **version**: version number, which is mandatory and unique. In the example file, the value of version is 2.0.
- **params**: global build parameters of BuildFlow. This parameter is shared by all jobs.
- **strategy**: running policy. There are two running modes. If there is no explicit definition, the **Eager** mode is used by default.
  - **Lazy**: The build of a sub-job with a higher priority is triggered first. After the sub-job with a higher priority is successfully executed, the build of a sub-job with a lower priority is then triggered.

    NOTE

    The build takes a long time but saves build resources. Therefore, you are advised to use this method when the number of parallel jobs is insufficient.
  - **Eager**: Trigger the build of all sub-jobs synchronously. For sub-jobs that depend on other jobs, prepare the environment and code first and wait until the dependent jobs are successfully built.

NOTE

Resources may be idle, but the build time can be shortened. You are advised to use this function when the number of concurrent requests is large enough.

- **Jobs**: jobs to be orchestrated. In the example file, there are three parameters under **Jobs**.
  - **job**: build task name, which can be customized.
  - **depends_on**: build task on which the build job depends.
  - **build_ref**: YAML build script that needs to be run during building of the job.

  In this example, three build jobs Job1, Job2, and Job3 are configured. The build jobs share the defined parameters **params**, and Job3 depends on Job1 and Job2.

Introduction to BuildFlow Jobs

BuildFlow jobs are used to define jobs to be orchestrated in BuildFlow. Each job must have a unique name as the unique identifier.

NOTE

- If sub-job A depends on sub-job B, B has a higher priority.
- Sub-jobs with the same priority are triggered synchronously.

BuildFlow jobs example:

```
buildflow:
  strategy: lazy
  jobs:
   - job: Job3
     depends_on:
         - Job1
         - Job2
     build_ref: .cloudbuild/build3.yml
   - job: Job1
     build_ref: .cloudbuild/build1.yml
   - job: Job2
     build_ref: .cloudbuild/build2.yml
```

As shown in the preceding information, Job3 depends on and has lower priority than Job1 and Job2, which are triggered synchronously.

Introduction to BuildFlow params

BuildFlow params can define global parameters, that is, shared by all jobs. However, in some cases, the granularity of global parameters may be too large. You only need to define parameters on some jobs. You can also define parameters only for some jobs. Here is an example.

```
buildflow:
 jobs:
   - job: Job3
     depends_on:
         - Build Job1
         - Build job2
     build_ref: .cloudbuild/build3.yml
   - job: Job1
     params:
       - name: isSubmodule
```

```
      value: true
  build_ref: .cloudbuild/build1.yml
- job: Job2
  params:
    - name: isSubmodule
      value: true
  build_ref: .cloudbuild/build2.yml
```

As shown in the preceding information, the global parameter **params** is not defined in BuildFlow. Instead, the **isSubmodule** parameter is defined in Job1 and Job2.

NOTE

During **Build with YAML**, pay attention to the parameter priority.

Runtime parameters > Parameters configured in settings of a task > Parameters defined in the YAML file of the BuildFlow sub-jobs > Parameters defined in the job in the YAML file of the BuildFlow parent task > Global parameters defined in the YAML file of the BuildFlow parent task

7 Running a Build Task

Prerequisites

You have **created a build task** and you have permissions to run or disable the build task.

NOTE

- If runtime parameters have been configured for the build task and are referenced, the parameter setting dialog box is displayed. Set the parameters as required.

Running a Task

1. **Log in to the CodeArts Build homepage**.
2. Search for the target build task on CodeArts Build homepage and click ▷ to run the task.

Disabling a Task

1. Search for the target build task.
2. Click ··· in the row that contains build task and choose **Disable** from the drop-down list.
3. In the displayed **Disable Task** dialog box, enter the reason and click **OK**.

   NOTE

   - Running build tasks cannot be disabled or deleted.
   - After the build task is disabled, **Disabled** is displayed next to the build task name.

     To run the build task, click ··· in the row that contains the build task and choose **Enable** from the drop-down list.

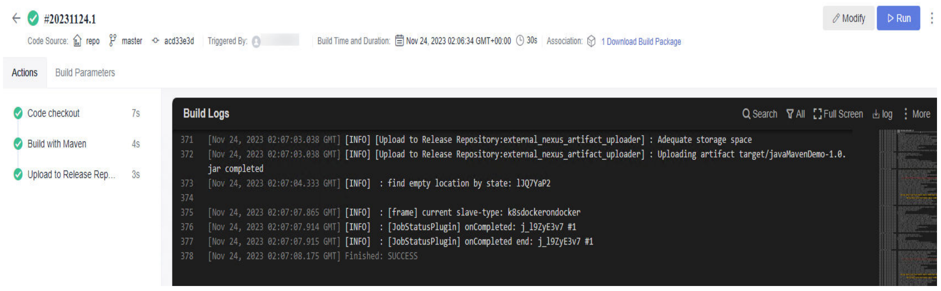
8 Viewing a Build Task

1. **Log in to the CodeArts Build homepage**.
2. The build task list related to the current user is displayed, showing the following information.

Item	Description
Build Tasks	Name of the project to which the build task belongs and the build task name. You can click the project name to go to the build list of the project and click the task name to go to the build history page. If the build is successful, a green icon is displayed. If the build fails, a red icon is displayed. If the build is suspended, a yellow icon is displayed. If the build is not completed, a light gray icon is displayed.
Last Executed	Information such as the task executor, triggering mode, branch of the used repository, and commit ID.
Result	The latest execution results are displayed from right to left. Green indicates that the execution is successful, blue indicates that the execution is in progress, and red indicates that the execution fails.
Build Time and Duration	Build task start time and build duration.
Operation	Click ▷ to start builds, ☆ to favorite tasks, and ⋯ to expand the drop-down list (edit, clone, disable, and delete tasks.) For details, see **Build Task Operations**.

3. Click the build task name to go to the **Build History** page. You can view the latest build history. (The build records in latest 30 days are displayed by default. You can customize the period using the date selection component in the upper left corner of the page.)
4. Click the **Dashboard** tab to view the build success rate and build performance distribution in the last seven days in a pie chart, line chart, or bar chart.

5. Click a build ID on the **Build History** tab to view details, including the code source, trigger source, build time and duration, action logs, build parameters, and association.



Code Source: repo master acd33e3d Triggered By: [redacted] Build Time and Duration: Nov 24, 2023 02:06:34 GMT+00:00 30s Association: 1 Download Build Package

```
[Nov 24, 2023 02:07:03.038 GMT] [INFO] [Upload to Release Repository:external_nexus_artifact_uploader] : Adequate storage space
[Nov 24, 2023 02:07:03.038 GMT] [INFO] [Upload to Release Repository:external_nexus_artifact_uploader] : Uploading artifact target/javaMavenDemo-1.0.
jar completed
[Nov 24, 2023 02:07:04.333 GMT] [INFO]  : find empty location by state: 1JQ7YaP2

[Nov 24, 2023 02:07:07.865 GMT] [INFO]  : [frame] current slave-type: k8sdockerondocker
[Nov 24, 2023 02:07:07.914 GMT] [INFO]  : [JobStatusPlugin] onCompleted: j_19ZyE3v7 #1
[Nov 24, 2023 02:07:07.915 GMT] [INFO]  : [JobStatusPlugin] onCompleted end: j_19ZyE3v7 #1
[Nov 24, 2023 02:07:08.175 GMT] Finished: SUCCESS
```

– Click the code source link in the upper left corner to access the code repository page.
– Click **Download Build Package** to download the package.
– Click an action node (such as **Code checkout**) on the left to view the build logs.
– When viewing logs, click **Full Screen** in the upper right corner of the log window to maximize the log window, click **Exit Full Screen** to exit the maximized log window, click **log** to download all log files, and click an action node on the left to view logs of the corresponding action.
– Click **Modify** or **Run** in the upper right corner to edit or run the build task. Click ⋮ and clone the task, save the task as a template, view the badge status, or disable the task.

9 Managing and Configuring a Build Task

9.1 Editing, Deleting, Copying, and Favoriting a Build Task

Ensure you have the required permissions before you perform operations on build tasks.

Editing a Build Task

1. **Log in to the CodeArts Build homepage**.
2. Search for the target build task.
3. In the row of the target build task, click ⋯ and choose **Edit** from the drop-down list.
   - On the **Basic Information** tab page, configure the task name, code source, source code repository, branch, and task description.
   - On the **Build Actions** tab page, modify build steps and parameters.
   - On the **Parameters** tab page, customize parameters for running the build task.
   - On the **Schedule** tab page, configure continuous integration (the triggering event) and scheduled execution.
   - On the **Change History** tab page, view the change history of the build task.
   - On the **Permissions** tab page, configure permissions for different roles.
   - On the **Notifications** tab page, configure task event type notification information (Build succeeded, Build failed, Task deleted, Task configurations updated, and Task disabled).
4. Edit the information on a tab page, and click **Save**.

Deleting the Build Task

1. Search for the target build task.
2. Click ⋯ in the row of the build task and choose **Delete** from the drop-down list. Exercise caution when performing this operation.

You can view the deleted build task in the **recycle bin**.

Cloning the Build Task

1. Search for the target build task.
2. Click ··· in the row of the build task, and choose **Clone** from the drop-down list.
3. On the page that is displayed, modify the task information and click **Clone**.

NOTE

Cloning a task retains the permission matrix of the original task.

Favoriting the Build Task

1. Search for the target build task.
2. Move the cursor to the row of the build task and click ☆. If the color of the icon changes, the task is successfully followed.
3. (Optional) Click ★ to unfollow the task.

NOTE

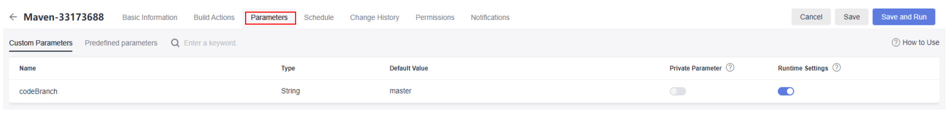
- After you favorite a build task, the task is displayed on the top of the task list when you refresh the page or access the task list next time. If you favorite many build tasks, the tasks are sorted by task creation time in descending order.
- If you favorite a task that is not created by yourself, you can obtain the corresponding notification based on the notification event type set for the task.

9.2 Configuring Build Parameters

By default, the **codeBranch** parameter and predefined parameters are generated for a build task. You can modify the type and value of **codeBranch** and add custom parameters as required. Predefined parameters and values are automatically generated and can be referenced using *${parameter_name}*.

Parameter Settings

1. **Log in to the CodeArts Build homepage**.
2. Search for the target build task.
3. In the row of the target build task, click ··· and choose **Edit** from the drop-down list.
4. Switch to the **Parameters** tab.



Name	Type	Default Value	Private Parameter	Runtime Settings
codeBranch	String	master	○	●

The parameters are described in the following table.

Basic Information	Description
Name	**codeBranch** and predefined parameters are generated by the system and their names cannot be changed. You can change the names of other custom parameters.
Type	Parameter types include string and enumeration.
Default Value	Default values are provided for different types of parameters. You can change the values as required.
Private Parameter	If a parameter is private, the system encrypts the input for storage and only decrypts the parameter when using it. Private parameters are not displayed in run logs.
Runtime Settings	If this function is enabled, parameters can be changed when a build task is executed independently and will be reported to the pipeline. Runtime parameters need to be entered during execution.

– Add a string parameter

  Click **Create Parameter**. A string parameter is added by default. You can edit the parameter as required.

– Add an enumeration parameter

  i. Click **Create Parameter**. A string parameter is added by default.

  ii. Click next to the type and select **Enumeration** from the drop-down list. The **Enumeration** dialog box is displayed.

  iii. Set values for the parameter. Each value must end with a comma (,).

  iv. After the setting is complete, select a value from the drop-down list in the **Default Value** column.

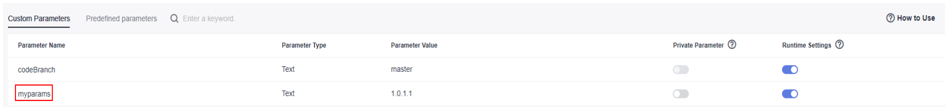
Using Parameters

This section describes how to use custom and predefined parameters.

- **Custom parameters**

  a. Configure an execution parameter.

     Edit the build task, click the **Parameters** tab, add a parameter with a custom name and value (in this example, set the name to **myparam** and value to **1.0.1.1**), and enable **Runtime Settings**.



Parameter Name	Parameter Type	Parameter Value	Private Parameter	Runtime Settings
codeBranch	Text	master	○	●
myparams	Text	1.0.1.1	○	●

  b. Use the execution parameter.

     Switch to the **Build Actions** tab, configure a build action, enter **$ {myparam}** in the **Version** text box, and save the build task.

c. Run the build task.

   In the displayed dialog box for setting parameters and running the task, enter a value or use the default value.

d. Query the build package of this task in CodeArts Artifact. (It is assumed that this build task is built with Maven and CodeArts Artifact is enabled.)

   Go to the release repo and find the build package. The version of this package is the value of **myparam**.

- **Predefined parameters**

  a. Configure an execution parameter.

     Edit the build task, click the **Build Actions** tab, configure a build action, enter **${BUILDNUMBER}** in the **Version** text box, and save the build task.

Parameter	Description
BUILDNUMBER	Build ID in the format of **date.times that this build task run on that day**. For example: **20200312.3**.
GIT_COMMIT	Code commit ID. For example: **b6192120acc67074990127864d3fecaf259b20f5**.
TIMESTAMP	Build running timestamp. For example: **20190219191621**.
INCREASENUM	Total number of times that the task is run. The value starts from 1 and is incremented by 1 each time the task is run.
PROJECT_ID	Project ID.
WORKSPACE	Workspace, which is the root directory of the source code.
GIT_TAG	Code tag name. The tag has a value only when used for build.

  b. Run the build task.

  c. Query the build package of this task in release repos. (It is assumed that this build task is built with Maven and CodeArts Artifact is enabled.)

     Go to the release repo and find the build package. The version of this package is the value of **BUILDNUMBER**.

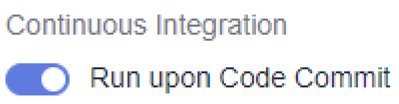
9.3 Configuring Execution Plans

With CodeArts Build, you can configure triggering events and scheduled tasks, so developers can achieve continuous project integration.

Continuous Integration

1. **Log in to the CodeArts Build homepage**.

1. Search for the target build task.
2. Click ··· in the **Operation** column and choose **Edit** from the drop-down list. Click the **Schedule** tab page.
3. Enable **Run upon Code Commit**.



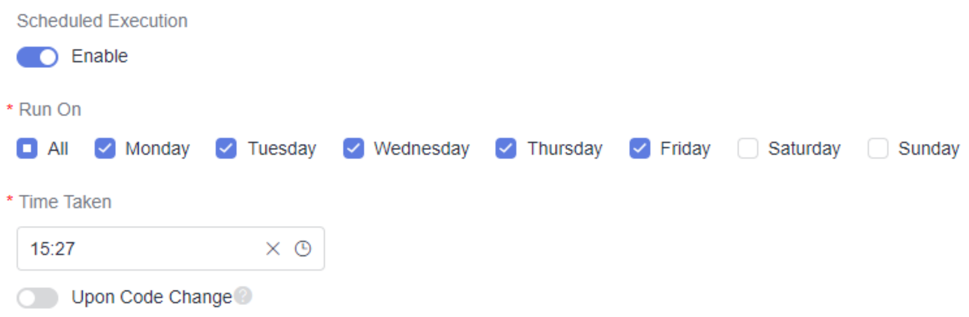
Continuous Integration

Run upon Code Commit

4. After this function is enabled, a build task is triggered when the source code referenced by the build task is committed.

Scheduled Execution

1. Enable **Scheduled Execution**.



Scheduled Execution

Enable

* Run On

[✓ All] [✓ Monday] [✓ Tuesday] [✓ Wednesday] [✓ Thursday] [✓ Friday] Saturday Sunday

* Time Taken

15:27

Upon Code Change

2. Select a scheduled execution time of the build task. Enable **Upon Code Change** as required.
3. After this function is enabled, the build task executes tasks based on the scheduled execution date and time.
4. If you enable both **Scheduled Execution** and **Upon Code Change**, the build task is executed only when the specified execution date and time are reached and the code changes compared with the last build.

9.4 Configuring Role Permissions

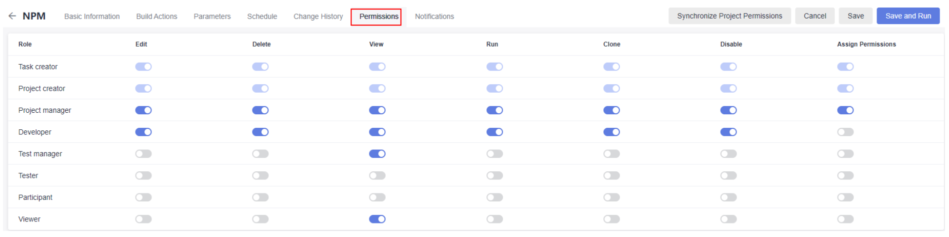
CodeArts Build allows you to configure permissions for each role of a build task.

Procedure

Step 1 Log in to the CodeArts Build homepage.

Step 2 Search for the target build task.

Step 3 In the row of the target build task, click ··· and choose **Edit** from the drop-down list. Then click the **Permissions** tab.



Role	Edit	Delete	View	Run	Clone	Disable	Assign Permissions
Task creator	●	●	●	●	●	●	●
Project creator	●	●	●	●	●	●	●
Project manager	●	●	●	●	●	●	●
Developer	●	●	●	●	●	●	○
Test manager	○	○	●	○	○	○	○
Tester	○	○	○	○	○	○	○
Participant	○	○	○	○	○	○	○
Viewer	○	○	●	○	○	○	○

You can configure permissions for different roles as required.

Click **Synchronize Project Permissions** to synchronize the current build task permissions as the project permissions. For details about how to configure project permissions, see **Setting Project Permissions**.

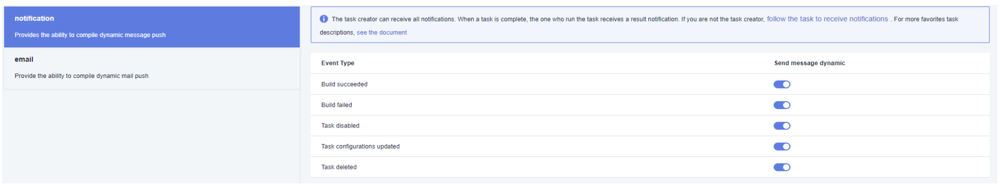
----End

9.5 Configuring Event Notifications

CodeArts Build can notify you of build successes or failures, task disabling, task configuration updates, and task deletion by message or email.

Message/Email Notifications

1. **Log in to the CodeArts Build homepage**.
1. Search for the target build task.
2. Click ··· in the **Operation** column and choose **Edit** from the drop-down list. The **Build Actions** tab page is displayed.
3. Click the **Notifications** tab and configure **Notification** and **Email** separately.
   By default, message notifications are sent for all events and email notifications are sent for build failures. Click to enable notifications or click to disable notifications.



notification
Provides the ability to compile dynamic message push

email
Provide the ability to compile dynamic mail push

The task creator can receive all notifications. When a task is complete, the one who run the task receives a result notification. If you are not the task creator, follow the task to receive notifications . For more favorites task descriptions, see the document

Event Type	Send message dynamic
Build succeeded	●
Build failed	●
Task disabled	●
Task configurations updated	●
Task deleted	●

10 Other Operations

10.1 Configuring Code Source

10.1.1 Introduction

By default, code is pulled from CodeArts Repo for build. Service endpoints are extensions or plug-ins of CodeArts and provide the capability of connecting to third-party services.

CodeArts Build uses service endpoints to connect to Git repositories to obtain project code. You can create, edit, and delete such connections.

NOTE

- The network may be unstable or other problems may occur when a third-party repository is used.
- Use the code import function of CodeArts Repo for secure, stable, and efficient download and build.

10.1.2 Using Git for Build

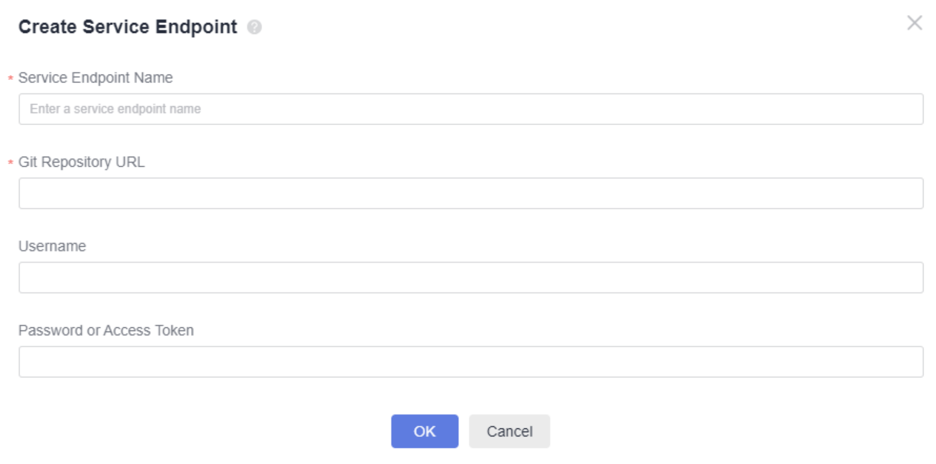
- By default, CodeArts Build pulls code from CodeArts Repo. For code hosted on other services, you can use a Git connection to pull the code.
- Git connections are authorized using AccessToken and are used only to pull code during build.

Procedure

Step 1 Create a build task and select **Git** for **Code Source**. If you use the Git connection for the first time, create an endpoint instance first.

Step 2 Click **Create** next to **Endpoint**.

Step 3 In the **Create Service Endpoint** dialog box displayed, configure parameters.


Create Service Endpoint

* Service Endpoint Name
Enter a service endpoint name

* Git Repository URL

Username

Password or Access Token

Table 10-1 Parameters

Parameter	Description
Service Endpoint Name	Name of the service endpoint.
Git Repository URL	URL of the Git repository (HTTPS address).
Username	Username used for logging in to the Git repository.
Password or Access Token	Git repository password or access token.

Step 4 Click **OK**.

----End

10.2 Operations Recorded by CTS

With CTS, you can record operations associated with CodeArts Build for future query, audit, and backtrack operations.

After you enable CTS, it starts recording operations on build resources.

View traces of the last seven days on the CTS console. For details about viewing logs in CTS, see **Querying Real-Time Traces**.

Table 10-2 Operations that can be recorded by CTS

Operation	Resource Type	Event
Creating a build task	CloudBuildsServer	createJob
Running a build task	CloudBuildServer	buildJob
Deleting a build task	CloudBuildServer	deleteJob

Operation	Resource Type	Event
Updating a build task	CloudBuildServer	updateJob
Disabling a build task	CloudBuildServer	disableJob
Enabling a build task	CloudBuildServer	enableJob
Uploading a keystore file	CloudBuildServer	uploadKeystore
Updating a keystore file	CloudBuildServer	updateKeystore
Deleting a keystore file	CloudBuildServer	deleteKeystore
Initializing the EFS directory and storage quota	CloudBuildCache	initEFSDirAndQuota
Uploading a report (including the unit test and dependency analysis)	CloudBuildReport	uploadReport
Creating a custom template	CloudBuildTemplateSer-vice	createCustomTemplate
Deleting a custom template	CloudBuildTemplateSer-vice	deleteCustomTemplate
Updating nextfs information	nextfsInfo	updateNextfsInfo
Creating nextfs	nextfsInfo	createNextfsInfo
Associating nextfs with a tenant	tenantNextfs	createTenantNextfs
Disassociating a tenant from nextfs	tenantNextfs	deleteTenantNextfs
Modifying License information	licenseInfo	updateLicenseInfo
Creating a tenant license	licenseInfo	createLicenseInfo
Creating code cache information	codeCacheInfo	createCodeCacheInfo
Deleting code cache information	codeCacheInfo	deleteCodeCacheInfo
Creating records of using code cache	cacheHistoryInfo	createCacheHistoryInfo
Updating usage info of code cache	cacheHistoryInfo	updateCacheHistoryInfo

10.3 Recycle Bin

Deleted build tasks are stored in the recycle bin. You can manage these tasks using your account.

Step 1 **Log in to the CodeArts Build homepage**.

Step 2 In the upper right corner of CodeArts Build homepage, click **More** and choose **Recycle Bin** from the drop-down list.

The page displays deleted build tasks and allows the operations listed in the following table.

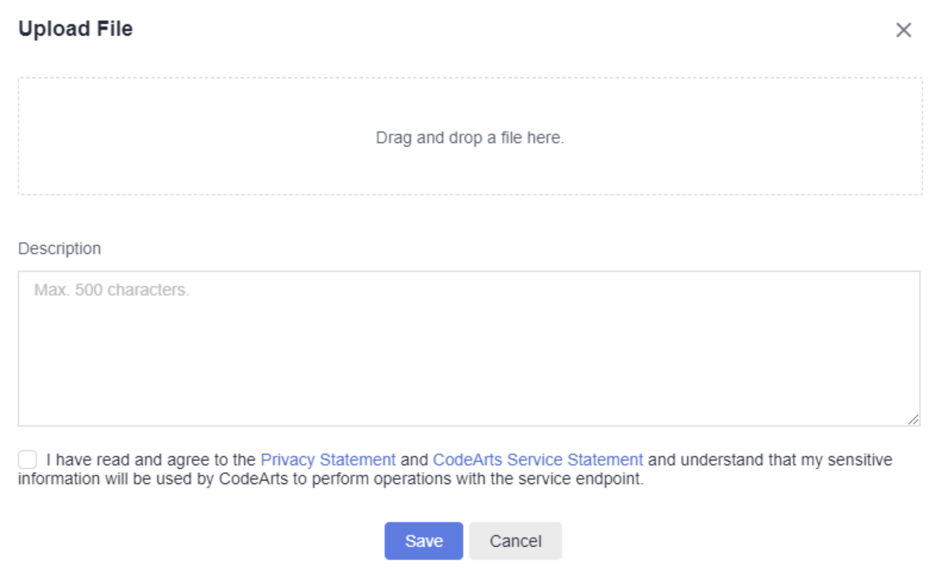
Operation	Description
Modify the task retention period	Click the select box next to **Task Retention Period** and select from 1 to 30 days.
Search for a task	Enter a keyword in the search box and click 🔍 to search.
Delete a task	Select the task to be deleted from the list and click **Delete**.
Restore a task	Select the task to be restored from the list and click **Restore**.
Clear the recycle bin	Click **Empty Recycle Bin** to delete all tasks from the recycle bin.

----End

10.4 File Management

Android APK signature files and **Maven build** configuration file **settings.xml** can be created, edited, deleted, and controlled for access. The maximum file size is 100 KB, and the file type can be **.xml**, **.key**, **.keystore**, **.jks**, **.crt**, or **.pem**.

- Uploading a file
  a. **Log in to the CodeArts Build homepage**.
  b. Click **More** in the upper right corner and choose **Files**.
  c. Click **Upload File**.
  d. In the displayed dialog box, select a file, add a description, agree to the statements, and click **Save**.



Upload File

Drag and drop a file here.

Description

Max. 500 characters.

I have read and agree to the Privacy Statement and CodeArts Service Statement and understand that my sensitive information will be used by CodeArts to perform operations with the service endpoint.

- Managing a file

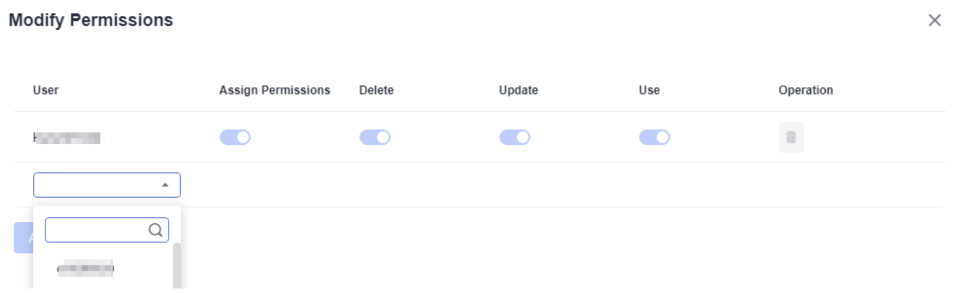
  After uploading a file, you can edit, download, and delete it, and configure file operation permissions for the user.

  – Click in the **Operation** column to modify the file name and specify whether to allow all members of your account to use the file in CodeArts Build.

  – Click in the **Operation** column to download the file.

  – Click in the **Operation** column and confirm the deletion as prompted.

  – Click in the **Operation** column to configure file operation permissions for the user.



Modify Permissions

User	Assign Permissions	Delete	Update	Use	Operation
[redacted]	●	●	●	●	

Table 10-3 File management permissions

Permission	Role with the Permission
Create a file	All users in the project
View a file	File creator and users under the same account
Use a file	File creator and users with the use permissions configured by the file creator
Update a file	File creator and users with the update permissions configured by the file creator

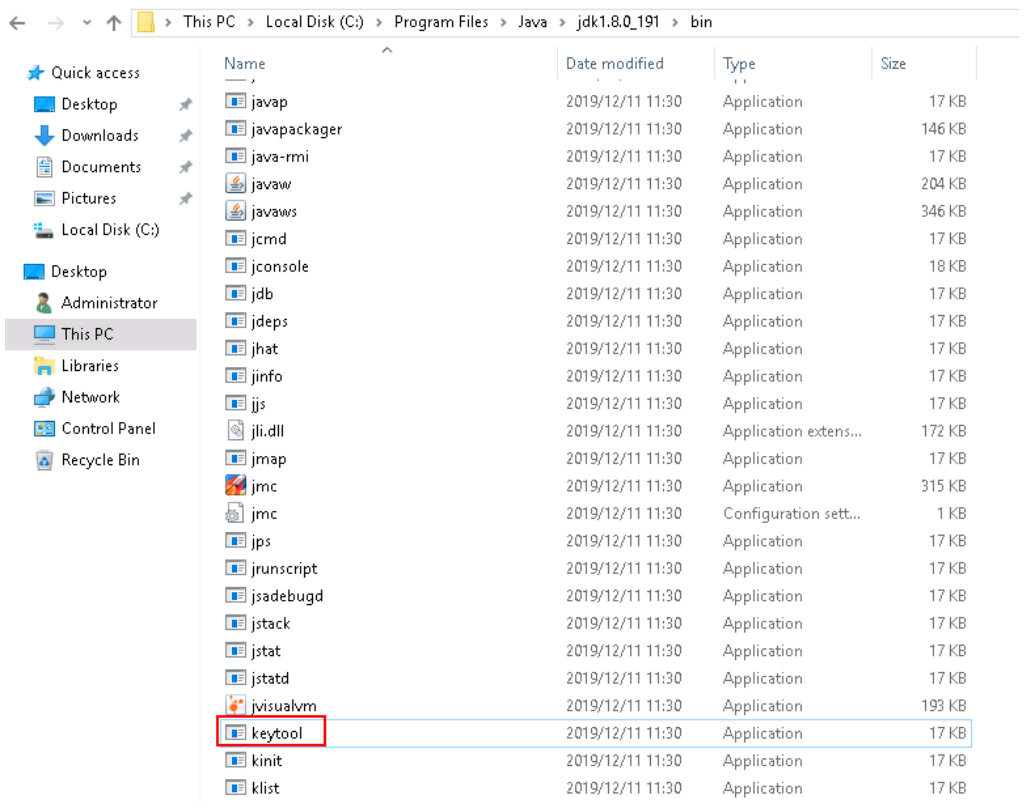
Permission	Role with the Permission
Delete a file	File creator and users with the delete permissions configured by the file creator
Manage permissions	File creator

NOTE

By default, the creator has all permissions, which cannot be deleted or modified.

Generating Keystore Signature Files

- **Using Keytool to Generate Signature Files**
  a. Find the JDK installation path and run **keytool.exe**.

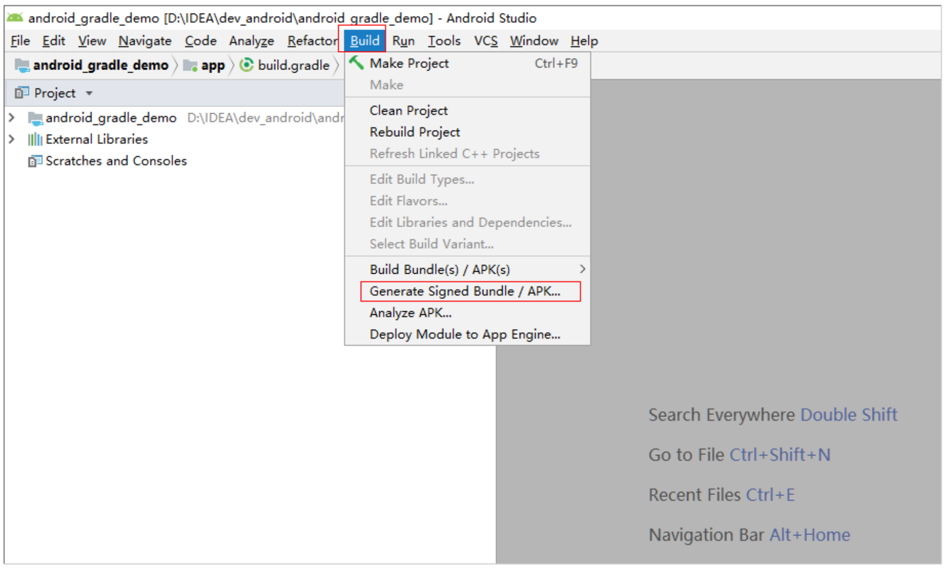


Name	Date modified	Type	Size
javap	2019/12/11 11:30	Application	17 KB
javapackager	2019/12/11 11:30	Application	146 KB
java-rmi	2019/12/11 11:30	Application	17 KB
javaw	2019/12/11 11:30	Application	204 KB
javaws	2019/12/11 11:30	Application	346 KB
jcmd	2019/12/11 11:30	Application	17 KB
jconsole	2019/12/11 11:30	Application	18 KB
jdb	2019/12/11 11:30	Application	17 KB
jdeps	2019/12/11 11:30	Application	17 KB
jhat	2019/12/11 11:30	Application	17 KB
jinfo	2019/12/11 11:30	Application	17 KB
jjs	2019/12/11 11:30	Application	17 KB
jli.dll	2019/12/11 11:30	Application extens...	172 KB
jmap	2019/12/11 11:30	Application	17 KB
jmc	2019/12/11 11:30	Application	315 KB
jmc	2019/12/11 11:30	Configuration sett...	1 KB
jps	2019/12/11 11:30	Application	17 KB
jrunscript	2019/12/11 11:30	Application	17 KB
jsadebugd	2019/12/11 11:30	Application	17 KB
jstack	2019/12/11 11:30	Application	17 KB
jstat	2019/12/11 11:30	Application	17 KB
jstatd	2019/12/11 11:30	Application	17 KB
jvisualvm	2019/12/11 11:30	Application	193 KB
keytool	2019/12/11 11:30	Application	17 KB
kinit	2019/12/11 11:30	Application	17 KB
klist	2019/12/11 11:30	Application	17 KB

  b. Run the following command to generate a .jks file:

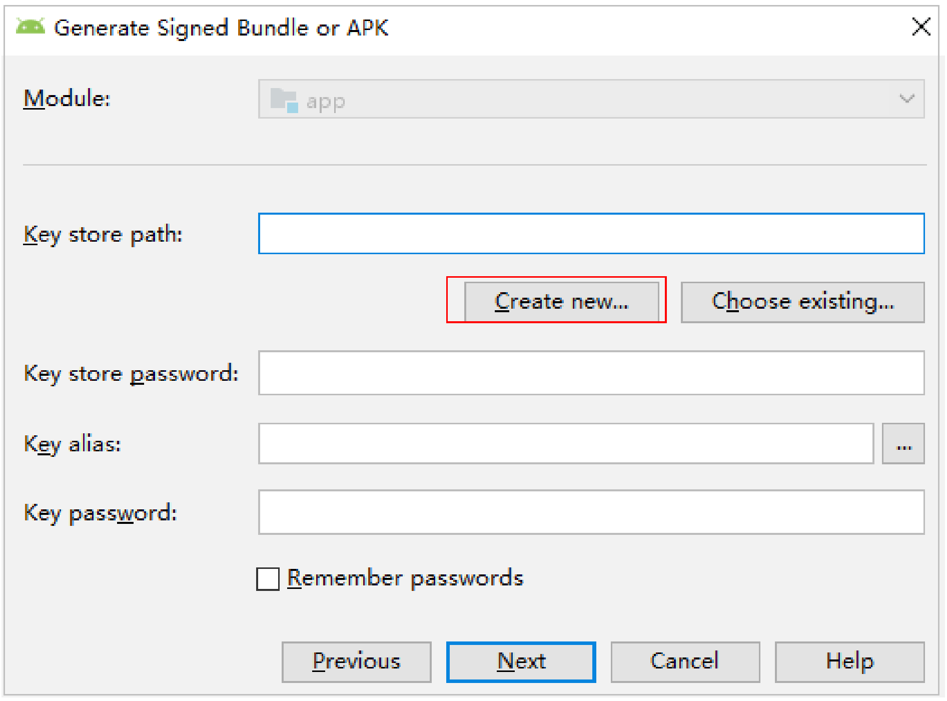
```
keytool -genkeypair -storepass 123456 -alias apksign -keypass 123456 -keyalg RSA -validity
20000 -keystore D:/android.jks
```

- **Using Android Studio to Generate Signature Files**
  a. Open Android Studio and choose **Build** > **Generate Signed Bundle/APK**.

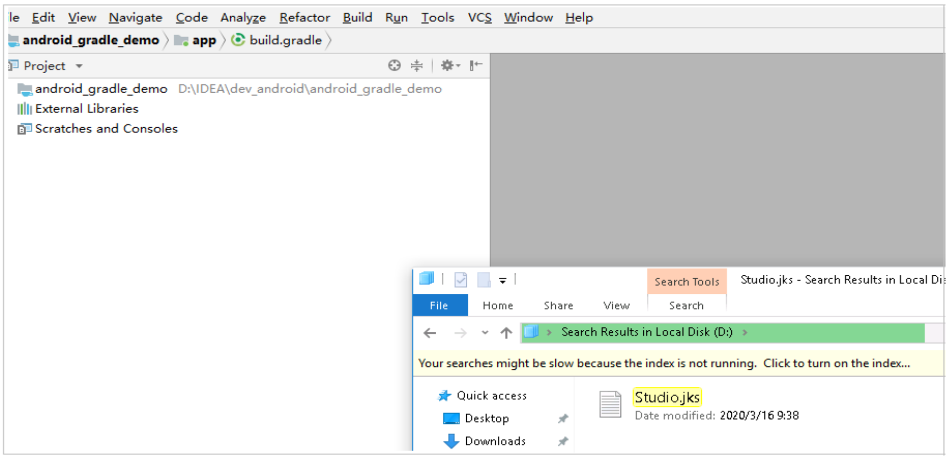

android_gradle_demo [D:\IDEA\dev_android\android_gradle_demo] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
android_gradle_demo app build.gradle
Project
android_gradle_demo D:\IDEA\dev_android\andr
External Libraries
Scratches and Consoles
Make Project Ctrl+F9
Make
Clean Project
Rebuild Project
Refresh Linked C++ Projects
Edit Build Types...
Edit Flavors...
Edit Libraries and Dependencies...
Select Build Variant...
Build Bundle(s) / APK(s)
Generate Signed Bundle / APK...
Analyze APK...
Deploy Module to App Engine...
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home

b. Click **APK** and click **Next**.

c. Click **Create new**. In the dialog box displayed, enter related information, and click **OK**. Then click **Next**.



d. View the signature generated file.


le Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
android_gradle_demo 〉 app 〉 build.gradle 〉
Project
android_gradle_demo D:\IDEA\dev_android\android_gradle_demo
External Libraries
Scratches and Consoles
Search Tools
Studio.jks - Search Results in Local Di
File Home Share View Search
Search Results in Local Disk (D:)
Quick access
Desktop
Downloads
Studio.jks
Date modified: 2020/3/16 9:38

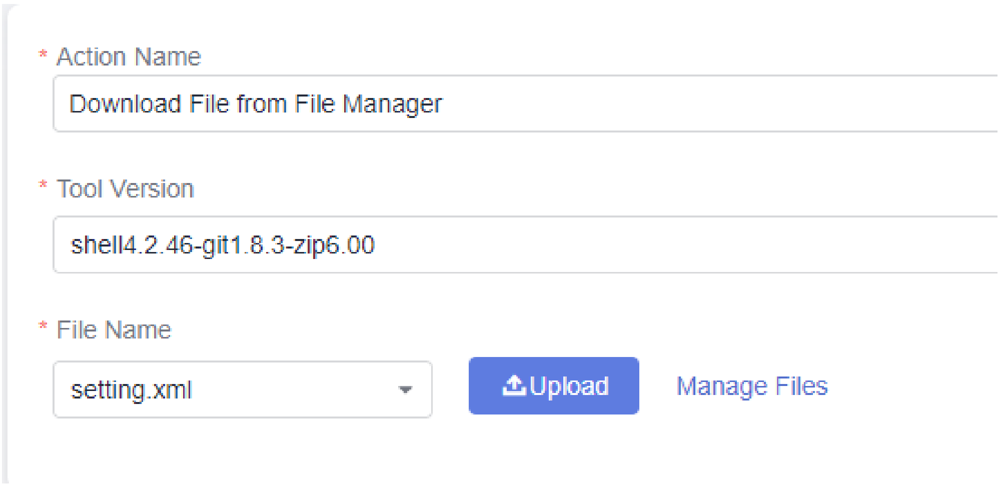
Your searches might be slow because the index is not running. Click to turn on the index...

NOTE

You can upload the generated signature file to **Files** for unified management.

Using the settings.xml File

1. When creating or editing a Maven build task, add the **Download File from File Manager** action on the **Build Actions** tab page, and select the uploaded **settings.xml** file.



* Action Name

Download File from File Manager

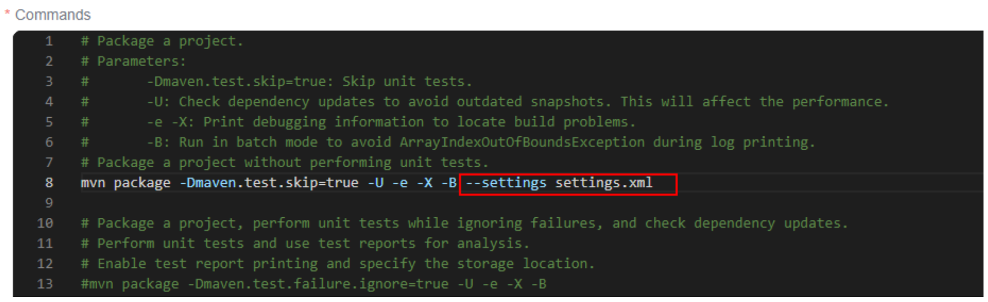
* Tool Version

shell4.2.46-git1.8.3-zip6.00

* File Name

setting.xml

2. Add **--settings settings.xml** to the end of the default Maven build command so that you can use the file for build.



* Commands

```
# Package a project.
# Parameters:
#       -Dmaven.test.skip=true: Skip unit tests.
#       -U: Check dependency updates to avoid outdated snapshots. This will affect the performance.
#       -e -X: Print debugging information to locate build problems.
#       -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 --settings settings.xml

# 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
```

10.5 Customizing Templates

Build template selection: If the preset build templates cannot meet build requirements, you can customize a build template.

Step 1 **Log in to the CodeArts Build homepage**.

Step 2 Select a build task from the list and click the task name. The **Build History** tab page is displayed.

NOTE

If no task is available, **create a build task**.

Step 3 Click ⋮ in the upper right corner of the page. Select **Make Template** from the drop-down list.

Step 4 Enter the template name and description, and click **Save**.

Step 5 Click the username in the upper right corner, and select **All Account Settings** from the drop-down list.

Step 6 In the navigation pane, choose **Build** > **Templates**. The saved template is displayed.

You can perform the following operations on saved templates.

Table 10-4 Managing custom templates

Operation	Description
Favorite a template	Click ☆ to add the template to your favorites.
Delete a template	Click 🗑. In the dialog box that is displayed, click **Yes** to delete the template.

----End

10.6 Custom Build Environments

Background

If the common build environments cannot meet your requirements, customize an environment. To do this, add dependencies and tools required by the project to the base image of the custom environment, **build the image into a Docker image and push it to SWR for public use**. Then you can **use the public image through SWR**.

Base Image

CodeArts Build uses CentOS 7 and Ubuntu 18 as the base images, which are provided with multiple common environment tools. You can configure custom environments as required.

The built-in environment tools include:

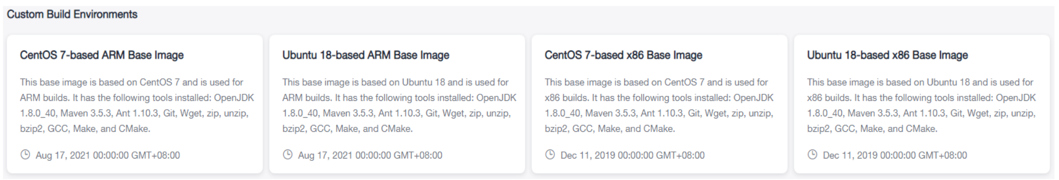
JDK 1.8, Maven, Git, Ant, zip, unzip, GCC, CMake, and Make

Procedure

Step 1 Log in to the CodeArts Build homepage.

Step 2 In the upper right corner of CodeArts Build homepage, click **More** and choose **Custom Build Environments** from the drop-down list.

Step 3 On the **Custom Build Environments** page, click a base image to download the Dockerfile template.



Custom Build Environments

CentOS 7-based ARM Base Image

This base image is based on CentOS 7 and is used for ARM builds. It has the following tools installed: OpenJDK 1.8.0_40, Maven 3.5.3, Ant 1.10.3, Git, Wget, zip, unzip, bzip2, GCC, Make, and CMake.

Aug 17, 2021 00:00:00 GMT+08:00

Ubuntu 18-based ARM Base Image

This base image is based on Ubuntu 18 and is used for ARM builds. It has the following tools installed: OpenJDK 1.8.0_40, Maven 3.5.3, Ant 1.10.3, Git, Wget, zip, unzip, bzip2, GCC, Make, and CMake.

Aug 17, 2021 00:00:00 GMT+08:00

CentOS 7-based x86 Base Image

This base image is based on CentOS 7 and is used for x86 builds. It has the following tools installed: OpenJDK 1.8.0_40, Maven 3.5.3, Ant 1.10.3, Git, Wget, zip, unzip, bzip2, GCC, Make, and CMake.

Dec 11, 2019 00:00:00 GMT+08:00

Ubuntu 18-based x86 Base Image

This base image is based on Ubuntu 18 and is used for x86 builds. It has the following tools installed: OpenJDK 1.8.0_40, Maven 3.5.3, Ant 1.10.3, Git, Wget, zip, unzip, bzip2, GCC, Make, and CMake.

Dec 11, 2019 00:00:00 GMT+08:00

Step 4 Edit the downloaded Dockerfile.

You can add other dependencies and tools required by the project as required to customize the Dockerfile. The following figure shows an example of adding JDK and Maven tools.

```
RUN yum install -y java-1.8.0-openjdk.x86_64
RUN yum install -y maven
RUN echo 'hello world!'
RUN yum clean all
```

----End