CodeArts Build

User Guide

Issue	01
Date	2025-06-09

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

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 Working with CodeArts Build

Build refers to the process of compiling source code into one or more target files, and packaging these target files along with 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 higher efficiency. With just a few clicks, you can easily create, configure, and run build tasks to automate code retrieval, build, and packaging. CodeArts Build also monitors build status in real time.

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

For more information about CodeArts Build, see **Service Overview**.

Steps

Figure 1-1 Basic process of using CodeArts Build

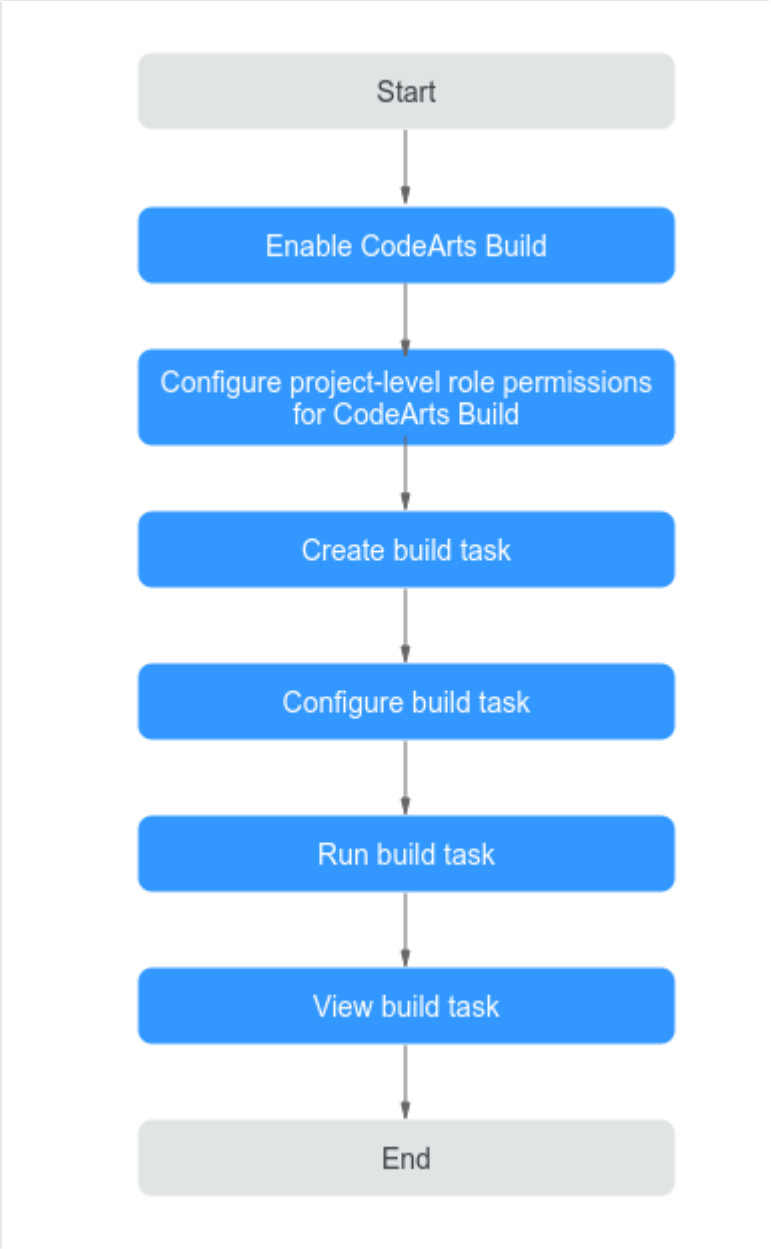


Table 1-1 Steps in using CodeArts Build

Step	Description
Enable CodeArts Build	In this step, you **enable CodeArts Build**.
Configure project-level role permissions for CodeArts Build	Before using CodeArts Build, you need to configure **basic project-level permissions**. Additionally, you can **access the CodeArts Build homepage** to gain an **overall understanding of its features**.

Step	Description
Create build task	In this step, you **create a build task through either the GUI** or **YAML**. You can configure **parameters**, **schedules**, **roles and permissions**, and **event notifications** for the task.
Configure build actions	You can choose from **over 30 build tools** to configure your build by following the GUI guide or referring to the sample code for the YAML file.
Run build task	In this step, you **run a build task**, which can be triggered by pipelines or schedulers.
View build task	In this step, you **check the information and execution results of the build task**.

2 Purchasing CodeArts Build

2.1 Purchasing CodeArts Build

Prerequisites

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

1. Visit **Huawei Cloud official website**.
2. Click **Sign Up** and create your account as instructed.

   Once your account is created, the system automatically redirects you to your personal information page.
3. Complete individual or enterprise real-name authentication. For details, see **Real-Name Authentication**.

Procedure

For details, see **Purchasing CodeArts**.

3 Configuring Project-Level Role Permissions

Assign a specific role to the new member. Each role comes with its own default permissions. For details, see **Table 3-1**.

Table 3-1 Default roles and permissions matrix for CodeArts Build

Role	Create	Edit	Delete	View	Execute	Clone	Disable	Assign Permissions	Group
Project manager	✓	✓	✓	✓	✓	✓	✓	✓	✓
Product manager	✗	✗	✗	✓	✗	✗	✗	✗	✗
Test manager	✗	✗	✗	✓	✗	✗	✗	✗	✗
Operation manager	✗	✗	✗	✗	✗	✗	✗	✗	✗
System engineer	✓	✓	✓	✓	✓	✓	✓	✗	✓
Committer	✓	✓	✓	✓	✓	✓	✓	✗	✗

Role	Create	Edit	Delete	View	Execute	Clone	Disable	Assign Permissions	Group
Developer	✓	✓	✓	✓	✓	✓	✓	✗	✗
Tester	✗	✗	✗	✗	✗	✗	✗	✗	✗
Participant	✗	✗	✗	✗	✗	✗	✗	✗	✗
Viewer	✗	✗	✗	✗	✗	✗	✗	✗	✗
Project admin	✓	✓	✓	✓	✓	✓	✓	✓	✓

NOTE

✓ indicates that the roles have the permission, and ✗ indicates that they do not.

Prerequisites

- You have **enabled CodeArts Build**.
- You have added members by referring to *CodeArts User Guide* > "Preparations" > "Adding Project Members", and assigned roles to the new members by referring to "Managing Permissions".

Accessing the CodeArts Build Homepage

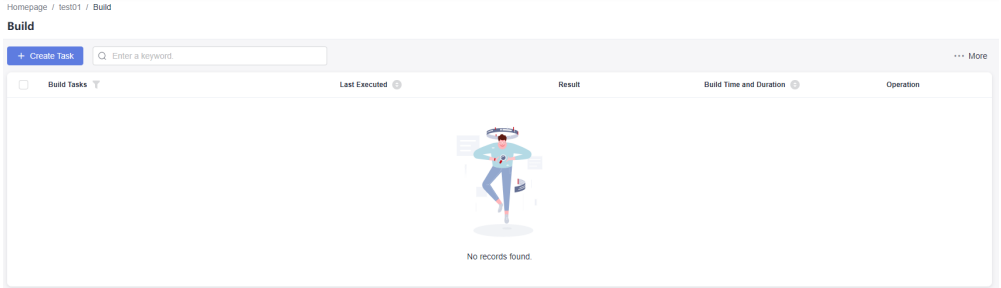
Step 1 **Log in to the Huawei Cloud console** with your Huawei Cloud account.

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

Step 3 You can access CodeArts Build from either the homepage or the project list.

- **From the homepage**

  Click **Access Service** to go to the CodeArts Build homepage, where you can find your build task list.



- **From the project list**

  a. Click **Access Service** to see the CodeArts Build homepage.

  b. Click **Homepage** on the menu bar.

  c. Click the name of the target project.

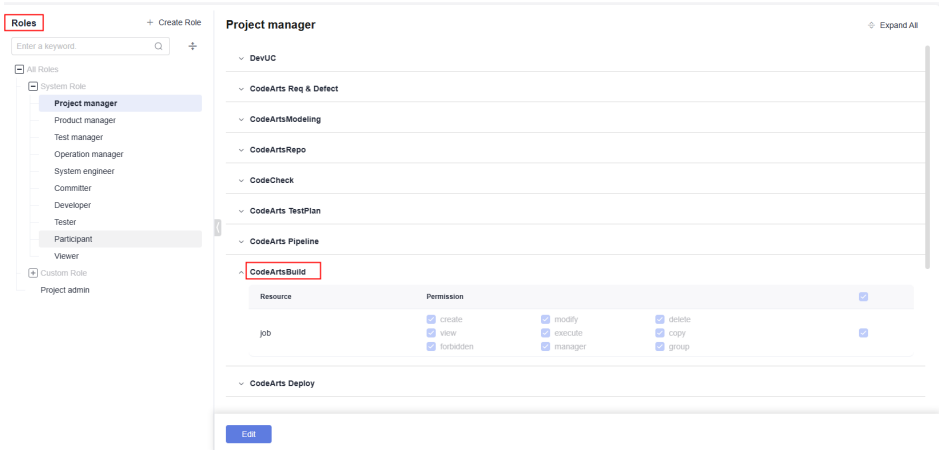
  d. Choose **CICD** > **Build**. The build task list page of the specified project is displayed.

----End

Configuring Role Permissions

1. Access the CodeArts Build Homepage from the project list.
2. In the navigation pane, choose **Settings** > **Permissions**.
3. On the displayed page, configure permissions for different roles on CodeArts Build resources.

Figure 3-1 Configuring project-level role permissions



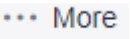
For details about different roles' operation permissions on the specified build task, see **Configuring Roles and Permissions**.

Menu Icons and Their Usage on the CodeArts Build Homepage

CodeArts Build provides multiple choices of UI themes and layouts. This section walks you through the navigation bar that uses the **Infinite** theme and the **Classic** layout.

Table 3-2 Menu icons and their usage

Menu Icon	Description
	Click this icon to expand the drop-down list and select the region you want to switch to. Data and resources are isolated between regions. Use your resources in the region where you purchased it.

Menu Icon	Description
	Click this icon to expand the drop-down list and select **Build** to go to the CodeArts Build homepage, where you can find your build task list.
	Click this project icon to expand the drop-down list and select the project you want to switch to when accessing CodeArts Build from the project page.
	Click this icon to expand the drop-down list and select the desired item to manage **custom templates**, custom build environments, **files**, **recycle bin**, or pools.
	Click this icon to **run the build task**.
	Click this icon to **favorite a build task**.
	Click this icon to expand the drop-down list and select the desired action to **edit**, **clone**, **disable**, or **delete** the build task.

4 Creating a Build Task

4.1 Defining a Build Task on GUI

CodeArts Build provides a graphical user interface (GUI) where you can configure build tools and parameters.

Preparations

- To use CodeArts Repo repositories, you must have the operation permissions on CodeArts Repo.
- Create a project by referring to *CodeArts User Guide* > "Preparations" > "Creating a Project".

  If you already have a project available, skip this step.
- Create a repository by referring to *CodeArts Repo User Guide* > "Creating a CodeArts Repo Repository".

  If you already have a CodeArts Repo repository or are using a third-party repository, skip this step.

Creating a Build Task with GUI

1. **Access the CodeArts Build Homepage** from the project list.
2. Click **Create Task**. On the displayed page, configure the basic information of the build task by referring to **Table 4-1**. Click **Next**. The page for selecting a build template is displayed.

Table 4-1 Basic information

Parameter	Description
Name	Assign a custom name to the build task. • Letters, digits, underscores (_), and hyphens (-) allowed. • 1 to 115 characters.

Parameter	Description
Project	Select the project to which the build task belongs. • This parameter is autofilled when you access CodeArts Build through the project list, so you can leave it as default. • When accessing the service through the service homepage, select the project created in **preparations**.
Code Source	Select the source from where the code to be compiled is pulled. • **Repo**: Code is pulled from **CodeArts Repo** for your build. • **Other Repo**: Code is pulled from CodeArts Repo repositories of other projects for your build. Select a project, and then select a code repository and a default branch. • **Pipeline**: If **Pipeline** is selected as a code source, the build tasks can be executed only by running the corresponding pipeline and cannot be executed independently. The following code repositories are provided by third-party sources and not by CodeArts. • **Git**: Code is pulled from other services for your build.
Service Endpoint	Optional. You need to set this parameter when the **Code Source** is set to a third-party code repository. If you are using a third-party code repository for the first time, you will need to create a service endpoint. For details, see **Creating a Service Endpoint**.
Repository	Select the repository from where the code to be compiled is pulled.
Default Branch	Select a default branch.
Description	Optional. Enter additional information to describe the build task. Max. 512 characters.

3. CodeArts Build has more than 30 built-in build templates. You can select a template that suits your requirements and click **OK** to create the build task. For details about templates, see **Table 4-2**.
   - You can also select **Blank Template** and add desired build actions when **configuring a build task**.
   - If preset templates do not meet your needs, you can also **customize a template**.

Table 4-2 System template description

Template	Description	Language	Stage
Maven	Build a Java project with Apache Maven.	Java	Configure the build environment, download code, build with Maven, and upload the software package to a release repo.
Quick App	Build a quick app with npm.	JavaScript	Configure the build environment, download code, build an Android quick app, and upload the software package to a release repo.
Django	Build a Django project.	Python	Configure the build environment, download code, build with Setuptools, and upload the software package to a release repo.
Ionic Android App	Develop an Android app with Ionic, an HTML5 mobile app development framework.	Java	Configure the build environment, download code, build an Android app with Ionic, and upload the software package to a release repo.
Ant	Apache Ant is a tool used to compile and deploy Java projects.	Java	Configure the build environment, download code, build with Ant, and upload the software package to a release repo.
CMake	Build a cross-platform project with CMake.	C/C++	Configure the build environment, download code, build with CMake, and upload the software package to a release repo.
Grails	Build a Web application with Grails, a Web application development framework.	Groovy	Configure the build environment, download code, build with Grails, and upload the software package to a release repo.

Template	Description	Language	Stage
Grunt	Build a JavaScript project and run the project with Grunt.	JavaScript	Configure the build environment, download code, build with Grunt, and upload the software package to a release repo.
gulp	Use gulp to build an automation workflow and a frontend IDE.	JavaScript	Configure the build environment, download code, build with gulp, and upload the software package to a release repo.
Go	Build a Go project.	Go	Configure the build environment, download code, build with Go, and upload the software package to a release repo.
mono-linux	Compile a project with MSBuild and .NET on Mono Linux (x86 and Arm).	JavaScript	Configure the build environment, download code, build on Mono Linux, and upload the software package to a release repo.
Maven-Container	Use Apache Maven to build a Java project, create an image from Dockerfile, and push the image to SWR.	Java	Configure the build environment, download code, Build with Maven, create an image, and push the image to SWR.
ServiceStage-Maven-Image Build	Use Apache Maven to build a Java project, create an image from Dockerfile, and push the image to SWR.	Java	Configure the build environment, download code, Build with Maven, create an image, push the image to SWR, run shell commands, and upload the software package to a release repo.

Template	Description	Language	Stage
servicestage-npm-war	Build a WAR package with npm and upload the package to a release repo.	JavaScript	Configure the build environment, download code, Build with npm, run shell commands, Build with Ant, and upload the software package to a release repo.
Release Maven Private Package	To build a Java project with Maven using private dependency packages, you must first upload these packages to a self-hosted Maven repo in CodeArts Release.	Java	Configure the build environment, download code, and Build with Maven.
PHP	Declare, install, and package the PHP libraries your project depends on by using the PHP runtime environment and Composer.	PHP	Configure the build environment, download code, build in PHP, and upload the software package to a release repo.
Bazel	Build a project with Bazel.	Java	Configure the build environment, download code, build with Bazel, and upload the software package to a release repo.
Shell	Run shell commands.	N/A	Configure the build environment, download code, and run shell commands.
npm	Build Vue and Webpack projects with npm.	JavaScript	Configure the build environment, download code, build with npm, and upload the software package to a release repo.
Android APK	The Android build system compiles application resources and source code, and then packages them into APKs that can be deployed, signed, and distributed.	Java	Configure the build environment, download code, build with Android, and upload the software package to a release repo.

Template	Description	Language	Stage
Gradle	Build a Java, Groovy, or Scala project with Gradle.	Java	Configure the build environment, download code, build with Gradle, and upload the software package to a release repo.
serverless-npm	Build Vue and Webpack projects with npm.	JavaScript	Configure the build environment, download code, build with npm, and upload the software package to a release repo.
DevStar-Maven-Container Build	Use Apache Maven to build a Java project, create an image from Dockerfile, and push the image to SWR.	Java	Configure the build environment, download code, Build with Maven, create an image, and push the image to SWR.
DevStar-npm-Container Build	Use npm to build a JavaScript project, create an image from Dockerfile, and push the image to SWR.	JavaScript	Configure the build environment, download code, Build with npm, create an image, and push the image to SWR.
DevStar-Python-Container Build	Build a Python project, create an image from Dockerfile, and push the image to SWR.	Python	Configure the build environment, download code, Build with Setuptools, create an image, and push the image to SWR.
Yarn	Build a JavaScript project with Yarn.	JavaScript	Configure the build environment, download code, build with Yarn, and upload the software package to a release repo.
buildImage	Build an image with a Dockerfile and push the image to SWR.	Java	Configure the build environment, download code, create an image, and push the image to SWR.

Template	Description	Language	Stage
PyInstaller	Use the Pyinstaller tool to compress the Python file into an executable program.	Python	Configure the build environment, download code, build with Pyinstaller, and upload the software package to a release repo.
Setuptools	Build a Python project with Setuptools.	Python	Configure the build environment, download code, build with Setuptools, and upload the software package to a release repo.
GNU-ARM	Design, develop, and use an Arm simulator with the GNU Arm toolchain.	C	Configure the build environment, download code, build with GNU Arm, and upload the software package to a release repo.

4. On the displayed **Build Actions** page, click **Save**. The build task is created.

   You can continue to configure this build task by referring to **Configuring a Build Task**.

Turning a Task Into a Template

You can save the current build task as a template for later build task creation. The procedure is as follows:

Step 1 On the **Build History** page, click ··· in the upper right corner and select **Make Template** from the drop-down list.

Step 2 Enter the template name and description, and click **Save**.

Step 3 Click the username, and select **All Account Settings** from the drop-down list.

Step 4 In the navigation pane, choose **Build** > **Templates**. The saved template is displayed in the list.

You can perform the following operations on saved templates.

Table 4-3 Managing custom templates

Operation	Description
Search for a template	Enter a keyword in the search box to search for a template.
Favorite a template	Click ☆ to add the template to your favorites.

Operation	Description
Delete a template	Click . In the displayed dialog box, click **Yes** to delete the template. You can only delete templates that you have created yourselves.

----End

Creating a Service Endpoint

When you select any third-party repository on the **Basic Information** page, the **Endpoint** is a mandatory setting.

Service endpoints are extensions or plug-ins of CodeArts and provide the capability of connecting to third-party services.

By default, CodeArts Build pulls code from CodeArts Repo for your build. CodeArts Build also uses service endpoints to connect to third-party repositories to obtain project source code.

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.

Git

1. Click **Create** next to **Service Endpoint**.
2. In the displayed dialog box, configure the following parameters.

Parameter	Description
Service Endpoint Name	Assign a custom name to the service endpoint. Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
Git Repository URL	Enter the HTTPS address of the target Git repository.
Username	Optional. Enter the username of the target Git repository. Max. 300 characters.
Password or Access Token	Optional. Enter the password of the target Git repository. Max. 300 characters.

3. Click **Confirm**.

4.2 Defining a Build Task with Code

4.2.1 Creating a Build Task with Code

CodeArts Build allows you to define your build as code using YAML. Your configurations, such as build environments, parameters, commands, and actions, reside in a YAML file named **build.yml**. After creating this file, add it along with the source code to a code repository. The file will be used as a script by the system to run a build.

Defining your build as code has the following advantages:

- Your YAML file collects and clearly describes build parameters, commands, steps, as well as post-build operations, ensuring a trusted build process.
- The build configurations in **build.yml** are versioned alongside the commits in the code repository. This enables you to rerun earlier build tasks despite configuration changes.
- To modify the build script for a new feature, create a branch to modify the **build.yml** file for testing. By this way, you will not have to worry about affecting other branches.

Constraints

You can only use the code hosted in CodeArts Repo for code-based builds.

Preparations

- Obtain permissions for CodeArts Repo.
- Create a repository by referring to *CodeArts Repo User Guide* > "Creating a CodeArts Repo Repository".
- Create a project by referring to *CodeArts User Guide* > "Preparations" > "Creating a Project".

  If you already have a project available, skip this step.

Creating a YAML File for Your Code-based Build

1. **Access the CodeArts Build Homepage** from the project list.
2. In the navigation pane, choose **Code** > **Repo**.
3. On the CodeArts Repo console, click **New Repository**. On the displayed page, select **Common** and click **Next**. Then set parameters according to **Table 4-4**, and click **OK**.

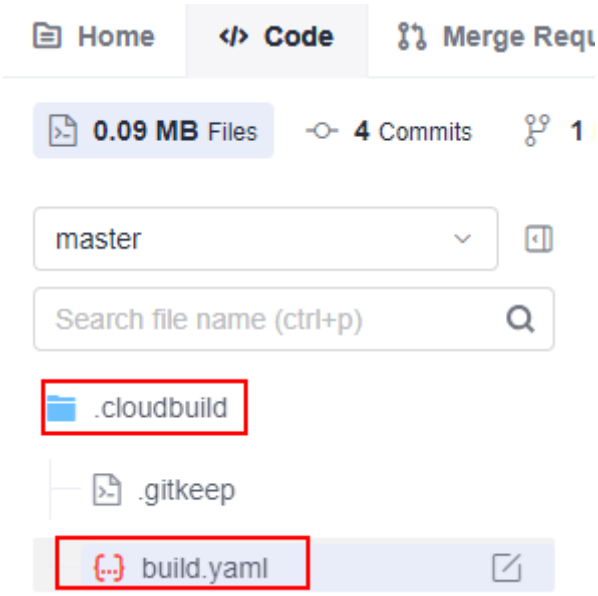
Table 4-4 Creating a code repository

Parameter	Description
Repository Name	Assign a custom name to the code repository, for example, **maven_yml_build**. • The name starts with a digit, letter, or underscore (_). • The name can contain periods (.) and hyphens (-). • The name cannot end with **.git**, **.atom**, or a period (.).

Parameter	Description
Description	Optional. Enter additional information to describe the code repository.
.gitignore Programming Language	Optional. Select the appropriate programming language, such as **Java**, for the **.gitignore** file.
Initial Settings	Select all. • **Generate README**: Select this option to create a README file where you can add details about the project's architecture and purpose, similar to a repository-wide comment. • **Automatically create check task (free of charge)**: Select this option to auto-generate a code check task for the repository upon creation. The check task will appear in the check task list.
Visibility	Select **Private**. • **Private**: Only repository members can access and commit code. • **Public**: The repository is open and read-only to all guests, but is not displayed in their repository list or search results. You can select an open-source license as the remarks.

4. Choose **Create** > **Create Directory** to create a directory named **.cloudbuild**.
5. In the **.cloudbuild** directory, choose **Create** > **Create File** to create a file named **build.yml**. **Figure 4-1** shows the directory that stores files of the code repository.

Figure 4-1 Directory



If the YAML file is not stored in the **.cloudbuild** directory, you can use parameter **CB_BUILD_YAML_PATH** to specify the path of the YAML file in the code repository. For details about parameter settings, see **Adding Custom Parameters**.

6. Click and write the **build.yml** file by referring to the sample code in **Creating the build.xml File for a Single Task**.

Configuring Basic Information

1. In the navigation pane, choose **CICD** > **Build**.
2. Click **Create Task**. On the displayed page, configure the basic information of the build task by referring to **Table 4-5**. Click **Next**. The page for selecting a build template is displayed.

Table 4-5 Basic information

Parameter	Description
Name	Assign a custom name to the build task. • Letters, digits, underscores (_), and hyphens (-) allowed. • 1 to 115 characters.
Project	Select the project to which the build task belongs. • This parameter is autofilled when you access CodeArts Build through the project list, so you can leave it as default. • When accessing the service through the service homepage, select the project created in **preparations**.

Parameter	Description
Code Source	If you select **Repo**, code is pulled from CodeArts Repo for your build.
Repository	Select the repository from where the code to be compiled is pulled.
Default Branch	Select a default branch.
Description	Optional. Enter additional information to describe the build task. Max. 512 characters.

Selecting a Build Template

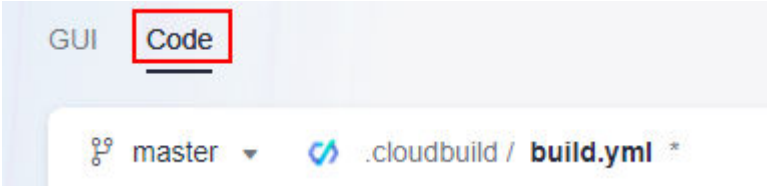
On the page for selecting a build template, select **Blank Template** and click **OK**. The **Build Actions** page is displayed.

NOTE

No matter which build template you select, code-based builds remain unaffected.

Configuring Build Actions

In the upper left corner of the **Build Actions** page, click the **Code** tab. The system automatically reads the YAML file from the code repository and the branch **configured in basic information**.



You can modify the YAML file by referring to the sample code in the "Build as Code" part of **build task configurations**. Any changes made to the YAML file will overwrite the original **YAML file you create for your code-based build** once the build task is completed.

Complete all the configurations and click **Save** to create a build task.

4.2.2 Creating the build.xml File for a Single Task

CodeArts Build allows you to define your build as code using YAML. Your configurations, such as build environments, parameters, commands, and actions, reside in a YAML file named **build.yml**. After creating this file, add it along with the source code to a code repository. The file will be used as a script by the system to run a build.

The following example uses a built-in x86 executor with 8 vCPUs and 16 GB memory. The tasks involve compiling and building code from CodeArts Repo using Maven, and then uploading the resulting software package to a release repo.

For details about the sample code of different build actions, see the "Build as Code" part of each build action in **Configuring Build Actions**. For details about the YAML file structure of multiple tasks, see **Creating the build.xml File for Multiple Tasks**.

```
version: 2.0 # The version number is a mandatory and unique parameter that must be set to 2.0.
params: # Build parameters that can be referenced during your build. If no build parameters are set here, parameters created during task configuration are preferentially used.
  - name: paramA
    value: valueA
  - name: paramB
    value: valueB
env: # Optional. It defines the build environment. x86 is used by default if neither env or envs is set.
  resource:
    type: docker # Agent pool type. The value can be docker or custom. docker indicates that the default executor is used, and custom indicates that a custom executor is used.
    arch: X86 # The host type of the build environment can be either 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: # It prepares for the build process by performing tasks such as downloading code and running shell commands.
    - checkout:
        name: Code download # Optional
        inputs: # Action parameters
          scm: codehub # Code source: CodeArts Repo only
          url: xxxxxxxxx # This refers to the SSH address of the repository that the code is pulled from.
          branch: ${codeBranch} # Pulled code branch, which can be parameterized.
    - sh:
        inputs:
          command: echo ${paramA}
  BUILD: # It defines the build action. Only one BUILD can be configured.
    - maven: # Action keyword. Only specified keywords are supported.
    name: maven build # Optional
    image: xxx # Image address.
    inputs:
      command: mvn clean package
    - upload_artifact:
    inputs:
          path: "**/target/*.?ar"
```

4.2.3 Creating the build.xml File for Multiple Tasks

A build task is the smallest unit that a build project can be broken down into for simple service scenarios. However, for more complex requirements, you may need to:

- Use a multi-repo approach to distribute build tasks that depend on each other across multiple machines.
- Set up multiple build tasks in a modular and fine-grained way, and run them in a specific order. Each task depends on the successful completion of its dependency task.

To handle such complex builds, CodeArts Build offers a task model called BuildFlow, which organizes multiple build tasks in a directed acyclic graph (DAG) and runs them in parallel based on their dependencies.

The following is a YAML file example for multiple tasks:

```
version: 2.0 # The version number is a mandatory and unique parameter that must be set to 2.0.
params: # Parameters that can be referenced by builds.
  - name: p
```



```
    value: 1
# env and envs are optional. Use envs if you need to specify conditions to determine the host specification and type.
env: # This parameter takes precedence once being set. The host specification and type defined here will be used instead of the ones set in the build environment configuration.
  resource:
    type:docker # Agent pool type. The value can be docker or custom. docker indicates that the default executor is used, and custom indicates that a custom executor is used.
    arch:X86 # The host type of the build environment can be either 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.
envs:
    - condition: p == 1 # The following host specification and type are used if this condition is met.
      resource:
        type: docker
        arch: ARM
    - condition: p == 0 # The following host specification and type are not used if this condition is not met.
      resource:
        type: docker
        arch: X86
# Configure either buildflow or buildflows. Use buildflows if you need to specify conditions to control job executions.
buildflow:
 strategy: lazy # Define the running policy of buildflow, which can be lazy or eager. The eager mode is used by default if this parameter is not defined.
    jobs: # Build tasks
      - job: Job3 # Assign a custom name to the child task.
        depends_on: # Define the task dependency. In this practice, the configuration indicates that 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
buildflows:
 - condition: p == 1 # All jobs under this collection are executed if this condition is met.
    jobs: # It defines the tasks to be orchestrated.
      - job: Job1 # Assign a custom name to the child task.
        build_ref: 1.yml # YAML build script that needs to be run during a build.
        params:
          - name: abc
            value: 123
      - condition: p == 1 # Job2 is executed if this condition is met.
        job: Job2
        build_ref: 2.yml
        params:
          - name: abc
            value: 123
```

NOTE

- **lazy**: A job with a higher priority is triggered first. After successful execution, a job with a lower priority is then triggered. The build takes a long time but saves resources. Therefore, you are advised to use this method when the number of parallel jobs is insufficient.
- **eager**: All jobs are triggered synchronously. For jobs that depend on other jobs, prepare the environment and code first and wait until the dependency jobs are successfully executed. Resources may be idle, but the build time can be shortened. You are advised to use this mode when the parallel job quota is sufficient.

jobs

jobs is used to define jobs to be orchestrated. Each job must have a unique name as its identifier. If job A depends on job B, B has a higher priority. Jobs with the same priority are triggered synchronously.

The following is a code sample:

```
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 example, **Job3** depends on and has lower priority than **Job1** and **Job2**, which are triggered synchronously.

params

params can define global parameters to be shared by all 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 example, global parameters (**params**) are not defined. Instead, the **isSubmodule** parameter is defined in **Job1** and **Job2**.

NOTE

During a build with YAML, parameters are used in the following priority (as shown in the preceding sample code):

Runtime parameters configured on the **Parameters** page > Default values of parameters configured on the **Parameters** page > Parameters defined in **build_ref** > Parameters defined in **params** under a job > Global parameters defined in **params** under BuildFlow

5 Configuring a Build Task

5.1 Performing Basic Configurations

5.1.1 Configuring the Build Environment

Configure a global runtime environment for a build task.

CodeArts Build also allows you to use custom executors, such as **LINUX**, **LINUX_DOCKER**, **WINDOWS**, and **MAC** (Linux, Linux Docker, Windows, and macOS executors). For build scenarios supported by these executors, see **Table 5-1**.

Table 5-1 Executor types and their description

Executor Type	Description
LINUX	• You can run shell commands to run the build task on a Linux VM. • Before using CodeArts Build, you need to install build tools, such as Maven and Gradle, on custom executors. • Only the following build actions are available: **Running Shell Commands**, **Uploading a Software Package to Release Repos**, and **Downloading a Package from Release Repos**.

Executor Type	Description
LINUX_DOCKER	• When you run the build task, CodeArts Build starts a Linux Docker container in which the task is run. • The entire build process runs in the container. Once the task is finished, the container automatically removes the build image, which includes the code pulled during the build, the process data, and the build products. • You can configure the mapping between the host directory and container directory to share the host directory in the image. • The following build actions are functions are unavailable: **Download File from File Manager**, **Building with Android**, and **Unit Test** in **Building with Maven**.
WINDOWS	• You will run build tasks on the Windows executor. This allows you to execute Windows-related build tasks. • You can use Git Bash to run the shell script for your build. • Only the following build actions are available: **Running Shell Commands**, **Uploading a Software Package to Release Repos**, and **Downloading a Package from Release Repos**. • You can use Windows 7, Windows 10, Windows Server 2012, or Windows Server 2016. • Before customizing a Windows executor, ensure that you have installed the JDK and Git. • Install the compilation tool. For example, install Maven if you will use it for your build.
MAC	• You will run shell commands on the macOS executor for your build. This allows you to execute macOS-related build tasks. • Only the following build actions are available: **Running Shell Commands**, **Uploading a Software Package to Release Repos**, and **Downloading a Package from Release Repos**. • You can select any macOS version in use.

Constraints

When building with YAML, you have an available environment with custom executors.

Build on GUI

1. **Access the CodeArts Build Homepage** from the project list.
2. On the CodeArts Build homepage, search for the target task and click its name. The build task configuration page is displayed.

   CodeArts Build presets the **Configure Build Environment** action. Set the parameters according to **Table 5-2**.

Table 5-2 Build environment parameters

Parameter	Description
Environment	x86/Kunpeng (Arm) server **NOTE** Select the appropriate type of host you intend to use for software running on different chipset architecture. For example, if your software is designed for Arm (Kunpeng) servers, select **Arm (Kunpeng)**.
Execution Host	Select the compute resource used to run your build task. In CodeArts Build, virtual machines (VMs) are used. Executors can be built-in or custom executors. • **Built-in executor**: Provided by CodeArts Build with out-of-the-box availability. The default executor specifications are 2 vCPUs and 8 GB memory. • **Custom executor**: Compute resources provided by users. They can be hosted in CodeArts Build after registration. CodeArts Build schedules these executors to run build tasks. For details, see **Table 5-1**. 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.
Mapping Between Host and Container Directories	Set up the directory mapping between the custom executor and the container, and then you can mount files like dependencies from the custom executor to the container for your builds. (This mapping needs to be set when **Execution Host** is set to **Custom executor**.) If the **Host Directory** is set to **/home** and the **Container Directory** is set to **/opt**, then the content in the executor's local **/home** directory will be mounted to the **/opt** directory in the container.

Build with Code

Modify **env** settings in **the YAML file** by referring to the following sample code of build environment configurations.

```
version: 2.0 # The version number is a mandatory and unique parameter that must be set to 2.0.
env: # Optional. It defines the build environment. x86 is used by default if neither env or envs is set.
  resource:
    type:docker # Agent pool type. The value can be docker or custom. docker indicates that the default executor is used, and custom indicates that a custom executor is used.
    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.
```

Add the following code information to **the YAML file for your code-based build** by referring to the following sample code of BuildSpace.

NOTE

You have an available environment with custom executors.

```
version: 2.0
buildspace: # BuildSpace is used.
  fixed: true
  path: kk
  clean: true
  clean_exclude:
    - cache # Excluded path
    - aa # Excluded path
    - bb # Excluded path
```

Table 5-3 Parameters in the sample code of BuildSpace

Parameter	Type	Description
fixed	String	Optional. In CodeArts Build, an empty path (for example, **/devcloud/ws/sMMM/workspace/j_X/**) is randomly assigned to a build task as the root directory by default. This directory is called a "BuildSpace". Even for build tasks in the same project, BuildSpaces are randomly assigned to them. However, a fixed BuildSpace path is necessary in some scenarios. CodeArts Build allows users to configure BuildSpace to specify a fixed directory for a build. ● **true**: A fixed path is used. ● **false**: A random path is used. The default value is **false**.
path	String	Optional. The fixed path is in the following format: **/devcloud/slavespace/usr1/+"${*domainId*}"+/**. You can set the **path** parameter to add a path after the fixed path. For example, if the **path** is set to **kk**, the fixed path is **/devcloud/slavespace/usr1/+"${*domainId*}"+/kk**.

Parameter	Type	Description
clean	String	Optional. • **true**: The fixed path will be cleared. Files in the fixed path will be deleted each time the build task is complete. • **false**: The fixed path will not be cleared. When the total size of files reaches the maximum capacity of the workspace, you need to manually free up space by setting **clean** to **true**. **NOTE** • If there is no clearance setting for the fixed path, all files within the current tenant's fixed path will be automatically deleted once the total file size reaches the upper limit of the workspace. • The workspace refers to the custom executor specification. The default value is **true**.
clean_exclude	String	Optional. Specify paths to exclude from the cleanup. Only level-1 folders in a fixed path can be specified.

5.1.2 Configuring the Code Download

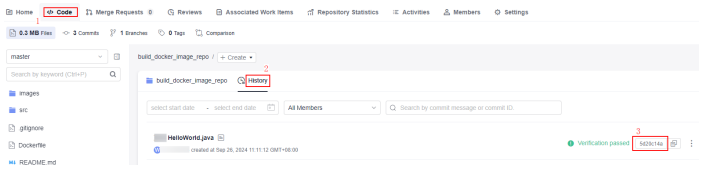
In this action, you set the download mode for pulling code from the repository during the build process.

Build on GUI

You can specify a source version with a code repository tag or commit ID. Besides, you can enable auto update of submodules and Git Large File Storage (LFS) for your build.

The **Configure Code Download** action is preset. Set the parameters according to **Table 5-4**.

Table 5-4 Parameters for configuring the code download

Parameter	Description
Specify Repository Tag or Commit ID	Specify whether to specify a tag or a commit ID when running a build task. • **Do Not specify**: All code is pulled for your build. • **Tag**: Only the code with the specified tag is pulled for your build. When you run the build task, a dialog box will appear prompting you to enter a tag. A tag marks a point in a code repository. If you select **Repo** as the code source, you can create a tag by referring to **Managing Tags**. When using a third-party code repository as the source, you need to create a tag within that repository. • **Commit ID**: Only the code with the specified commit ID is pulled for your build. When you run the build task, a dialog box will appear prompting you to enter a commit ID. A commit ID is the number generated when the code is committed. For example, the commit ID in a CodeArts Repo repository is shown in **Figure 5-1**. **Figure 5-1** Commit ID 
Clone Depth	Optional. The clone depth is the number of recent commits that will be cloned. A larger value indicates more commits will be fetched. The clone depth must be a positive integer. The recommended maximum value is 25. For example, setting the clone depth to **5** instructs the system to fetch only the five most recent commits, but no earlier records.
Auto Update	A submodule is a Git tool used to manage shared repositories for higher team efficiency. Submodules allow you to keep a shared repository as a subdirectory of your repository. You can isolate and reuse repositories, and pull latest changes from or push commits to shared repositories. For details, see **Configuring a 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.

Parameter	Description
Enable Git LFS	Determine whether to enable Git LFS to pull all files, including large files, such as audios, videos, and images, during a build. By default, these files are not pulled.

Build with Code (Downloading Code from a Single Repo)

Modify the code in the **PRE_BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  PRE_BUILD:
    - checkout:
        name: checkout
        inputs:
          scm: codehub # Code source: CodeArts Repo only
          url: xxxxxxxxx # This refers to the SSH address of the repository that the code is pulled from.
          branch: ${codeBranch} # Mandatory at any time and can be parameterized.
          commit: ${commitId}
          lfs: true
          submodule: true
          depth: 100
          tag: ${tag}
          path: test
```

Table 5-5 Parameters in the sample code for downloading a single repository

Parameter	Type	Description
scm	String	Enter a code source. Currently, only CodeArts Repo is supported. If this parameter is not configured in the YAML file, the code repository information configured in the build task will be used. The default value is **codehub**.
url	String	Enter the SSH address of the code repository from which the code will be pulled.
branch	String	Specify the branch from which to pull the code. You can use *${codeBranch}* to reference this parameter.
commit	String	Optional. If needed, you can enter a commit ID to indicate the specific version of the source code to be pulled for your build. You can also use *${commitId}* to reference this parameter.
tag	String	Optional. If needed, you can enter a tag to indicate the specific version of the source code to be pulled for your build. You can also use *${tag}* to reference this parameter. If you provide both the commit ID and tag, the build using the commit ID will be run first.

Parameter	Type	Description
depth	Int	Optional. 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. The default value is **1**.
submodule	Bool	Optional. Specify whether to pull the submodules. • **true**: Pull. • **false**: Do not pull. The default value is **false**.
lfs	Bool	Optional. Specify whether to enable Git LFS. • **true**: Enable. • **false**: Disable. By default, CodeArts Build does not pull large files such as audios, videos, and images. You can enable Git LFS to pull all files. The default value is **false**.
path	String	Optional. Sub-path for cloning: The code is downloaded to the sub-path.

Build with Code (Downloading Code from Multiple Repos via Manifest)

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

CodeArts Build integrates the Repo download tool. You only need to perform simple configurations to download multiple code repositories. Currently, this feature only applies to CodeArts Repo repositories.

Modify the code in the **PRE_BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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}"
```


Table 5-6 Parameters in the sample code for downloading multiple repositories via Manifest

Parameter	Type	Description
name	String	Optional. Enter the action name. The default value is **manifest_checkout**.
manifest_url	String	Enter the address of a Manifest repository that includes an XML file.
manifest_branch	String	Optional. Enter the Manifest branch or revision. The default value is **HEAD**.
manifest_file	String	Optional. Manifest file path. The default value is **default.xml**. The repositories defined in **manifest_file** must be of the same code source.
path	String	Optional. Download path of all sub-repositories of the custom Manifest file, which is the relative path of the working path. The path cannot start with a slash (/) and cannot contain any period (.). The default value is the working path.
repo_url	String	Optional. Enter the address of the CodeArts Repo repository. The default value is **https://gerrit.googlesource.com/git-repo**. You must have been granted the download permission for the CodeArts Repo repository specified by **repo_url** (the repository can either be open-source or private, but configured with account and password).
repo_branch	String	Optional. Enter the branch of the CodeArts Repo repository. The default value is **stable**.
username	String	Optional. Enter the username used for downloading the repository. This parameter is mandatory for private repositories.
password	String	Optional. Enter the HTTPS password used for downloading the repository. This parameter is mandatory for private repositories.

- **manifest_url** and **manifest_file** must use the same code source. For private repositories, the account indicated by the **username** and **password** you use must have been granted the download permission.
- If the optional parameters mentioned above are left empty, default values will be used.
- When using private repositories, you are advised to configure the username and password using private build parameters. For details, see **Configuring Parameters**.

5.2 Selecting Build Actions

You can select desired build actions that suits your scenarios.

Build on GUI

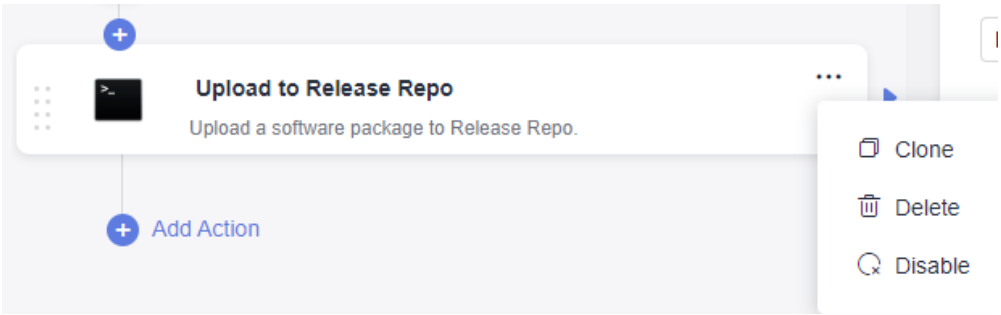
Go to the **Build Actions** page, where you can see the default action combination of the selected template.

- You can click on a build action to add it to your build task. For details about how to configure each build action, see their "Build on GUI" instructions in **Configuring Build Actions**.

  If the tool version preset in the build action cannot meet your requirements, you can **customize the build environment**.
- To delete a build action, select it and choose > **Delete**.
- To copy a build action, select it and choose > **Clone**.
- To disable a build action that is reserved, select it and choose > **Disable**.

  To enable the action again, select it and choose > **Enable**.

Figure 5-2 Adding, cloning, deleting, and disabling a build action



Build with Code

- To set up the environment for running your code-based build task, set **env** of **the YAML file** by referring to the sample code in the "Build as Code" of **Configuring the Build Environment**.
- To set up the code download mode, set **PRE_BUILD** of **the YAML file** by referring to the sample code in the "Build as Code" of **Configuring the Code Download**.
- To set up the build actions, set **BUILD** of **the YAML file** by referring to the sample code in the "Build as Code" of each build action in **Configuring Build Actions**.

5.3 Configuring Build Actions

5.3.1 Building with Maven

Maven's primary goal is to build Java-based projects and manage project information and dependencies.

Constraints

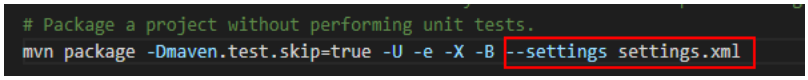
- If you need to use a custom **settings** file for a Maven build, ensure it meets the following requirements:
  - The maximum file size is 100 KB.
  - The file type must be **.xml**, **.key**, **.keystore**, **.jks**, **.crt**, or **.pem**.
  - A maximum of 20 files can be uploaded.

Build on GUI

Add **Build with Maven**, when **configuring build actions**. Set the parameters according to **Table 5-7**.

Table 5-7 Parameters for building with Maven

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Maven commands, or use the default ones. For more commands, see the **Maven official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Parameter	Description
setting File Configuration	• **Generate setting File with Repositories**: The optimal repository access mode is automatically configured based on your IP address when you access the **setting.xml** file provided by CodeArts. Your IP address may be in regions in or outside the current country. You are advised to retain the default settings. The **setting.xml** file defines the default dependency pull sequence and mirror source proxy. If you need to use a custom **setting.xml** file, **add a custom setting.xml file** and add **--settings settings.xml** to the end of the default packaging command. Then, you can use the added **settings.xml** file to build with Maven.  • **Public Repositories**: By default, Huawei Mirrors is added, and Huawei SDK repositories has been configured. This configuration is used only when you need to add a public repository that is not provided by CodeArts. The procedure is as follows: 1. Click **Add**. 2. Enter the repository address, and select **Release** and **Snapshot** as required. Select either **Release** or **Snapshot**, or both. **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. • **Private Repositories**: Self-hosted repos provided by CodeArts have been configured by default. This configuration is used only when you need to add other private repository. The procedure is as follows: 1. Create a Nexus repository service endpoint by referring to **CodeArts** *User Guide* > "Preparations" > "Creating Service Endpoints". 2. Click **Add**, select the service endpoint created in the previous step, and select **Release** and **Snapshot** as required.

Parameter	Description
	NOTE **Release** and **Snapshot** are two types of repositories. Pay attention to their differences. If you upload a dependency to a release repository, it cannot be downloaded during a build. • **Snapshot**: 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.

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. Before the configuration, **create a self-hosted repo**. The configuration procedure is as follows:
	• **Do not configure POM**: Private dependencies do not need to be released to the self-hosted repo of CodeArts Artifact.
	• **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 a self-hosted repo. In the command window, use the number sign (#) to comment out the **mvn package -Dmaven.test.skip=true -U -e -X -B** command, as shown in the following figure.
	`# Package a project without performing unit tests.`
	`#mvn package -Dmaven.test.skip=true -U -e -X -B`
	Delete the number sign (#) before the **#mvn deploy -Dmaven.test.skip=true -U -e -X -B** command, as shown in the following figure.
	`# 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`
	The uploaded private dependency can be referenced by adding the **groupId**, **artifactId**, and **version** coordinates in the **pom.xml** file to other projects.

Unit Test	To process unit test results, set the parameters. For details, see **Configuring a Unit Test**.

Parameter	Description
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. The dependency cache files of a Maven build task cannot be updated. The cache directory will be updated only when new dependencies are imported to the task.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - maven:
        image: cloudbuild@maven3.5.3-jdk8-open
        inputs:
          settings:
            public_repos:
              - https://mirrors.example.com/maven
          cache: true # Determine whether to enable caching.
          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
          ignore_fail: true
```

Table 5-8 Parameters in the sample code

Parameter	Type	Description
image	String	The image address can be in either of the following formats: • Use **cloudbuild@maven3.5.3-jdk8-open**. This address starts with **cloudbuild** and uses the at sign (@) as a separator, with the default image version provided by CodeArts Build following it. • Use a complete SWR image path, for example, **swr.example.example.com/codeci_test/demo:141d26c455abd6d7***xxxxxxxxxxxxxxxxxxx*.

Parameter	Type	Description
settings	Map	Optional. If this parameter is not set, the **setting.xml** file provided by CodeArts is used by default. If you need to use a custom **settings.xml** file, **add a custom setting.xml file** and add **--settings settings.xml** to the end of the default packaging command **mvn package -Dmaven.test.failure.ignore=true -U -e -X -B**.
cache	Bool	Optional. Specify whether to enable cache. • **true**: Enable. • **false**: Disable. The default value is **false**.
command	String	Configure the Maven command. For more commands, see the **Maven official website**.
unit_test	Map	Optional. Configure the unit test. For details, see **Configuring a Unit Test**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

Adding a Custom setting.xml File

The file restrictions are as follows:

- The maximum file size is 100 KB.
- The file type must be **.xml**, **.key**, **.keystore**, **.jks**, **.crt**, or **.pem**.
- A maximum of 20 files can be uploaded.

You can add files in either of the following ways:

- **Build on GUI**
  a. In the **Commands** window of the **Build with Maven** action, run the **cat /home/build/.m2/settings.xml** command. After the task is complete, the content of the **settings.xml** file will be displayed in the build logs.
  b. Customize a new **settings.xml** file according to the **settings.xml** file's information displayed in the build logs.
  c. Add the **Download File from File Manager** action before the **Build with Maven** action.
     Assign a custom name to the action and select a tool version. Currently, only **shell4.2.46-git1.8.3-zip6.00** is supported.
  d. Click **Upload**. In the displayed dialog box, select the file created in **b**, add a description, select the agreements, and click **Save**.

e. Expand the **File Name** drop-down list and select the uploaded **setting.xml** file.

- **Build with Code**

  Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - download_file:
        inputs:
          name: settings.xml
          ignore_fail: true
```

Table 5-9 Parameters in the sample code for downloading a file

Parameter	Type	Description
name	String	Name of the setting file.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

You can access the uploaded files in either of the two ways.

- On the CodeArts Build homepage, click **More** and select **Files**.
- Alternatively, click **Manage Files** in the **Download File From File Manager** action.

On the **Files** page, you can edit, download, and delete files, as well as configure operation permissions for other users.

- Enter a keyword in the search box to search for a file.
- 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 select **Delete** from the drop-down list. Confirm the deletion as prompted.
- Click in the **Operation** column and select **Modify Permissions** from the drop-down list. In the displayed dialog box, configure file operation permissions for the user.

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

Figure 5-3 Configuring file operation permissions for a user

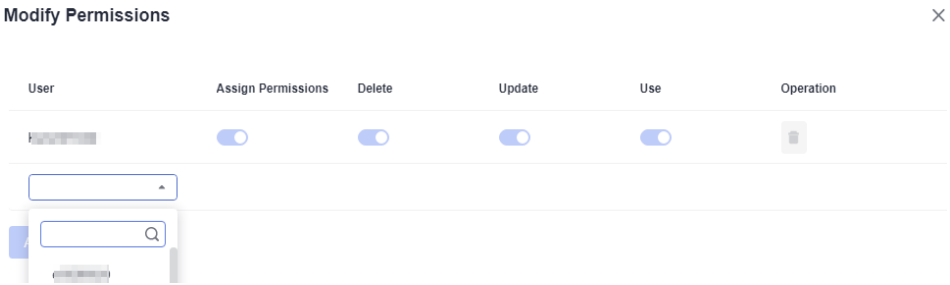


Table 5-10 Roles and their permissions on files

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

Configuring a Unit Test

The Maven build action leverages unit tests to ensure that your code's core functions operate as expected. Within a Maven project, testing frameworks like JUnit are used to write and execute these tests.

1. Before configuring a unit test, you need to write 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.
   - The project cannot contain the configuration code for ignoring unit test cases. Specifically, ensure that the **pom.xml** file of the project does not contain the following code:

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



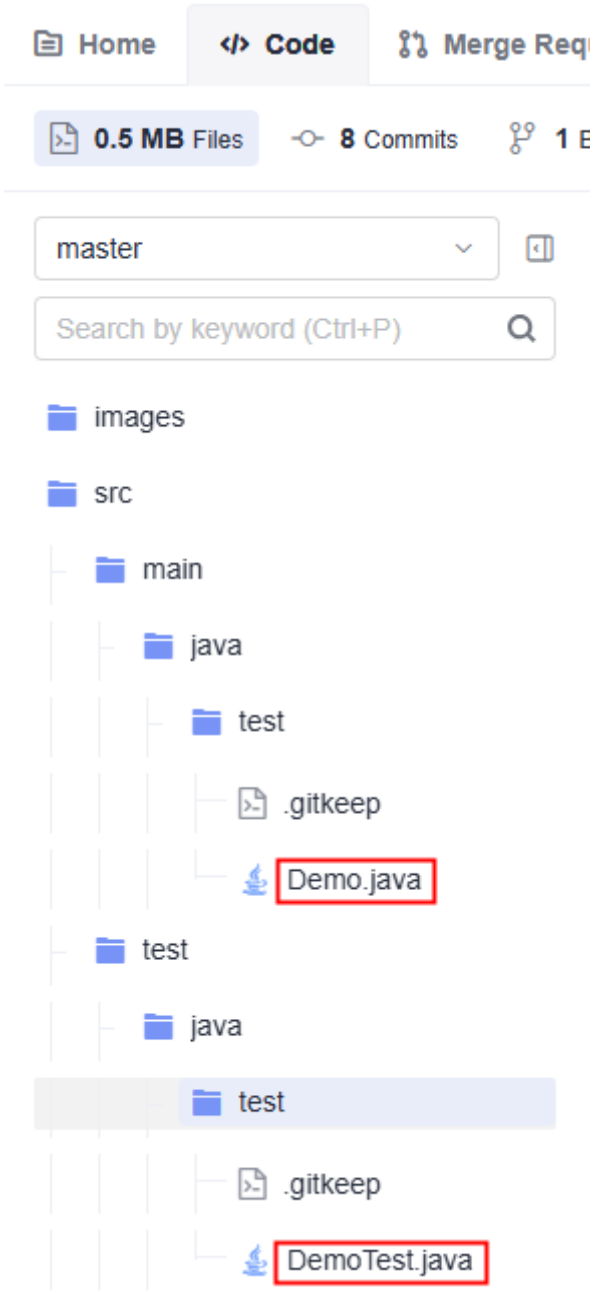
```
    </configuration>
</plugin>
```

- The JUnit dependency needs to be added to the **pom.xml** file. The following is the sample code that needs to be added:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.1</version>
</dependency>
```

2. Create a unit test class in the code repository, as shown in **Figure 5-4**.

Figure 5-4 Unit test file directory



The following sample code is from the **Demo.java** file:


```
package test;

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

The following sample code is from the **DemoTest.java** file:

```
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");
    }
}
```

3. Configure a unit test in the build action.
   - **Build on GUI**
     i. In the command window displayed in action **Build with Maven**, use the number sign (#) 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
     ```

     ii. Delete the number sign (#) before the **#mvn package -Dmaven.test.failure.ignore=true -U -e -X -B** command.

     ```
     # 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
     ```

     iii. Expand **Unit Test** and set the following parameters **Table 5-11**.

Table 5-11 Parameters for unit test configuration

Parameter	Description
Print Test Results	Specify whether to process unit test results. • **Yes**: Process unit test results. • **No**: Do not process unit test results.

Parameter	Description
Ignore Test Failure	If you choose to process the unit test result, you need to configure whether to ignore the case failure. • **Yes**: If the case fails, the build task will continue. • **No**: If the case fails, the build task will also fail.
Test Report	The unit test results are aggregated into a visual test report. To retrieve these results, you need to specify the path of the unit test result files. In most cases, retain the default path ****/TEST*.xml**. To improve the accuracy of the result, specify a precise report path, for example, **target/surefire-reports/TEST*.xml**.
Print Unit Test Results	Specify whether to process unit test coverage results. If you select **Yes**, a coverage test report is generated. For details about the configuration, see **Generating a Unit Test Coverage Report Using JaCoCo**.
Report Location	If you choose to process the unit test coverage results, enter the relative path to the project root directory, for example, **target/site/jacoco**. Once you have done this, all files in that directory will be packaged and uploaded.

- **Build with Code**

  Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - maven:
          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
```


Table 5-12 Unit test parameters

Parameter	Type	Description
enable	Bool	Optional. Specify whether to process unit test results. • **true**: Process unit test results. To set this parameter to **true**, add the **-Dmaven.test.failure.ignore=true** parameter to the end of the **mvn** command. • **false**: Do not process unit test results. The default value is **true**.
ignore_errors	Bool	Optional. Specify whether to ignore test failure. • **true**: Ignore test failure. If the case fails, the build task will continue. • **false**: Do not ignore test failure. If the case fails, the build task will also fail. The default value is **true**.
report_path	String	Enter the path for storing unit test data. Specify an accurate report path, for example, **target/surefire-reports/TEST*.xml**.
coverage	Bool	Optional. Specify whether to process coverage data. If you want to set this parameter to **true**, see **Generating a Unit Test Coverage Report Using JaCoCo**. • **true**: Process coverage data. • **false**: Do not process coverage data. The default value is **false**.
coverage_report_path	String	Optional. If you choose to process the unit test coverage results, enter the relative path to the project root directory, for example, **target/site/jacoco**. Once you have done this, all files in that directory will be packaged and uploaded.

4. Once the task is successfully completed, you can access the test report on the testing tab of the task execution details page. When you opt to print the unit test coverage report, a report is generated. You can download it by clicking the button for downloading the test coverage report.

Generating a Unit Test Coverage Report Using JaCoCo

The unit test coverage report distinguishes between the code that has been exercised through unit tests and those that have not been covered. It helps you understand whether your tests have checked all code paths and logic.

If you set **Print Unit Test Results** to **Yes**, complete configurations as follows:

- **Configuration method for a single-module project**

  To generate the unit coverage report, add **jacoco-maven-plugin** to the project by including the following configuration to the **pom.xml** file.

  By default, the JaCoCo report goal is bound to the **verify** phase. You need to change the report goal to the **test** phase.

```
<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> #It defines the target phase of the report.
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

- **Configuration method for a multi-module project**

  Assume that the code structure of a multi-module project looks like the following example. You will walk through how to configure and generate a unit test report.

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

  a. Add an aggregation module named **report** to the project. Your code directory structure then looks like this:

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

  b. Add **jacoco-maven-plugin** to the pom.xml file in the root directory of the project. The following is a code sample:

```
<!-- 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.

Use **dependency** elements to introduce all dependency modules and use **report-aggregate** to define the JaCoCo aggregation goal.

```
<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>
```

d. Once the configuration is completed, 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 an output path for the reports by configuring **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>
```

5.3.2 Building 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.

Build on GUI

1. Add **Build with Android**, when **configuring build actions**. Set the parameters according to **Table 5-13**.

   Table 5-13 Parameters for building with Android

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Gradle	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
JDK	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
NDK	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Gradle commands, or use the default ones. For more commands, see the **Gradle official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

2. If you need to use apksigner to sign Android APKs, add the **Sign Android APK** action and configure the parameters according to the following table.

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
APK Location	Location of the **.apk** file to be signed after the Android build. You can use regular expressions, such as **build/bin/*.apk**, to match the built APK package.
Keystore File	Select the keystore file used for signature from the drop-down list. For details about how to create and upload the file, see **Generating the Keystore Signature File and Uploading It for Management**.
Keystore Password	Optional. Enter the custom password of the keystore file.
Alias	Assign a custom alias to the key. • The value must start with a letter and can contain letters, digits, underscores (_), hyphens (-), and periods (.). • The value contains 1 to 128 characters.
Key Password	Optional. Enter a custom key password.
Apksigner Command	Enter a custom signature parameter. By default, **--verbose** is added to display the signature details.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Once you have finished configuration, run the build task. If the task is executed successfully, check the build logs. If the logs of the **Sign Android APK** action display **Signed**, it means that the signing process was successful.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

1. The following sample code is for the 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 Build is used for cache 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
    ignore_fail: true
```

Table 5-14 Parameters in the sample code for the Android build

Parameter	Type	Description
command	String	Enter the Gradle command. For more commands, see the **Gradle official website**.
gradle	String	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
jdk	String	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
ndk	String	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

2. The following sample code is to sign the 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
    ignore_fail: true
```

Table 5-15 Parameters in the sample code for signing the Android APK

Parameter	Type	Description
file_path	String	Location of the **.apk** file to be signed after the Android build. You can use regular expressions, such as **build/bin/*.apk**, to match the built APK package.
keystore_file	String	Name of the keystore file. For details about how to create and upload the file, see **Generating the Keystore Signature File and Uploading It for Management**.
keystore_password	String	Optional. Enter the custom password of the keystore file.
alias	String	Alias of the keystore file. • The value must start with a letter and can contain letters, digits, underscores (_), hyphens (-), and periods (.). • The value contains 1 to 128 characters.
key_password	String	Optional. Enter a custom key password.
apksigner_commond	String	Enter a custom signature parameter. By default, **--verbose** is added to display the signature details.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

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

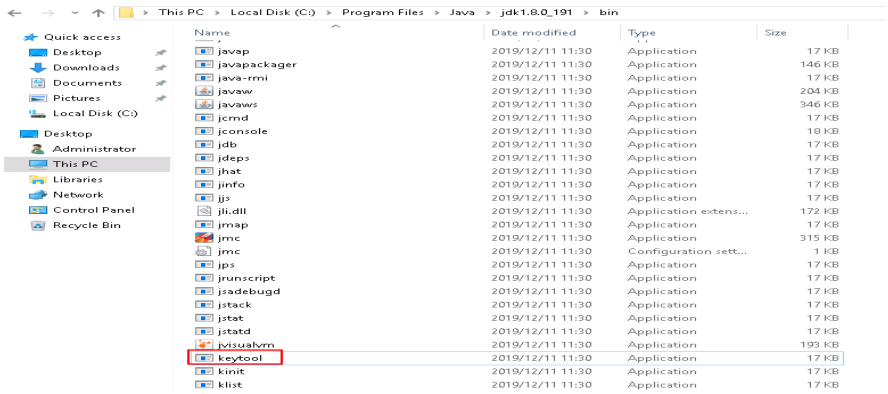
```
apply plugin: 'com.android.application'

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**.

Generating the Keystore Signature File and Uploading It for Management

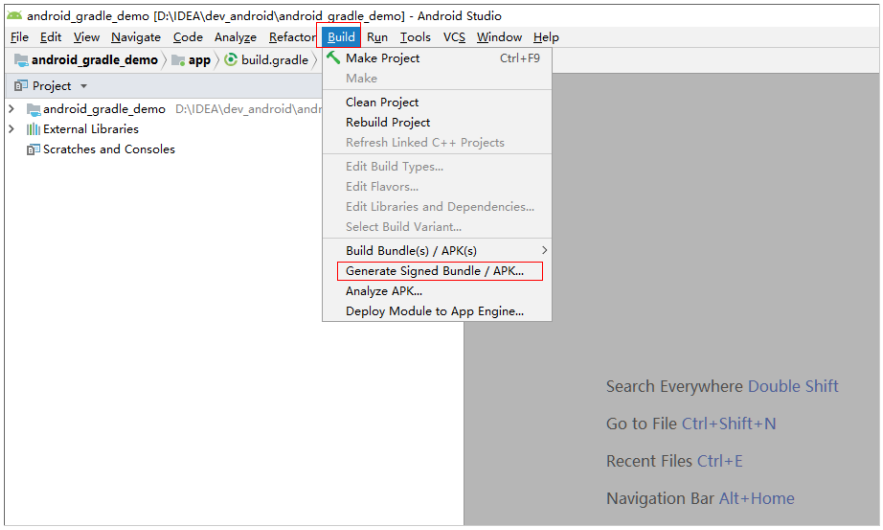
1. The keystore signature file can be generated in either of the following ways:
   - **Using Keytool in JDK to Generate Signature Files**
     i. Find the JDK installation path and run **keytool.exe**.



     ii. Run the following command to generate a **.jks** file:

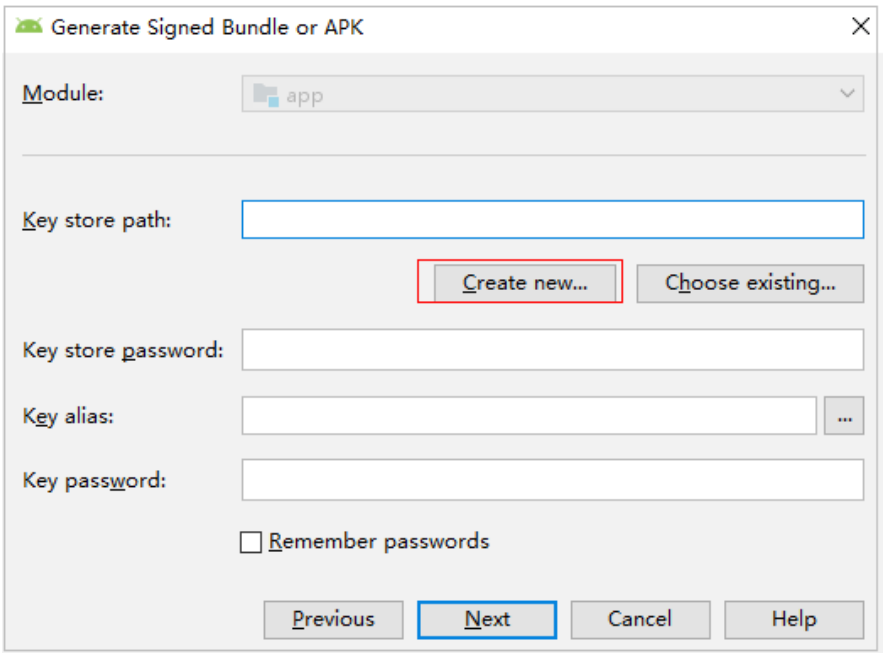
```
keytool -genkeypair -storepass 123456 -alias apksign -keypass 123456 -keyalg RSA -validity 20000 -keystore D:/android.jks
```

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

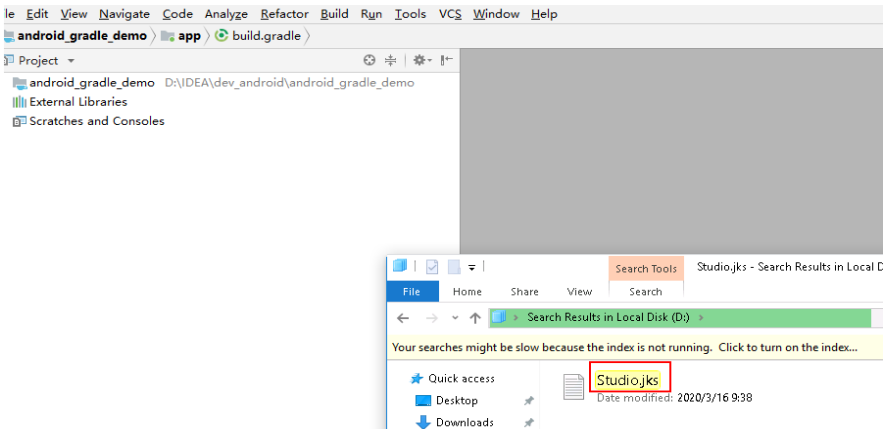


ii. Select **APK** and click **Next**.

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



iv. View the generated signature file.



2. You can upload the keystore signature file to the **Files** page in either of the following ways:
   - In the **Build with Android** action, click **Upload** next to **Keystore File**. In the displayed dialog box, select a file, add a description, select the related agreements, and click **Save**.
   - On the CodeArts Build homepage, choose **More** > **Files**. On the displayed page, click **Upload File**. In the displayed dialog box, select a file, add a description, select the related agreements, and click **Save**.

   On the **Files** page, you can edit, download, and delete files, as well as configure operation permissions for other users.
   - Enter a keyword in the search box to search for a file.
   - 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 select **Delete** from the drop-down list. Confirm the deletion as prompted.
   - Click in the **Operation** column and select **Modify Permissions** from the drop-down list. In the displayed dialog box, configure file operation permissions for the user.

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

     Figure 5-5 Configuring file operation permissions for a user

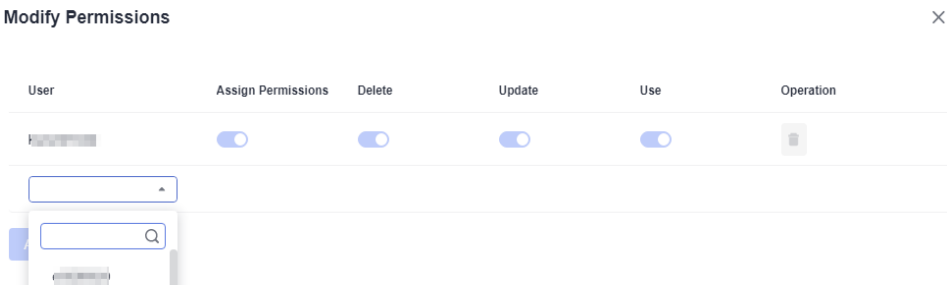


Table 5-16 Roles and their permissions on files

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

5.3.3 Building with npm

Build Vue and Webpack projects with npm.

Build on GUI

Add **Build with npm**, when **configuring build actions**. Set the parameters according to **Table 5-17**.

Table 5-17 Parameters for building with npm

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the npm commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Node.js official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - npm:
        image: cloudbuild@nodejs8.11.2
        inputs:
          command: |
            export PATH=$PATH:~/.npm-global/bin
            #Set the cache directory.
            npm config set cache /npmcache
            npm config set prefix '~/.npm-global'
            npm config set registry https://repo.example.com/repository/npm/
            #For Node.js 18 or higher: Configure the .npmrc file. For earlier versions: use these command lines.
            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 install --verbose
            npm run build
          ignore_fail: true
```

Table 5-18 Parameters in the sample code

Parameter	Type	Description
image	String	The image address can be in either of the following formats: • Use **cloudbuild@nodejs8.11.2**. This address starts with **cloudbuild** and uses the at sign (@) as a separator, with the default image version provided by CodeArts Build following it. • Use a complete SWR image path, for example, **swr.example.example.com/codeci_test/demo:141d26c455abd6d7***xxxxxxxxxxxxxxxxxxxx*.
command	String	Configure the npm commands. For more commands, see the **Node.js official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.4 Building with Gradle

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

Build on GUI

Add **Build with Gradle**, when **configuring build actions**. Set the parameters according to **Table 5-19**.

Table 5-19 Parameters for building with Gradle

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Gradle	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
JDK	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Gradle commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Gradle official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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 is used for cache 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
          ignore_fail: true
```


Table 5-20 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Gradle commands. For more commands, see the **Gradle official website**.
gradle	String	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
jdk	String	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.5 Building with Yarn

Build a JavaScript project with Yarn.

Build on GUI

Add **Build with Yarn**, when **configuring build actions**. Set the parameters according to **Table 5-21**.

Table 5-21 Parameters for building with Yarn

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Description
Commands	Configure the Yarn commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Yarn official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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 ./**
          ignore_fail: true
```

Table 5-22 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Yarn commands. For more commands, see the **Yarn official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.6 Building with Gulp

Build a frontend IDE with Gulp.

Build on GUI

Add **Build with Gulp**, when **configuring build actions**. Set the parameters according to **Table 5-23**.

Table 5-23 Parameters for building with Gulp

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Gulp commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Gulp official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```


Table 5-24 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Gulp commands. For more commands, see the **Gulp official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.7 Building with Grunt

Build a JavaScript project with Grunt.

Build on GUI

Add **Build with Grunt**, when **configuring build actions**. Set the parameters according to **Table 5-25**.

Table 5-25 Parameters for building with Grunt

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Grunt commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Grunt official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:


```
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
          ignore_fail: true
```

Table 5-26 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Grunt commands. For more commands, see the **Grunt official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.8 Building with Mono

Use Mono for MSBuild and .NET builds.

Build on GUI

Add **mono**, when **configuring build actions**. Set the parameters according to **Table 5-27**.

Table 5-27 Parameters for building with Mono

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Description
Commands	Configure the Mono commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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/*
          ignore_fail: true
```

Table 5-28 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Mono commands.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.9 Building in PHP

Use PHP to build within an installation and packaging environment that includes the necessary PHP code libraries for the project.

Build on GUI

Add **Build in PHP**, when **configuring build actions**. Set the parameters according to **Table 5-29**.

Table 5-29 Parameters for building in PHP

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the PHP commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **PHP official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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 *
          ignore_fail: true
```

Table 5-30 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the PHP commands. For more commands, see the **PHP official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.10 Building with Setuptools

Use Setuptools to package Python applications.

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 setup file, see the **official instructions of Python**.

Build on GUI

Add **Build with Setuptools**, when **configuring build actions**. Set the parameters according to **Table 5-31**.

Table 5-31 Parameters for building with Setuptools

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the pack 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. For more commands, see the **Setuptools official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
       ignore_fail: true
```

Table 5-32 Parameters in the sample code

Parameter	Type	Description
name	String	Optional. Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
image	String	Optional. Enter the image version, which should include the fixed part **cloudbuild@** and the supported Python version following it. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**. The default value is **cloudbuild@python3.6**.
command	String	Configure the pack 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. For more commands, see the **Setuptools official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.11 Building with PyInstaller

Use PyInstaller to package Python scripts into standalone executables.

Build on GUI

Add **Build with PyInstaller**, when **configuring build actions**. Set the parameters according to **Table 5-33**.

Table 5-33 Parameters for building with PyInstaller

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the build and packaging commands or use the default commands, which will package the project into an executable. For more commands, see the **PyInstaller official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```


Table 5-34 Parameters in the sample code

Parameter	Type	Description
name	String	Optional. Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
image	String	Optional. Enter the image version, which should include the fixed part **cloudbuild@** and the supported Python version following it. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**. The default value is **cloudbuild@python3.6**.
command	String	Configure the build and packaging commands or use the default commands, which will package the project into an executable. For more commands, see the **PyInstaller official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.12 Running Shell Commands

You can use the action **Run Shell Commands** to create and run a build task. You can also use this action in conjunction with other build tools. For instance, in a Maven build, you can add the **Run Shell Commands** action to generate the necessary files for the subsequent build process.

Build on GUI

Add **Run Shell Commands**, when **configuring build actions**. Set the parameters according to **Table 5-35**.

Table 5-35 Parameters for running shell commands

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Enter the shell commands for your build.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **PRE_BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

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

Table 5-36 Parameters in the sample code

Parameter	Type	Description
command	String	Enter the shell commands for your build.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.13 Building with GNU Arm

Compile and build software for Arm processors.

Build on GUI

Add **Build with GNU Arm**, when **configuring build actions**. Set the parameters according to **Table 5-37**.

Table 5-37 Parameters for building with GNU Arm

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Commands	Configure the GNU Arm build commands, or use the default **make** command.
	• 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:
	– **Optional:** Image **gnuarm201405**: Run the **arm-none-linux-gnueabi-gcc** command as follows:
	`arm-none-linux-gnueabi-gcc -o main main.c`
	– Image **gnuarm-linux-gcc-4.4.3**: Run the **arm-linux-gcc** command as follows:
	`arm-linux-gcc -o main main.c`
	– Image **gnuarm-7-2018-q2-update**: Run the **arm-none-eabi-gcc** command as follows:
	`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 \#.

Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:


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

Table 5-38 Parameters in the sample code

Parameter	Type	Description

command	String	Configure the command for building with GNU Arm.
		• 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:
		– **Optional:** Image **gnuarm201405**: Run the **arm-none-linux-gnueabi-gcc** command as follows:
		`arm-none-linux-gnueabi-gcc -o main main.c`
		– Image **gnuarm-linux-gcc-4.4.3**: Run the **arm-linux-gcc** command as follows:
		`arm-linux-gcc -o main main.c`
		– Image **gnuarm-7-2018-q2-update**: Run the **arm-none-eabi-gcc** command as follows:
		`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 \#.

ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.14 Building with CMake

Build a cross-platform project with CMake.

Build on GUI

Add **Build with CMake**, when **configuring build actions**. Set the parameters according to **Table 5-39**.

Table 5-39 Parameters for building with CMake

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the CMake commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **CMake official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```

Table 5-40 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the CMake commands. For more commands, see the **CMake official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.15 Building with Ant

Build, test, and deploy a Java project using Ant.

Build on GUI

Add **Build with Ant**, when **configuring build actions**. Set the parameters according to **Table 5-41**.

Table 5-41 Parameters for building with Ant

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Ant build commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Ant official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - ant:
        inputs:
          command:  ant -f build.xml
          ignore_fail: true
```

Table 5-42 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Ant build commands. For more commands, see the **Ant official website**.

Parameter	Type	Description
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.16 Building with Go

In this action, you build a project with the Go language. This involves compiling the source code to produce executables, managing project dependencies, and customizing the build process.

Build on GUI

Add **Build with Go**, when **configuring build actions**. Set the parameters according to **Table 5-43**.

Table 5-43 Parameters for building with Go

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Go project build command, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Go official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
      ignore_fail: true
```

Table 5-44 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the command for building a Go project. For more commands, see the **Go official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.17 Building Android App with Ionic

In this action, you build an Ionic Android app, which is a mobile app that works across multiple platforms. This action allows you to quickly develop mobile apps, mobile web pages, hybrid apps, and web pages.

The project contains the project compilation description files such as **ionic.config.json**, **package.json**, and **angular.json**.

Build on GUI

Add **Build Android App with Ionic**, when **configuring build actions**. Set the parameters according to **Table 5-45**.

Table 5-45 Parameters for building an Android app with Ionic

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Gradle	Select a Gradle version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Description
JDK	Select a JDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
NDK	Select an NDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the packaging script in the command box. For more commands, see the **Ionic official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - ionic_android_app:
        inputs:
          gradle: '4.8'
          jdk: '33'
          ndk: '17'
          command: ./instrumented.apk
          ignore_fail: true
```

Table 5-46 Parameters in the sample code

Parameter	Type	Description
gradle	String	Select a Gradle version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Type	Description
jdk	String	Select a JDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
ndk	String	Select an NDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
command	String	Configure the packaging script in the command box. For more commands, see the **Ionic official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.18 Building an Android Quick App

Run the npm configuration command to build an Android quick app.

Build on GUI

Add **Build Android Quick App**, when **configuring build actions**. Set the parameters according to **Table 5-47**.

Table 5-47 Parameters for building an Android quick app

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure npm commands. For more commands, see the **Node.js official website**.

Parameter	Description
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
            # Default build
            npm run build
          ignore_fail: true
```

Table 5-48 Parameters in the sample code

Parameter	Type	Description
command	String	Configure npm commands. For more commands, see the **Node.js official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.19 Building in GFortran

Compile a single GFortran source code or an entire GFortran project.

Build on GUI

Add **Build in GFortran**, when **configuring build actions**. Set the parameters according to **Table 5-49**.

Table 5-49 Parameters for building in GFortran

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the GFortran commands. You can also use the default commands. If you have special build requirements, enter your custom build script in the text box.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```

Table 5-50 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the GFortran commands.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.20 Building with sbt

Build a Scala or Java project with sbt.

Build on GUI

Add **Build with sbt**, when **configuring build actions**. Set the parameters according to **Table 5-51**.

Table 5-51 Parameters for building with sbt

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Only the default version sbt1.3.2-jdk1.8 is supported currently.
Commands	Configure the sbt commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **sbt official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

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

Table 5-52 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the sbt commands. For more commands, see the **sbt official website**.

Parameter	Type	Description
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.21 Building with Grails

Build a web application with Grails.

Build on GUI

Add **Build with Grails**, when **configuring build actions**. Set the parameters according to **Table 5-53**.

Table 5-53 Parameters for building with Grails

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Grails commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Grails official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

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



```
command: grails war
ignore_fail: true
```

Table 5-54 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Grails commands. For more commands, see the **Grails official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.22 Building with Bazel

Use Bazel to compile and build.

Build on GUI

Add **Build with Bazel**, when **configuring build actions**. Set the parameters according to **Table 5-55**.

Table 5-55 Parameters for building with Bazel

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Bazel commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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/
          ignore_fail: true
```

Table 5-56 Parameters in the sample code

Parameter	Type	Description
command	String	Configure the Bazel commands.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.23 Building with Flutter

Build Android applications with Flutter.

Build on GUI

Add **Build with Flutter**, when **configuring build actions**. Set the parameters according to **Table 5-57**.

Table 5-57 Parameters for building with Flutter

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Flutter	Select a Flutter version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Description
JDK	Select a JDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
NDK	Select an NDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Commands	Configure the Flutter commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For more commands, see the **Flutter official website**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - flutter:
        inputs:
          flutter: '1.17.5'
          jdk: '3333'
          ndk: '23.1.7779620'
          command: ./instrumented.apk
          ignore_fail: true
```

Table 5-58 Parameters in the sample code

Parameter	Type	Description
flutter	String	Select a Flutter version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Type	Description
jdk	String	Select a JDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
ndk	String	Select an NDK version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
command	String	Configure the Flutter commands. For more commands, see the **Flutter official website**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.24 Running Docker Commands to Operate Images

Run Docker commands to perform image operations, such as login, push, and download.

Build on GUI

Add **Run Docker Commands**, when **configuring build actions**. Set the parameters according to **Table 5-59**.

Table 5-59 Parameters for running docker commands

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.

Parameter	Description
Commands	Click **Add** to add a command, and configure it as required. For details about the Docker commands supported by CodeArts Build, see **Docker Commands Supported by CodeArts Build**. You can drag and drop the commands into the desired execution sequence.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```

Table 5-60 Parameters in the sample code

Parameter	Type	Description
command	String	Each command takes up one line. For details about the supported Docker commands, see **Docker Commands Supported by CodeArts Build**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

Docker Commands Supported by CodeArts Build

- **docker login**: Log in to the Docker repository.

  Usage: docker login [options] [server]

  The following table describes how to set **options**. **server** indicates the Docker repository address.

Option	Short Form	Description
--password	-p	Password for logging in to the repository.
--username	-u	Username for logging in to the repository.
--password	-stdin	Password obtained from stdin

Example: **docker login -u jack -p 12345 mydocker-registry.com**

In this example, user **jack** remotely logs in to the **mydocker-registry.com** repository using password **12345**.

Advanced Usage

To read a password from the file, run **cat ~/my_password.txt | docker login --username jack --password-stdin**.

- **docker build**: Build an image from a Dockerfile or context. The context can be a local path (**Path**) where the build is executed, a remote URL (such as a Git repository, tarball, or text file), or a hyphen (-).

  Usage: docker build [options] Path | URL | -

  The following table describes how to set **options**. **Path/URL/-** indicates the context source.

Option	Short Form	Description
--file	-f	Dockerfile path. The default value is **./Dockerfile**.
--tag	-t	In the format of "*Image name*:*Tag*"

Example: **docker build -t mydocker-registry.com/org/alpine:1.0 -f ./Dockerfile .**

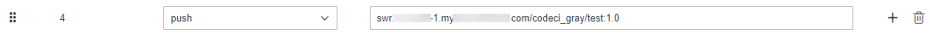
In this example, this command uses the Dockerfile with the tag **mydocker-registry.com/org/alpine:1.0** in the current directory to create an image.

- **docker push**: Push an image to a specified registry.

  Usage: docker push [options] name[:tag]

  Example: **docker push mydocker-registry.com/org/alpine:1.0**

  In this example, this command pushes tag 1.0 of the **mydocker-registry.com/org/alpine** image to the remote repository.



- **docker pull**: Pull an image from a registry.

  Usage: docker pull [options] name[:tag|@digest]

  The following table describes how to set **options**.

Option	Short Form	Description
--all-tags	-a	Download all tagged images.

  Example: **docker pull mydocker-registry.com/org/alpine:1.0**

  In this example, this command pulls the **mydocker-registry.com/org/alpine** image whose tag is **1.0** from the remote repository.

- **docker tag**: Modify the tag of the image.

  Usage: docker tag source_image[:tag] target_image[:tag]

  source_image[:tag] indicates the image whose tag needs to be modified, and **target_image[:tag]** indicates the target image with a new tag.

  Example: **docker tag mydocker-registry.com/org/alpine:1.0 mydocker-registry/neworg/alpine:2.0**

  In this example, this command changes the tag of the **mydocker-registry.com/org/alpine** image from **1.0** to **2.0**.

- **docker save**: Save one or more images to a **.tar** file (streamed to the standard output by default).

  Usage: docker save [options] image [image...]

  The following table describes how to set **options**.

Option	Short Form	Description
--output	-o	Write to a file instead of using standard output.

  Example: **docker save -o alpine.tar mydocker-registry.com/org/alpine:1.0 mydocker-registry.com/org/alpine:2.0**

  In this example, this command packages the **mydocker-registry.com/org/alpine:1.0** and **mydocker-registry.com/org/alpine:2.0** images into **alpine.tar**.

- **docker logout**: Log out of a Docker repository.

  Usage: docker logout [server]

  Example: **docker logout mydocker-registry.com**

  This example indicates that the image repository whose address is **mydocker-registry.com** is logged out.

5.3.25 Generating a Unit Test Report

The **Unit Test Report** action generates a visualized report by parsing the unit test result file that you have generated.

Prerequisites

Before running the **Unit Test Report** action, ensure that the test result file has been generated and the file framework is supported by CodeArts Build.

Build on GUI

Add **Unit Test Report**, when **configuring build actions**. Set the parameters according to **Table 5-61**.

Table 5-61 Parameters for generating the unit test report

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version that matches your current development environment. For tool versions supported by CodeArts Build, see **build tools and versions**. If the current tools and versions do not meet your requirements, you can **customize a build environment**.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.
Unit Test	• **Report Type**: Select a unit test framework. Currently, only JUnit is supported. • **Test Report**: Enter a relative path to specify the location relative to the project root directory, for example, **target/surefire-reports/TEST*.xml**. Currently, only the test report in the **.xml** format is supported. • **Print Unit Test Results**: Specify whether to process unit test coverage report. If you select **Yes**, ensure that your project can use **jacoco-maven-plugin** to generate unit coverage reports.

5.3.26 Building an Image and Pushing It to SWR

Upload the build artifact to SWR.

If necessary dependency packages and tools are missing, you can create a custom image from a Dockerfile and push the image to SWR. For details about how to use the pushed image, see **Using a Custom Build Environment**.

Preparations

- You have **created an organization** in SWR. For details about organization restrictions, see **notes and constraints** of SWR.
- If you want to push the created image to SWR of other Huawei Cloud users, perform the following operations.
  a. **Access the CodeArts Build Homepage** from the project list.
  b. In the navigation pane, choose **Settings** > **General** > **Service Endpoints**.
  c. Select **IAM user** from the **Create Endpoint** drop-down list box. In the displayed dialog box, enter the following information and click **Confirm**.
     - **Service Endpoint Name**: Assign a custom name to the endpoint. Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
     - Access key ID (AK) and secret access key (SK) are used like passwords to authenticate users who make API requests.

       On the CodeArts Build homepage, click **Console**, hover the cursor on the username in the upper right corner, and choose **My Credentials** from the drop-down list. In the navigation pane on the left, choose **Access Keys** to create a user key.
- If you want to push the created image to other image repositories, perform the following operations.
  a. **Access the CodeArts Build Homepage** from the project list.
  b. In the navigation pane, choose **Settings** > **General** > **Service Endpoints**.
  c. Select **Docker repository** from the **Create Endpoint** drop-down list box. In the displayed dialog box, enter the following information and click **Confirm**.
     - **Service Endpoint Name**: Assign a custom name to the endpoint. Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
     - **Repository Address**: Enter the address of the target image repository.
     - **Username**: Enter the username for logging in to the repository.
     - **Password**: Enter the password used for logging in to the repository.

Build on GUI

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

Retain the default values for the **Build with Maven** action. If the current parameter settings do not meet your requirements, modify the parameter settings by referring to **Building with Maven**. For details about the parameters for the **Build Image and Push to SWR** action, see **Table 5-62**.

Table 5-62 Parameters for creating an image and pushing it to SWR

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select the Docker version, or use the default one. Currently, CodeArts Build supports Docker 18.03 and Docker 20.10.
Image Repository	Select the target image repository. You can push the image to Huawei Cloud SWR and other image repositories.
Authorized User	Specify the user to which the target image repository belongs. You can push the image to the current user or other user's image repository. Ensure that you have permissions to edit or manage all images in the organization. For details, see **User Permissions**.
IAM Account	Expand the drop-down list and select the service endpoint created in **Preparations** for the specific IAM account. Then, use the service endpoint to push the files to the user's SWR. This parameter is mandatory when **Authorized User** is set to **Other**.
Push Region	Select the target region of your push. The built image will be pushed to the SWR repository in this region.
Docker Repository Endpoint	Select the **Docker repository** service endpoint created in **Preparations** and push the image to the corresponding repository through the service endpoint.
Organization	Select the organization created in **Preparations** from the drop-down list box. The image will be placed in this organization after being pushed to SWR.
Image Name	Enter the name of the created image. The value must start with a digit or letter and can contain 1 to 255 characters, including only lowercase letters, digits, underscores (_), and hyphens (-).
Image Tag	Specify the image tag, which can be customized. You can use *Image name*:*Tag* to uniquely specify an image. The value can contain 1 to 128 characters, including only letters, digits, periods (.), underscores (_), and hyphens (-). It cannot start with periods or hyphens.

Parameter	Description
Working Directory	Optional. The context path parameter in the **docker build** command is the relative path of the root directory of the repository. When Docker builds an image, the **docker build** command packs all content under the context path and sends it to the container engine to help build the image.
Dockerfile Path	Optional. Path of the Dockerfile. Set this parameter to a path relative to the working directory. For example, if the working directory is the root directory and the Dockerfile is in the root directory, set this parameter to **./Dockerfile**.
Add Build Metadata to Image	Specify whether to add the build information to the image. After the image is created, run the **docker inspect** command to view the image metadata.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

```
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
          ignore_fail: true
```


Table 5-63 Parameters in the sample code for creating an image and pushing it to SWR

Parameter	Type	Description
regions	List	Optional. Select the region of SWR where the image is to be uploaded to. By default, the image is uploaded to SWR in the region where the current task is located. If multiple regions are configured, the built image will be pushed to SWR in each region in sequence after the image is created.
organization	String	Enter the name of the organization to which the image belongs after being pushed to SWR. The organization name is the name of the organization created in **Preparations**.
image_name	String	Optional. Enter the name of the created image. The value must start with a digit or letter and can contain 1 to 255 characters, including only lowercase letters, digits, underscores (_), and hyphens (-). The default value is **demo**.
image_tag	String	Optional. Specify the image tag, which can be customized. You can use *Image name*:*Tag* to uniquely specify an image. The value can contain 1 to 128 characters, including only letters, digits, periods (.), underscores (_), and hyphens (-). It cannot start with periods or hyphens. The default value is **v1.1**.
context_path	String	Optional. The context path parameter in the **docker build** command is the relative path of the root directory of the repository. When Docker builds an image, the **docker build** command packs all content under the context path and sends it to the container engine to help build the image. The default value is **.**.
dockerfile_path	String	Optional. Path of the Dockerfile. Set this parameter to a path relative to the working directory. For example, if the working directory is the root directory and the Dockerfile is in the root directory, set this parameter to **./Dockerfile**. The default value is **./Dockerfile**.

Parameter	Type	Description
set_meta_data	Bool	Optional. Specify whether to add the build information to the image. After the image is created, run the **docker inspect** command to view the image metadata. • **true**: Add the build information to the image. • **false**: Do not add the build information to the image. The default value is **false**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.27 Using a Custom Build Environment

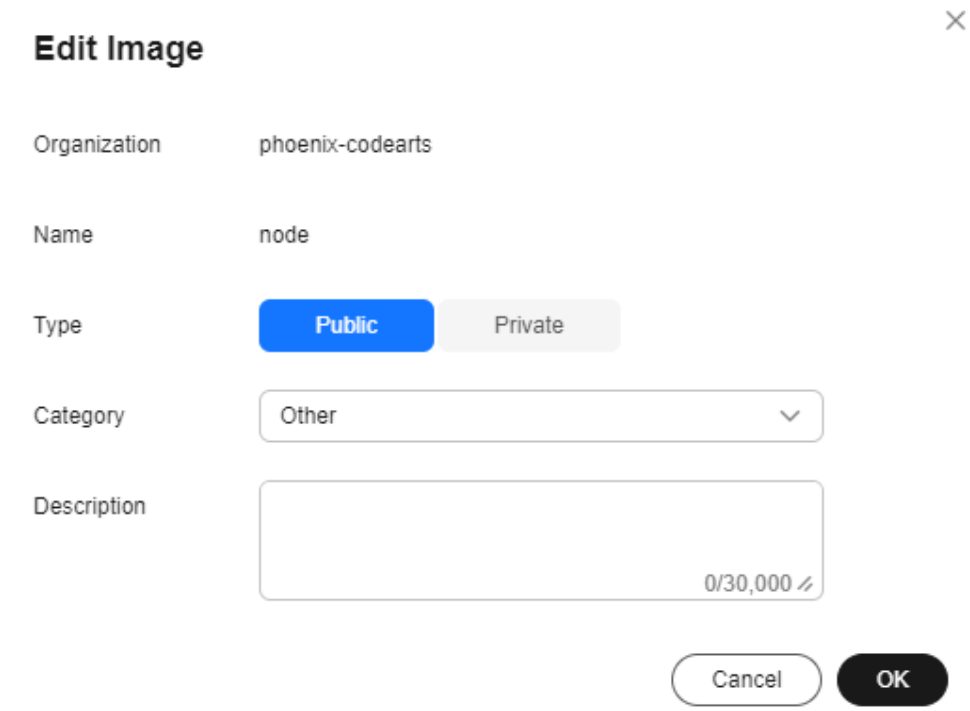
If the tool version supported by CodeArts Build does not meet your requirements, you can use a custom image that has been uploaded to SWR.

Setting the Image Type to Public

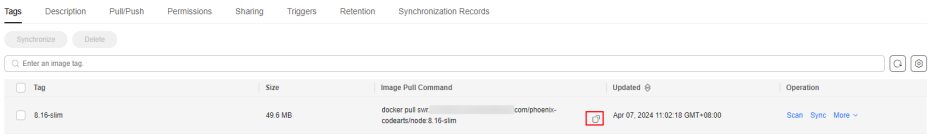
Private images in SWR cannot be pulled by CodeArts Build during the build process. Therefore, you need to set the image type to **Public** before starting the build.

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 displayed dialog box, set **Type** to **Public** and click **OK**.

Figure 5-6 Editing an image



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



Build on GUI

Add **Use SWR Public Image** when **configuring build actions**. Set the parameters according to **Table 5-64**.

Table 5-64 Parameters for using an SWR public image

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Image Address	Enter the image path obtained in **4**.

Parameter	Description
Commands	Configure commands, or use the default ones. If you have special build requirements, enter your custom build script in the text box. For example, if the image is used for a Maven build, configure commands for building with Maven. For an npm build, configure commands for building with npm. This rule also applies to other builds.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

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

Table 5-65 Parameters in the sample code for using an SWR image

Parameter	Type	Description
image	String	Set the image path in either of the following ways: • Use an address that starts with **cloudbuild** and uses the tag sign (@) as a separator, with the tool version supported by CodeArts Build following it. For example, **cloudbuild@maven3.5.3-jdk8-open**, where **maven3.5.3-jdk8-open** is the version of Maven being used. • Use the image path obtained in **4**.
command	String	Configure the command. For example, if the image is used for a Maven build, configure commands for building with Maven. For an npm build, configure commands for building with npm. This rule also applies to other builds.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.28 Downloading a Software Package from Release Repos

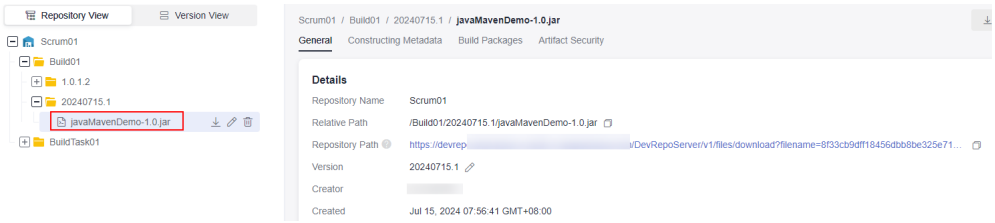
CodeArts Build allows you to download packages or files from the release repo to the build task root directory for use in subsequent build actions.

Obtaining the Package Download Address

Step 1 In the navigation pane, choose **Artifact** > **Release Repos**.

Step 2 Click the name of the package to be downloaded. On the package details page, the **Repository Path** is the download URL. Click next to the address to copy it.

Figure 5-7 Software package address



----End

Build on GUI

Add **Download Package from Release Repos** when **configuring build actions**. Set the parameters according to **Table 5-66**.

Table 5-66 Parameters for downloading a package from the release repo

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Tool Version	Select a tool version.
Package Address	Paste the address copied in **Step 2** to the text box.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:


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

Table 5-67 Parameters in the sample code

Parameter	Type	Description
url	String	Paste the address copied in **Step 2**.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.3.29 Uploading a Software Package to Release Repos

For details about the restrictions on uploading software packages, see **constraints** of CodeArts Artifact.

- Only 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 Repo** action. You can package the folder by running the packaging command or adding the **Run Shell Commands** action.

Build on GUI

Add **Upload to Release Repo**, when **configuring build actions**. Set the parameters according to **Table 5-68**.

NOTE

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

Table 5-68 Parameters for uploading a software package to the release repo

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Package Location	Directory for storing the build result. • 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 uses slashes (/) as separators, and the path is not case-sensitive. 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**.
Version	Optional. Set the name of the directory where the software package generated by the build task will be uploaded to the release repo. • Not specified (recommended): Use the build number to name the directory for storing files uploaded to release repos. • Specified: Files in the directory with the same name may be overwritten.
Package Name	Optional. Set the name for the software package generated by the build task. The name will be used when the package is uploaded to the release repo. • Not specified (recommended): Use the original file name to name the file uploaded to release repos. Leave **Package Name** unspecified so that all files matching the build package directory can be uploaded. • Specified: A file may be overwritten when another file with the same name is uploaded. For multiple file uploads with different package names, repeat the upload action for each file.

Parameter	Description
Custom Directory	Optional. After you specify the custom upload directory, the uploaded software package is uploaded to the *custom upload directory*/*version number*/*Software package name* directory.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

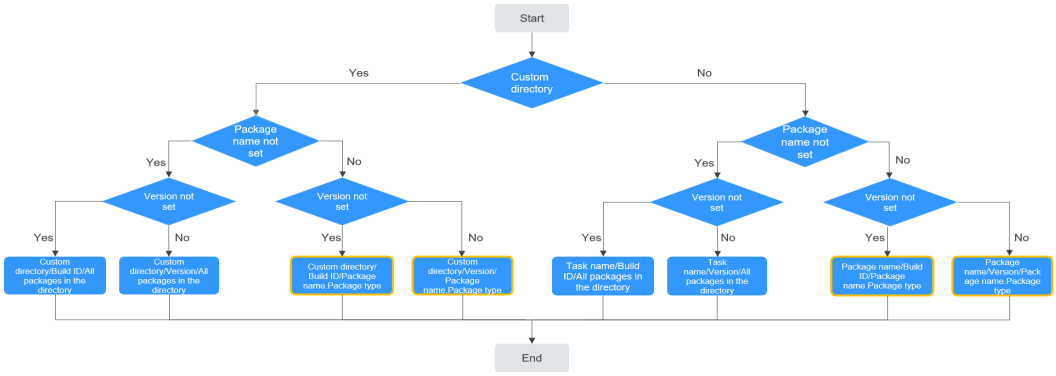
```
version: 2.0 # The value must be 2.0.
steps:
  BUILD:
    - upload_artifact:
          inputs:
            path: "**/target/*.?ar"
            version: 2.1
            name: packageName
            custom_upload_path: /phoenix-sample-ci/
            ignore_fail: true
```


Table 5-69 Parameters in the sample code

Parameter	Type	Description
path	String	Directory for storing the build result. ● 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 uses slashes (/) as separators, and the path is not case-sensitive. 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**.
version	String	Optional. Enter the release version number. Not specified (recommended): Use the build number to name the directory for storing files uploaded to release repos. Specified: Files in the directory with the same name may be overwritten.
name	String	Optional. Enter the name of the package generated during the build. Not specified (recommended): Use the original file name to name the file uploaded to release repos. Specified: A file may be overwritten when another file with the same name is uploaded.
custom_upload_path	String	Optional. After you specify the custom upload directory, the uploaded software package is uploaded to the *custom upload directory*/*version number*/*Software package name* directory.
ignore_fail	String	Whether to proceed after the current action fails. ● **true**: Yes ● Empty: No

How Release Versions and Package Names Impact Uploads

Figure 5-8 Impact of an unspecified release version and package name on uploads



5.3.30 Uploading Files to OBS

CodeArts Build allows you to upload build products to OBS. You can use this build action as required.

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

Preparations

To upload files to OBS of other users, perform the following operations.

1. **Access the CodeArts Build Homepage** from the project list.
2. In the navigation pane, choose **Settings** > **General** > **Service Endpoints**.
3. Select **IAM user** from the **Create Endpoint** drop-down list box. In the displayed dialog box, enter the following information and click **Confirm**.
   - **Service Endpoint Name**: Assign a custom name to the endpoint. Enter a maximum of 256 characters, including letters, digits, hyphens (-), underscores (_), periods (.), and spaces.
   - Access key ID (AK) and secret access key (SK) are used like passwords to authenticate users who make API requests.

     On the CodeArts Build homepage, click **Console**, hover the cursor on the username in the upper right corner, and choose **My Credentials** from the drop-down list. In the navigation pane on the left, choose **Access Keys** to create a user key.

Build on GUI

Add **Upload Files to OBS**, when **configuring build actions**. Set the parameters according to **Table 5-70**.

Table 5-70 Parameters for uploading files to OBS

Parameter	Description
Action Name	Assign a custom name to the build action. The name can contain: • Letters, digits, hyphens (-), underscores (_), commas (,), semicolons (;), colons (:), periods (.), slashes (/), and parentheses. • 1 to 128 characters.
Authorized User	Select the user. Your files will be pushed to the user's OBS. • **Current**: Upload files to an OBS bucket of the current user. • **Other**: Upload files to OBS of a specific user by specifying an IAM account.
IAM Account	Expand the drop-down list and select the service endpoint created in **Preparations** for the specific IAM account. Then, use the service endpoint to push the files to the user's OBS. This parameter is mandatory when **Authorized User** is set to **Other**.
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. 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**.
Bucket Name	Name of the target OBS bucket. (Cross-region upload is not supported.)
OBS Directory	Optional. 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	Optional. Enter the name (without the directory) that the resulting build file will be stored as in OBS. • If leave it as blank, you can upload multiple files and use the build product file name as the name it will be stored as in OBS. • If you do not leave it as blank, you can upload only one file, such as **application.jar**.

Parameter	Description
Upload Folder	You can choose whether to enable the function of uploading folders. • **Yes**: The folder is also uploaded. • **No**: The file is uploaded, but not the folder.
Headers	Optional. Add one or more custom response headers during the 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 by third-party web pages.
Continue After Failure	Specify whether to proceed after the current action fails by setting the parameter to either **Yes** or **No**.

Build with Code

Modify the code in the **BUILD** block in **Creating a YAML File for Your Code-based Build** by referring to the following sample code:

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


Table 5-71 Parameters in the sample code

Parameter	Type	Description
artifact_path	String	Optional. 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. 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**. The default value is **bin/***.
bucket_name	String	Name of the target OBS bucket. (Cross-region upload is not supported.)
obs_directory	String	Optional. 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. The default value is **./**.
artifact_dest_name	String	Optional. Enter the name (without the directory) that the resulting build file will be stored as in OBS. ● If leave it as blank, you can upload multiple files and use the build product file name as the name it will be stored as in OBS. ● If you do not leave it as blank, you can upload only one file, such as **application.jar**.
upload_directory	Bool	Optional. Specify whether to enable the function of uploading folders. ● **true**: The folder of the build product is also uploaded. ● **false**: All matched build products are uploaded to **obs_directory** in tile mode. The default value is **false**.

Parameter	Type	Description
headers	Map	Optional. Add one or more custom response headers during the file upload. The headers will be included in the response to download objects or query the object metadata. For example, you can set the value of **x-frame-options** to **false** to prevent web pages stored in OBS from being embedded into by third-party web pages.
ignore_fail	String	Whether to proceed after the current action fails. • **true**: Yes • Empty: No

5.4 Configuring Parameters

Predefined Parameters

The values of predefined parameters are automatically generated by the system and do not need to be defined, as shown in **Table 5-72**. You can use **${Parameter name}** to reference the parameters in the code.

Table 5-72 Predefined parameters

Parameter	Description
BUILDNUMBER	Build ID in the format of *date*.*times that this build task is run on that day*. For example: **20200312.3**.
TIMESTAMP	Build task running timestamp. For example: **20190219191621**.
INCREASENUM	Total number of times that the build task is run. The value starts from 1 and is incremented by 1 each time the task is run.
PROJECT_ID	ID of the project where the build task is located.
WORKSPACE	Root directory of the source code pulled by the build task, also known as the workspace.
GIT_TAG	Code tag name. This parameter only has a value if you have specified the **downloaded code** by tag.
COMMIT_ID_SHORTER	First eight digits of the code commit ID. This parameter only has a value if you have specified the **downloaded code** by commit ID.
COMMIT_ID	Code commit ID. For example: **b6192120acc67074990127864d3fecaf259b20f5**.

Adding Custom Parameters

On the page for configuring the build task, click the **Parameters** tab. On the displayed page, click **Create Parameter**, and set parameters according to **Table 5-73**.

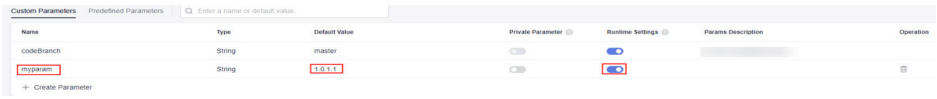
Table 5-73 Adding custom parameters

Name	Type	Default Value	Private Parameter	Runtime Settings	Params Description
Name of a custom parameter. The value can contain a maximum of 128 characters, including letters, digits, and underscores (_). **NOTE** • Do not use the following fields: **LD_PRELOAD**, **LD_LIBRARY_PATH**, **PATH**, **BASH_ENV**, **GIT_SSH_COMMAND**, and **path**. • Symbols are not supported.	String	Default value of the custom parameter. Max. 8,192 characters.	Specify whether the parameter is private or not. If the parameter is private, the system encrypts the input parameter for storage and only decrypts the parameter for usage. Private parameters are not displayed in run logs.	Specify whether to set this parameter when running the build task. If **Runtime Settings** is enabled, the parameter value can be changed when you click ▷ to run the build task, and the system reports the parameter to CodeArts Pipeline.	Enter additional information to describe the parameter. Max. 1,024 characters.
	Enumeration	In the displayed dialog box, enter enumerated values in the **Value** text box. Each parameter value must end with a semicolon (;). Max. 8,192 characters. Once you have set the enumerated values, select a default value for the parameter from the **Default Value** drop-down list box.			
	Auto Increment	Default value of the custom parameter. Max. 8,192 characters.			

Using Parameters

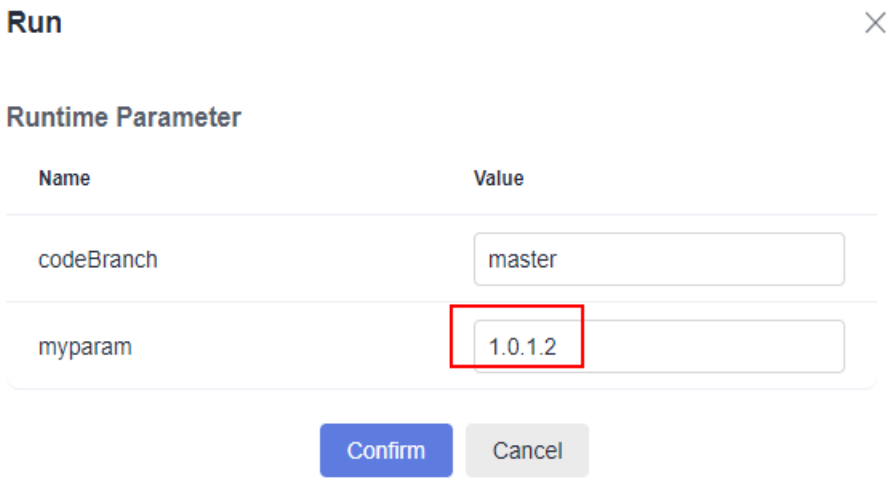
The following section describes how to use custom parameters, as shown in **Figure 5-9**.

Figure 5-9 Custom parameters



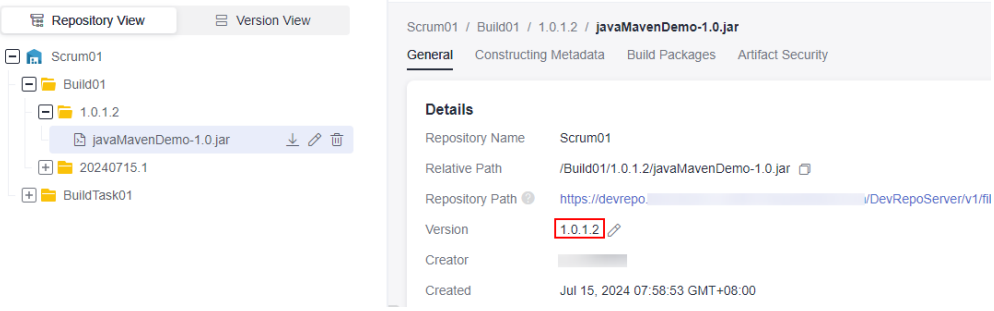
1. On the page for configuring the build task, click the **Build Actions** tab, enter **${myparam}** in the **Version** field of the **Upload to Release Repo** action, and click **Save and Execute**.
2. In the displayed dialog box, change the value of **myparam** to **1.0.1.2** and click **OK**. Wait until the build task is completed.

Figure 5-10 Setting the runtime parameters



3. Go to the release repo and find the resulting build package. The version number is the modified value of **myparam**.

Figure 5-11 Viewing the build package



5.5 Configuring Schedules

With CodeArts Build, you can configure triggers and schedule tasks, so developers can achieve continuous project integration.

Constraints

- The schedule setup is unavailable if the code source is set to **Pipeline**.

- Continuous integration is only supported when the code source is set to **Repo**.
- If a scheduled build task fails to run 10 times in a row, it will automatically stop being triggered further.

Scheduling a Build Task

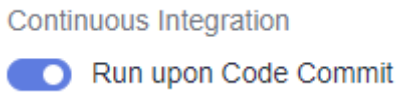
On the page for editing the build task, click the **Schedule** tab and configure an execution plan.

- **Continuous Integration**: Once **Run upon Code Commit** is enabled, committing the referenced code source will trigger a build task.

  NOTE

  This option can be enabled only when **Repo** is selected as the code source.

  Figure 5-12 Configuring continuous integration

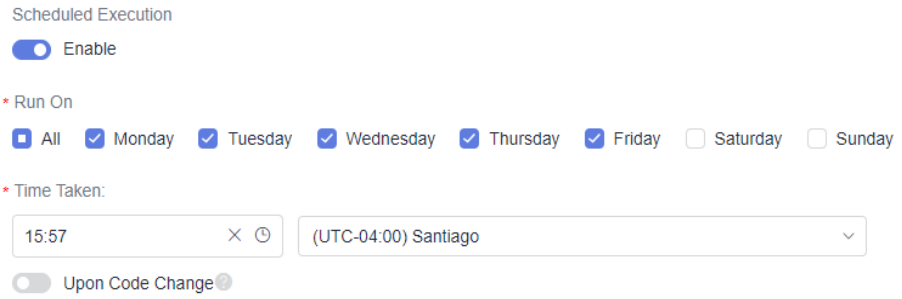
- **Scheduled Execution**: Enable this option and schedule the run time for the build task. Enable **Upon Code Change** if needed.

  After this function is enabled, the build task is run at the specified date and time.

  If you enable both **Scheduled Execution** and **Upon Code Change**, the build task will only run at the specified date and time if there have been changes to the code since the last build.

  NOTE

  If the scheduled build task fails to be run 10 consecutive times, it will not be triggered again.

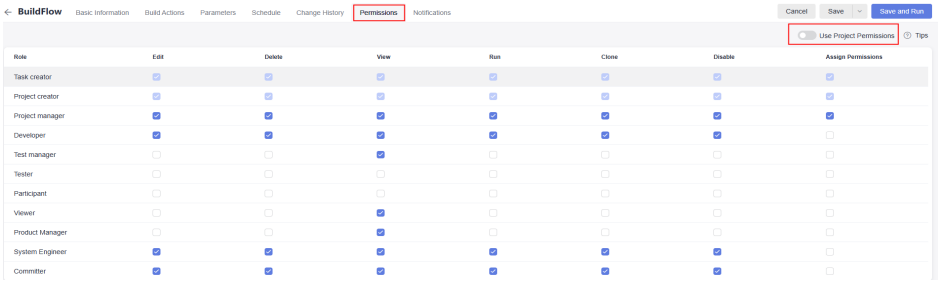
5.6 Configuring Roles and Permissions

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

1. On the page for editing the build task, click the **Permissions** tab page and configure operation permissions for different roles. For details about the default permissions of each role, see **Table 3-1**.

2. Toggle on **Use Project Permissions** to synchronize the current build task's permissions with the project's permissions. For details about how to configure project permissions, see **Configuring Role Permissions**.

Figure 5-13 Configuring roles and permissions for a build task



5.7 Configuring Notifications

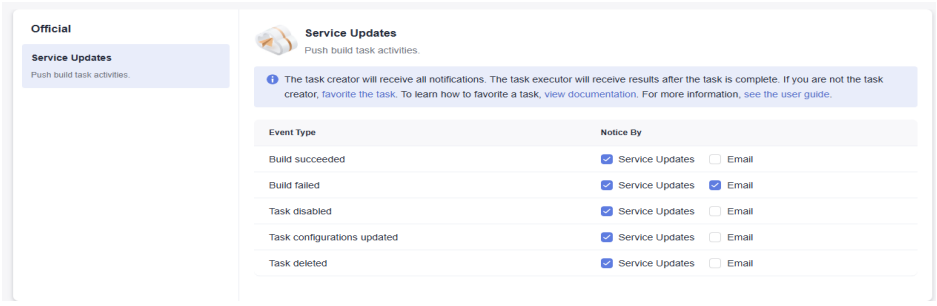
CodeArts Build can send you event notifications on your build task updates, including success, failure, disabling, changes, and deletion.

On the page for configuring a build task, click the **Notifications** tab and set the parameters.

Internal Messages and Emails

Click **Service Updates** under **Official**.

By default, you will receive internal messages for all events and emails for build failures. Adjust the notification settings as needed.



6 Running a Build Task

A build task can be triggered in the following ways:

- Run a single build task on the CodeArts Build page.
- Trigger task execution upon code commits to a CodeArts Repo repository. For details, see **Continuous Integration**.
- Execute the task on a scheduled basis. Alternatively, trigger task execution at specified time, but only if the code has changed since the last build. For details, see **Configuring Scheduled Execution**.
- Trigger a build task by running a **pipeline**.

This section describes how to run a single build task on the CodeArts Build page.

Prerequisites

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

Procedure

1. **Access the CodeArts Build Homepage** from the project list.
2. Search for the target build task on the CodeArts Build homepage and click  to run the task.

   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 and click **OK**.

   NOTE

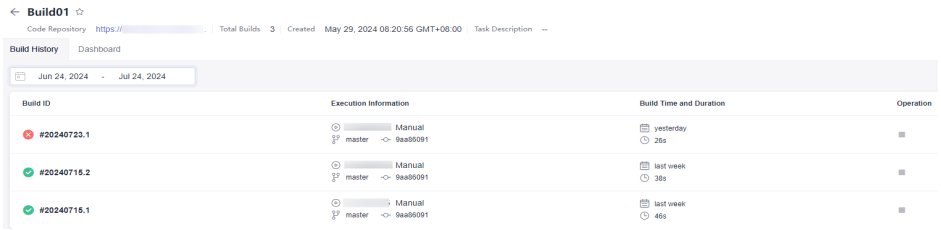
7 Viewing a Build Task

1. **Access the CodeArts Build Homepage** from the project list.
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.
Last Executed	Information such as the task executor, triggering mode, branch of the used repository, and commit ID.
Result	The most recent executions are shown from right to left in real time. You can tell the task status from the bar color (green: **Succeeded**, blue: **Running**, red: **Failed**, and gray: **Not built**).
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 **Managing Build Tasks**.

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 31 days are displayed by default. You can customize the period using the date selection component in the upper left corner of the page.)

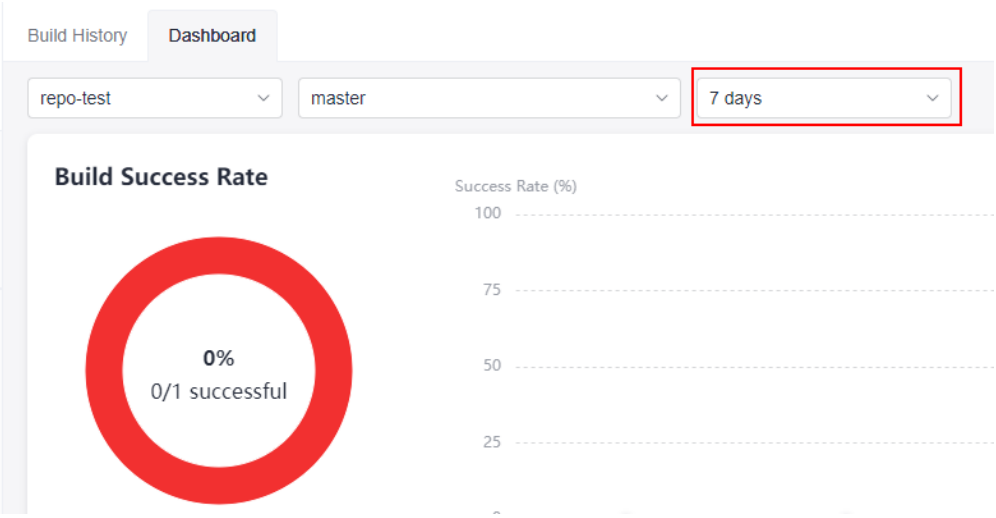
Figure 7-1 Build history



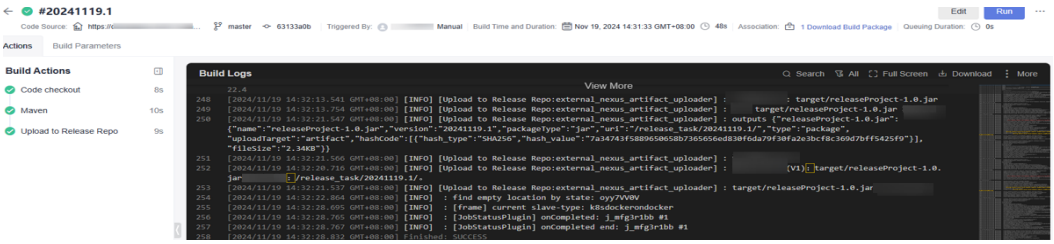
4. Click the **Dashboard** tab to view the build success rate and build performance distribution in the last seven days in pie, line, and bar charts.

   You can select a period from the drop-down list box.

Figure 7-2 Dashboard



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



   - Click the code source link in the upper left corner to access the code repository page.
   - Click **Download Build Package** and expand the drop-down list. To download all build packages, click **Download All**. To view all build packages, click **Go to Artifact** and go to the **Release Repos** page. To download a specified package, click the name of 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, choose **Download** > **Download Logs** to

download all log files, and click an action node on the left to view logs of the corresponding action.

- Click **Edit** or **Execute** 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.

8 Managing Build Tasks

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

Editing a Build Task

1. **Access the CodeArts Build Homepage** from the project list.
2. Search for the target build task.
3. In the row of the target build task, click ... and select **Edit** from the drop-down list.
   - On the **Basic Information** tab page, configure the task name, code source, code repository, default branch, and task description.
   - On the **Build Actions** tab page, configure build actions 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. Click **Compare Difference** to see what has been adjusted compared to the previous execution.
   - On the **Permissions** tab page, configure permissions for different roles.
   - On the **Notifications** tab page, configure notifications for different types of events (including **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

Click ... in the row that contains the target build task and select **Delete** from the drop-down list. Exercise caution when performing this operation.

You can view the deleted build task in the recycle bin. In the upper right corner of the CodeArts Build homepage, click **More** and select **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 .
Delete a task	Select the task to be deleted from the list and click **Delete** to delete the task from the recycle bin.
Restore a task	Select the task to be restored from the list and click **Restore**. Then you can find this task again in the task list of CodeArts Build.
Clear the recycle bin	Click **Empty Recycle Bin** to delete all tasks from the recycle bin.

Cloning the Build Task

1. Click ••• in the row of the build task and select **Clone** from the drop-down list.
2. On the displayed page, modify the task information as required and click **Save** to create a clone of the build task.

   If you want to both clone the build task and run the clone, click **Save and Execute**.

NOTE

Cloning a task will duplicate all of its permissions. The new task has identical access control settings as the original.

Disabling a Task

- The build task that is currently running 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 select **Enable** from the drop-down list.

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

Favoriting the Build Task

- 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.

1. Move the cursor to the row of the build task and click ☆. If the color of the icon changes, the task is successfully favorited.
2. (Optional) Click ★ to unfavorite the task.

Stopping a Build Task

1. Click the name of a running build task. The **Build History** page is displayed.
2. Click the **Build ID**.
3. On the displayed page, click **Stop** in the upper right corner.

9 Querying Audit Logs

Cloud Trace Service (CTS) records operations on CodeArts Build for query, audit, and backtrack.

Operations Recorded by CTS

Table 9-1 CodeArts Build operations recorded by CTS

Operation	Resource Type	Event
Creating a build task	BuildProjectService	createJob
Running a build task	BuildProjectService	buildJob
Deleting a build task	BuildProjectService	deleteJob
Updating a build task	BuildProjectService	updateJob
Disabling a build task	BuildProjectService	disableJob
Enabling a build task	BuildProjectService	enableJob
Uploading a keystore file	BuildProjectService	uploadKeystore
Updating a keystore file	BuildProjectService	updateKeystore
Deleting a keystore file	BuildProjectService	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	BuildProjectService	createCustomTemplate

Operation	Resource Type	Event
Deleting a custom template	BuildProjectService	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

Viewing Audit Logs

Query CodeArts Build traces on the CTS console. For details, see **Viewing Audit Logs**.

10 Old User Guide

10.1 Signing Android APK

Sign an APK with apksigner.

Build on GUI

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	Assign a custom name to the build action.
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.
Keystore File	Used for signing. You can create the file by referring to **Generating Keystore Signature Files**. Select one from those uploaded on the **file management** page.
Keystore Password	Keystore password.
Alias	Alias of the keystore file.
Key Password	Password of the key.
apksigner CLI	Custom signature parameter. By default, **--verbose** is added to display the signature details.

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.

Build with Code

```
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	Directory of the APK that needed to be signed	Yes	None
keystore_file	String	Keystore file name	Yes	None
keystore_password	String	Keystore file password	No	None
alias	String	Alias	Yes	None
key_password	String	Password	No	None
apksigner_commond	String	apksigner command	Yes	None

10.2 Downloading File from File Manager

CodeArts Build stores your Android APK signature files and **settings.xml** files of Maven builds, and helps you manage 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 **Managing Files**. Add the **Download File from File Manager** action to download files from **Files** to the working directory for use.

Build on GUI

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

The parameters are described in the following table.

Parameter	Description
Action Name	Assign a custom name to the build action.
Tool Version	Select a tool version.

Parameter	Description
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.

Build with Code

```
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

10.3 Managing Files

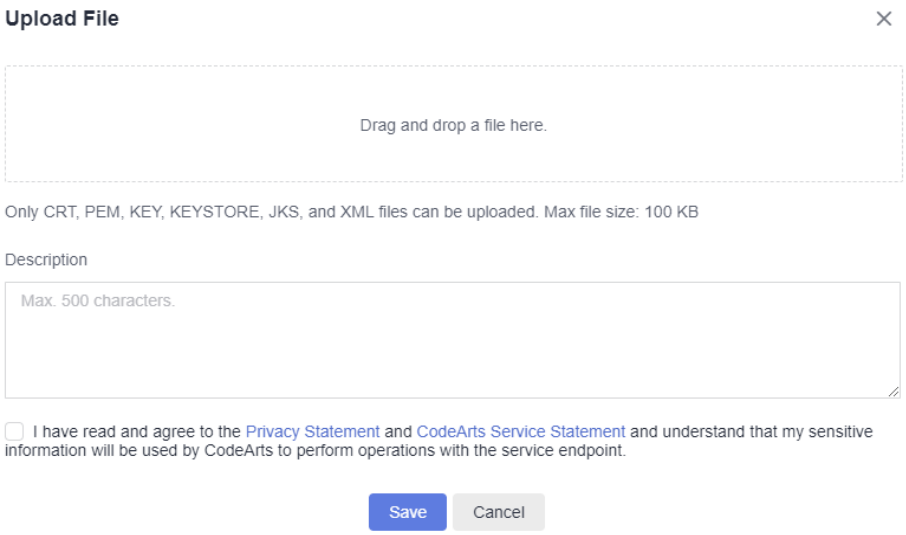
CodeArts Build stores your **Android APK signature files** and **settings.xml** files of **Maven builds**, and helps you manage these files. For example, you can create, edit, and delete these files, and modify users' permissions on them.

Constraints

- The maximum file size is 100 KB.
- The file type must be **.xml**, **.key**, **.keystore**, **.jks**, **.crt**, or **.pem**.
- A maximum of 20 files can be uploaded.

Uploading a File

1. **Access the CodeArts Build homepage**.
2. Click **More** and select **Files**.
3. Click **Upload File**.
4. In the displayed dialog box, select a file, add a description, select the check box to agree to the statements, and click **Save**.



Managing Files

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

- Enter a keyword in the search box to search for a file.
- 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 select **Delete** from the drop-down list. Confirm the deletion as prompted.
- Click in the **Operation** column and select **Modify Permissions** from the drop-down list. In the displayed dialog box, configure file operation permissions for the user.

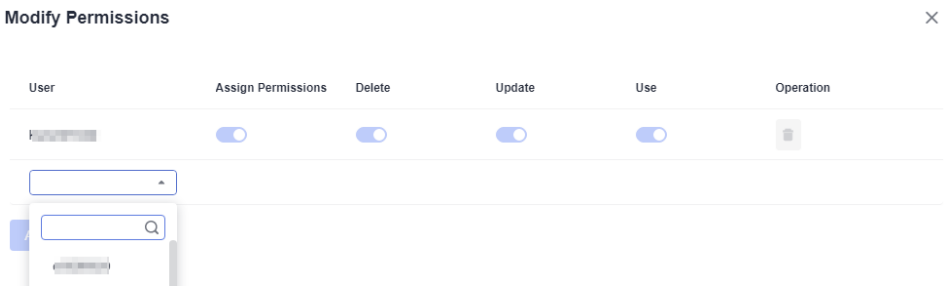


Table 10-1 Roles and their permissions on files

Permission	Role with the Permission
Add users	All users in the project
View a file	File creator and users under the same account

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

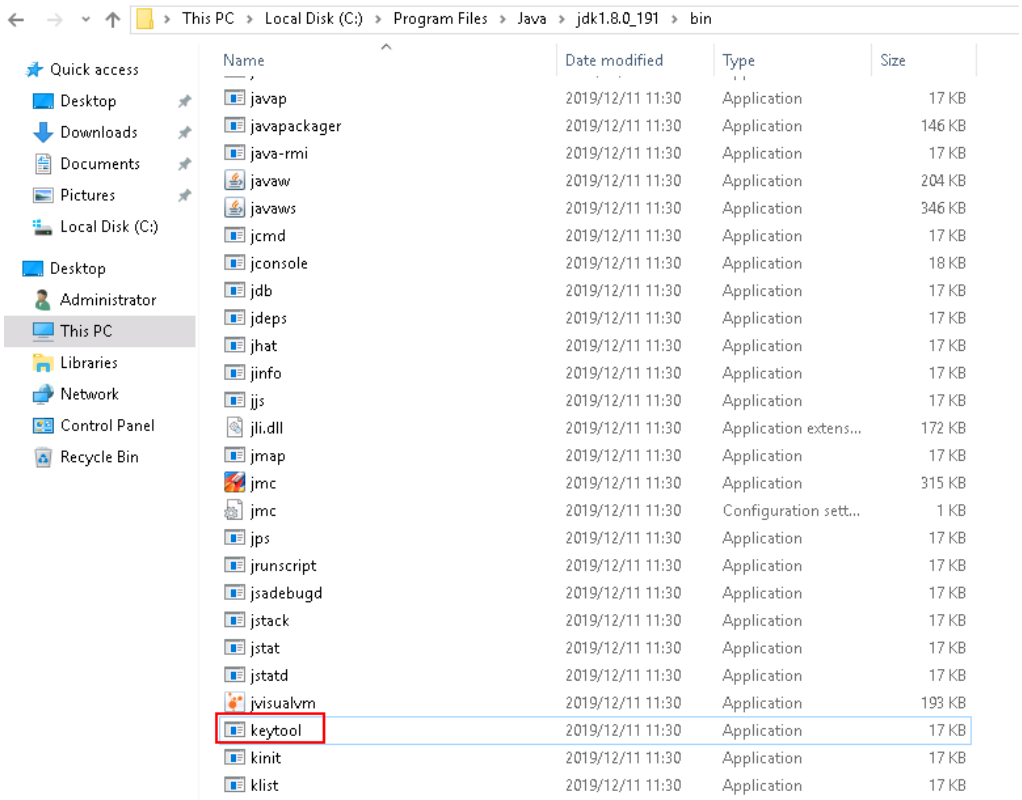
NOTE

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

Generating Keystore Signature Files

- **Using Keytool in JDK to Generate Signature Files**

  a. Find the JDK installation path and run **keytool.exe**.

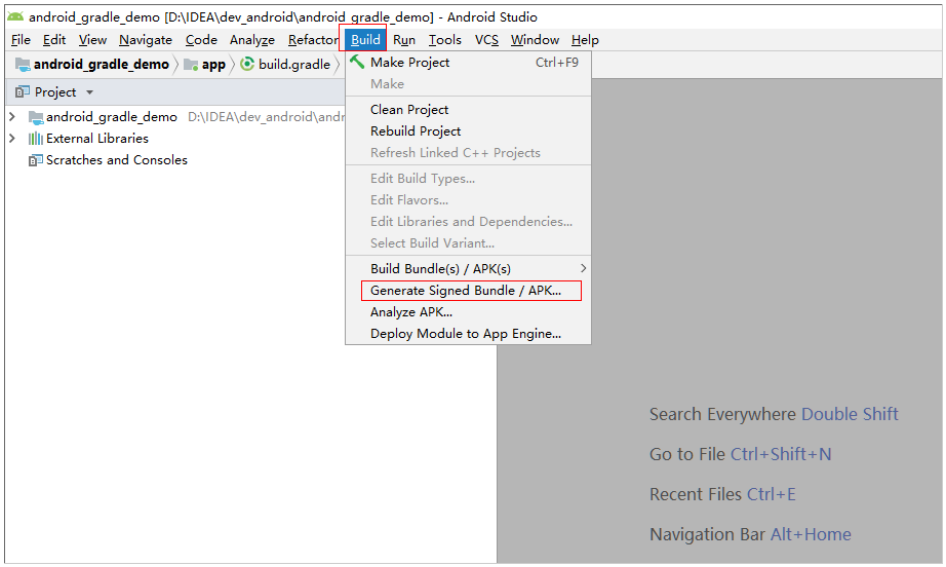


  b. Run the following command to generate a **.jks** file:

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

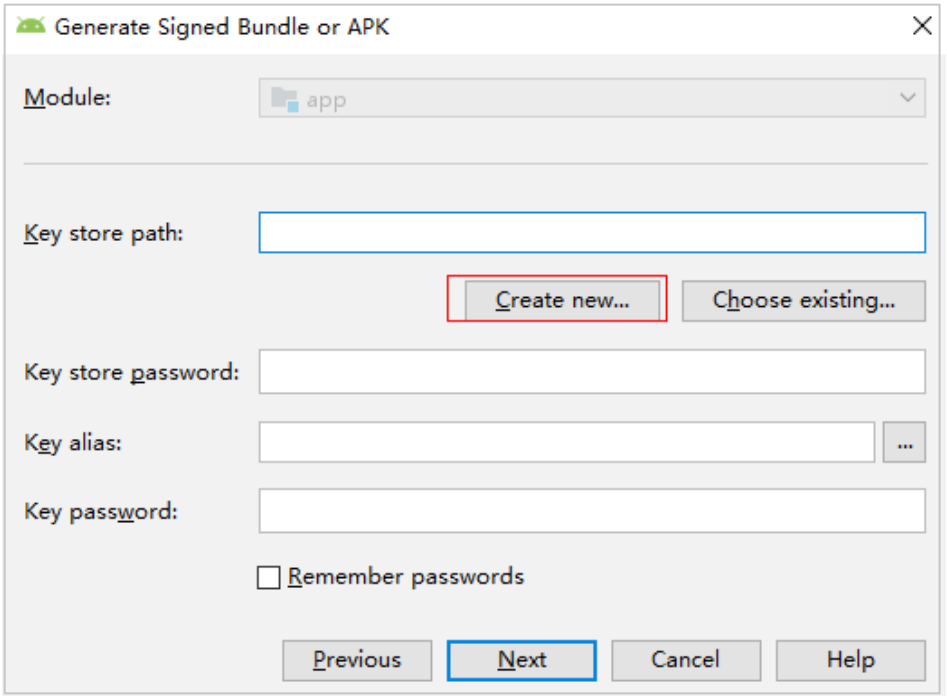
- **Using Android Studio to Generate Signature Files**

a. Open Android Studio and choose **Build** > **Generate Signed Bundle/APK**.

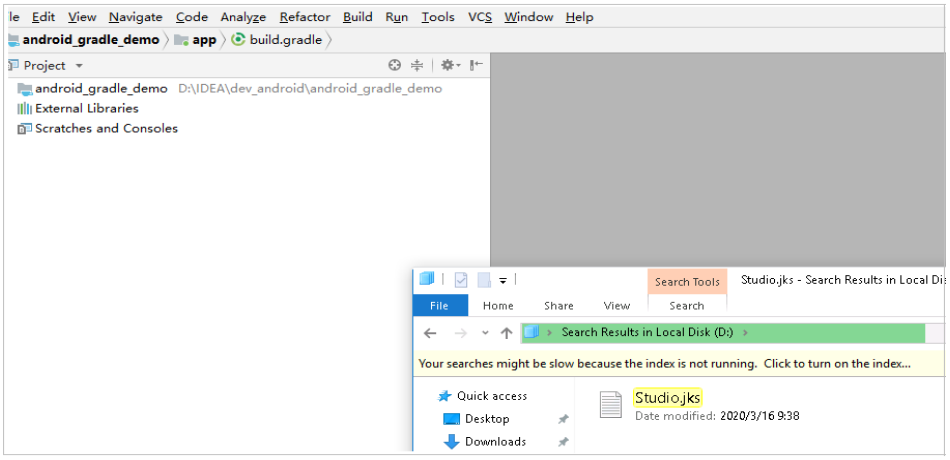


b. Select **APK** and click **Next**.

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



d. View the generated signature file.

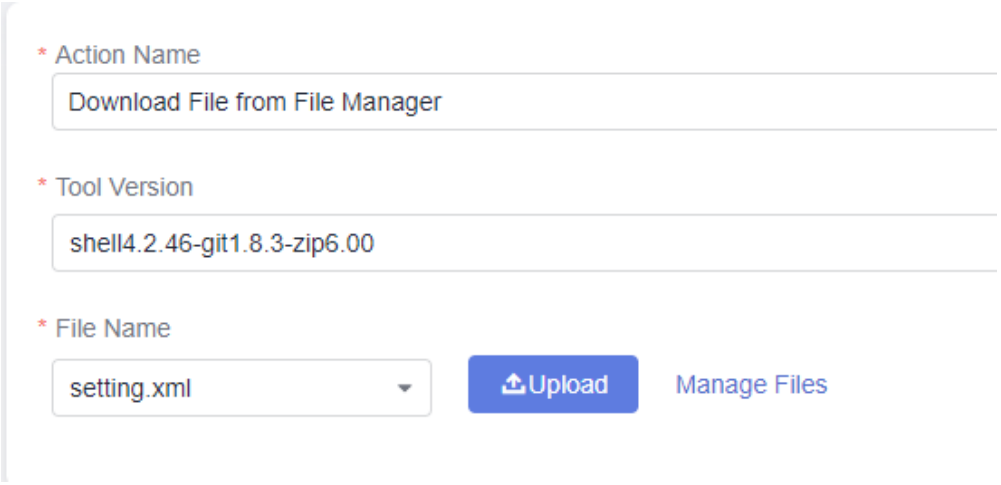


NOTE

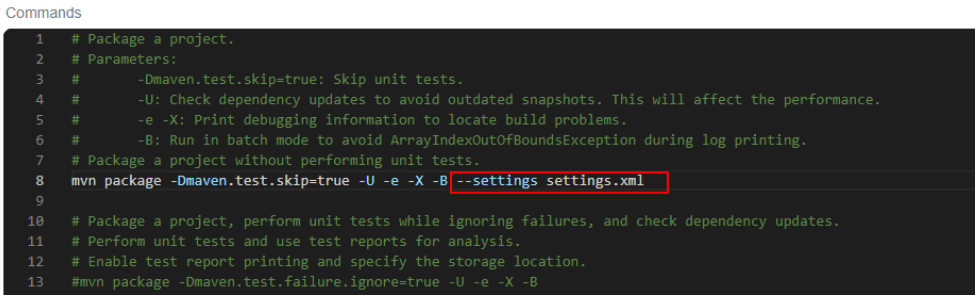
You can upload the generated signature file to **Files** for unified management.

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.



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



10.4 Custom Build Environments

Background

If the common build environments do not meet your requirements, **customize one** by adding required dependencies and tools to a base image to make a Dockerfile. Then you can **use the custom environment** for your build.

Base Images

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:

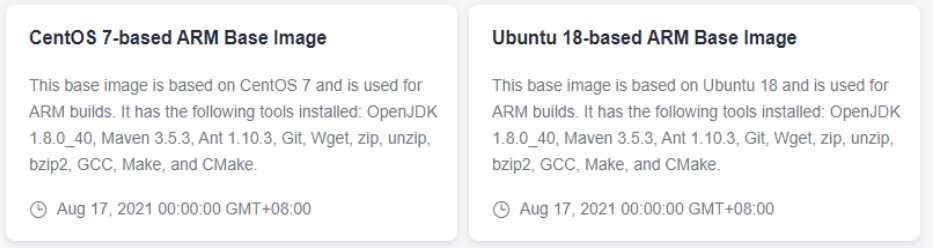
JDK 1.8, Maven, Git, Ant, zip, unzip, GCC, CMake, and Make.

Procedure

Step 1 **Access the CodeArts Build homepage**.

Step 2 In the upper right corner of the CodeArts Build homepage, click **More** and select **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.



Step 4 Edit the downloaded Dockerfile.

You can add other dependencies and tools required by the project 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

10.5 Custom Templates

Build template selection: If the preset build templates cannot meet your 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** page is displayed.

NOTE

If no task exists in the list, **create a build task on the GUI**.

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

NOTE

A build task that uses any private parameters cannot be saved as a template. For details about how to set build parameters, see **Configuring Parameters**.

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 in the list.

You can perform the following operations on saved templates.

Table 10-2 Managing custom templates

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

----End

10.6 Editing, Deleting, Cloning, Favoriting, and Stopping 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 select **Edit** from the drop-down list.
   - On the **Basic Information** tab page, configure the task name, code source, code repository, branch, and task description.
   - On the **Build Actions** tab page, configure build actions 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 notifications for different types of events (including **Build succeeded**, **Build failed**, **Task deleted**, **Task configurations updated**, and **Task disabled**).

4. Edit the information on a tab page, and click **Save and Execute** > **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 select **Clone** from the drop-down list.
3. On the displayed page, modify the task information as required and click **Clone**.

NOTE

Cloning a task will duplicate all of its permissions. The new task has identical access control settings as the original.

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 favorited.
3. (Optional) Click ★ to unfavorite 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.

Stopping a Build Task

1. Search for the target build task.

2. Click the name of a running build task. The **Build History** page is displayed.
3. Click the **Build ID**.
4. On the displayed page, click **Stop** in the upper right corner.

