Android Continuous Integration and Delivery

With Jenkins + Git + Gradle

Zilong Wang

A01001507

COMP 7082

BCIT

This document describes how to build a Jenkins environment and use it to build and release android application.

# Step 1 Install Jenkins

Before install Jenkins, the compute should have

* 256 MB of RAM, although more than 512MB is recommended
* 10 GB of drive space
* Java 8 or higher (either a JRE or Java Development Kit (JDK) is fine

1. Go Jenkins’s official website <https://jenkins.io/download/>, download LTS (Long-Term Support) releases for your operating system
2. Install Jenkins, for Windows, Run Jekins.msi to install.
3. Go <https://git-scm.com/downloads> to download and install Git.

After installing, Jenkins will be a Windows service. It can be start/restart/stop in Windows service management. See Figure 1:

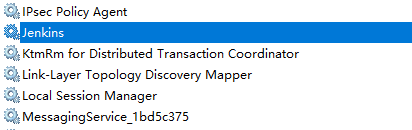
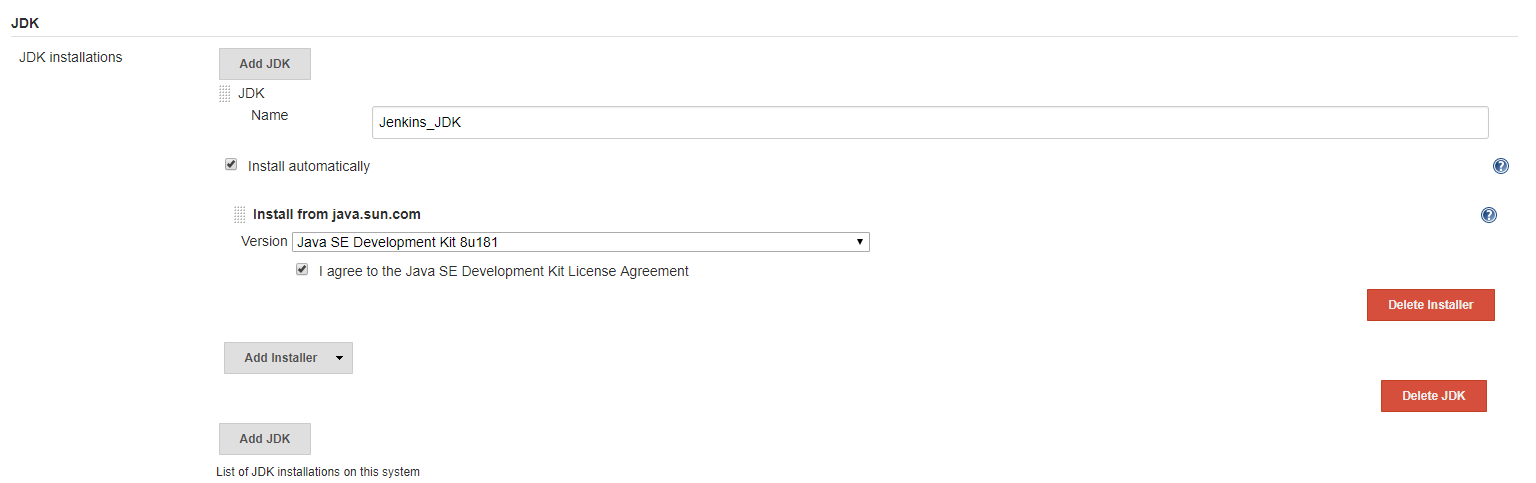


Figure 1 Windows service management

# Step 2 Set up Jenkins Plugins

1. Open http://localhost:8080 in browser
2. Select “Manage Jenkins” from left Menu
3. Select “Global Tool Configuration” from right Menu
4. Set up JDK, Git, Gradle as following figures, if there is no such option, go “Manage Jenkins>Manage plugins” to install them

  
Figure 2 JDK Setting

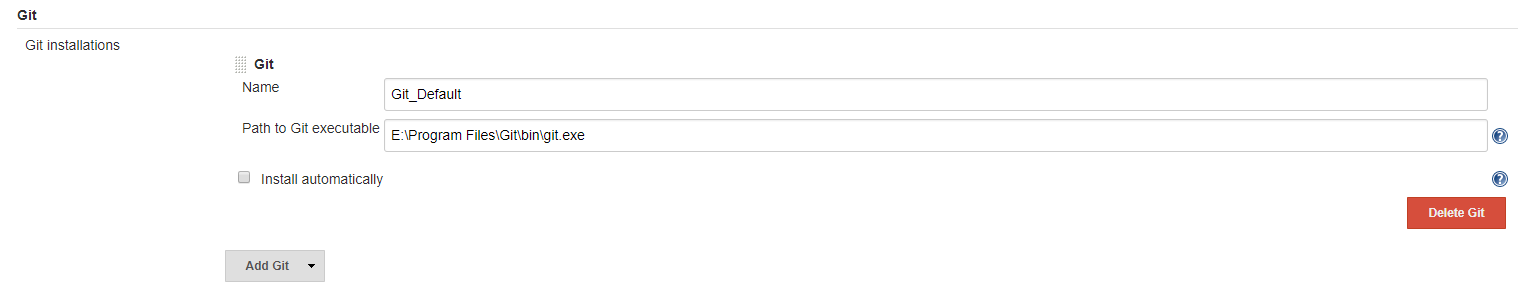


Figure 3 Git Setting

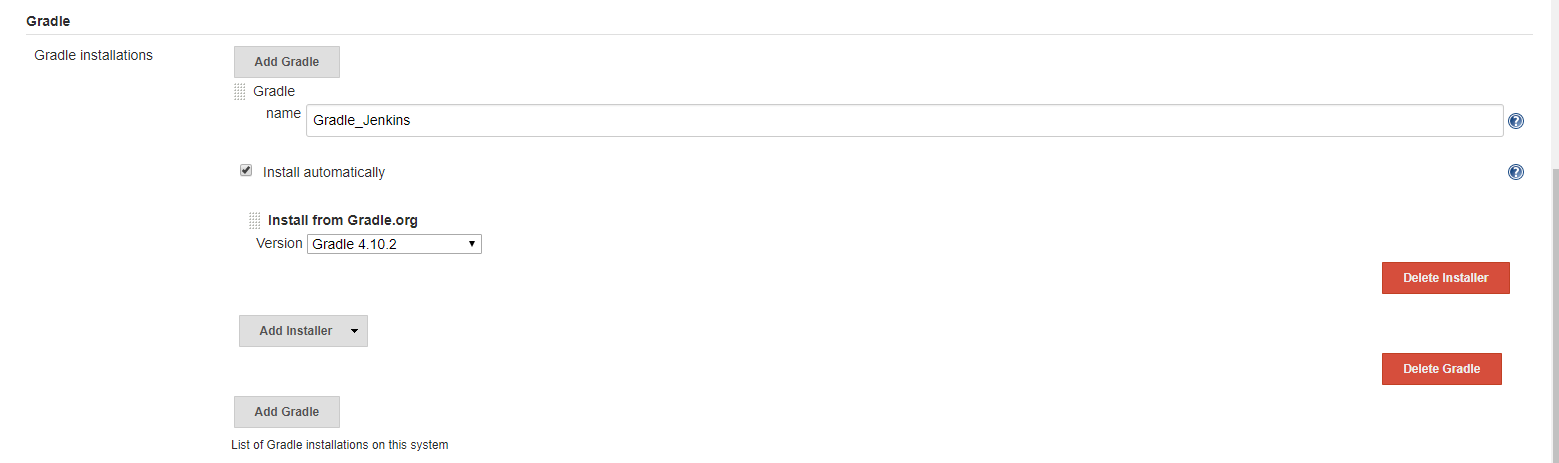


Figure 4 Gradle Setting

# Step 3 Create Project Item

1. Go “New Item”, input item name and select Freestyle project

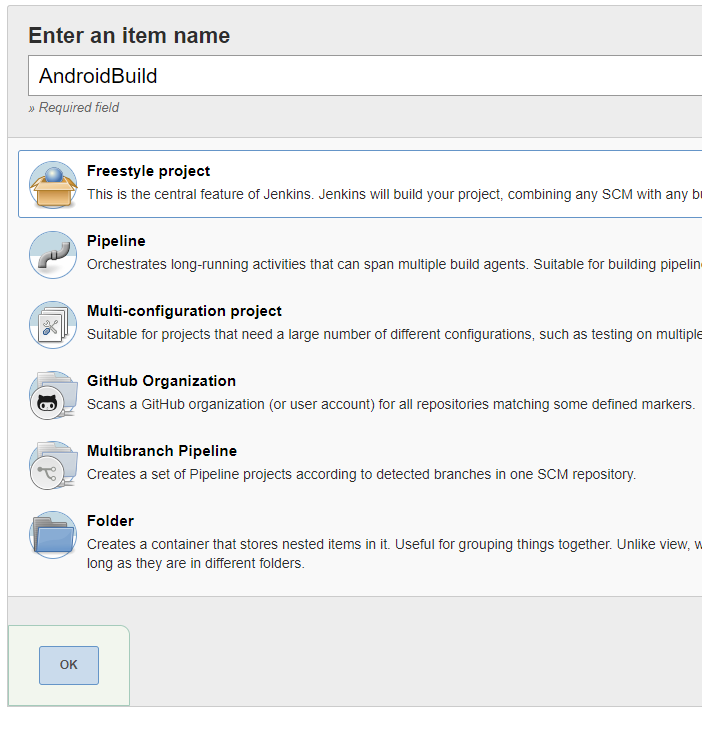


Figure 5 Add item

2. Set up Source Code Management by setting GitHub clone URL and credential

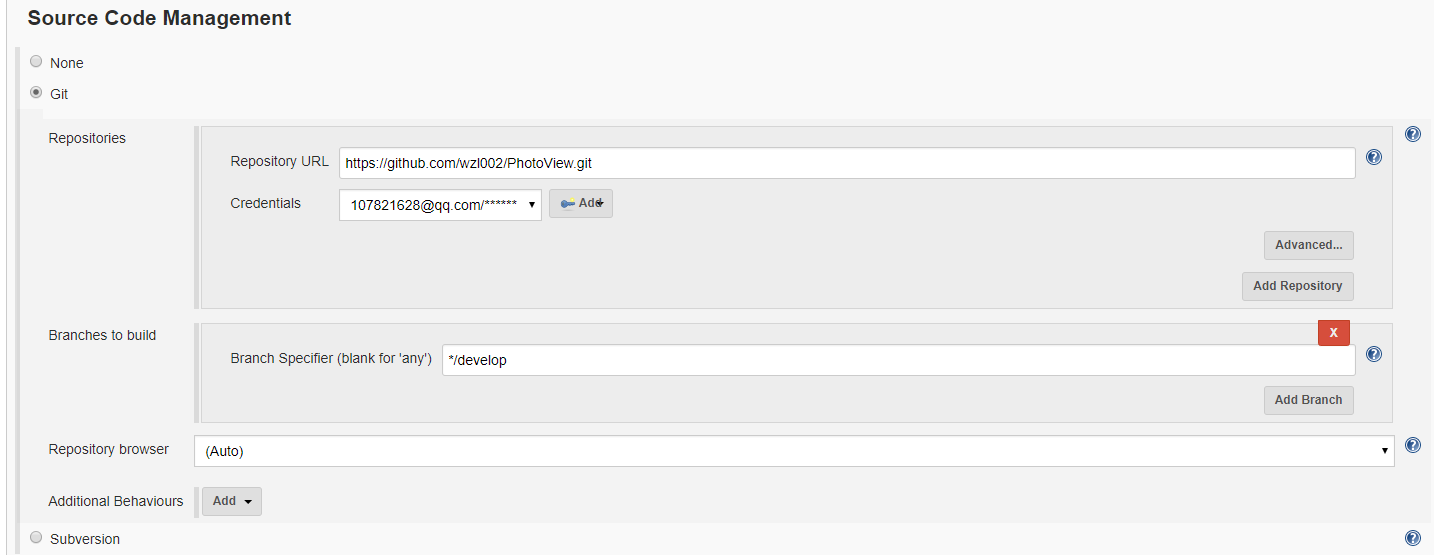


Figure 6 Config Item Git

3. Set up Build task with Gradle wrapper

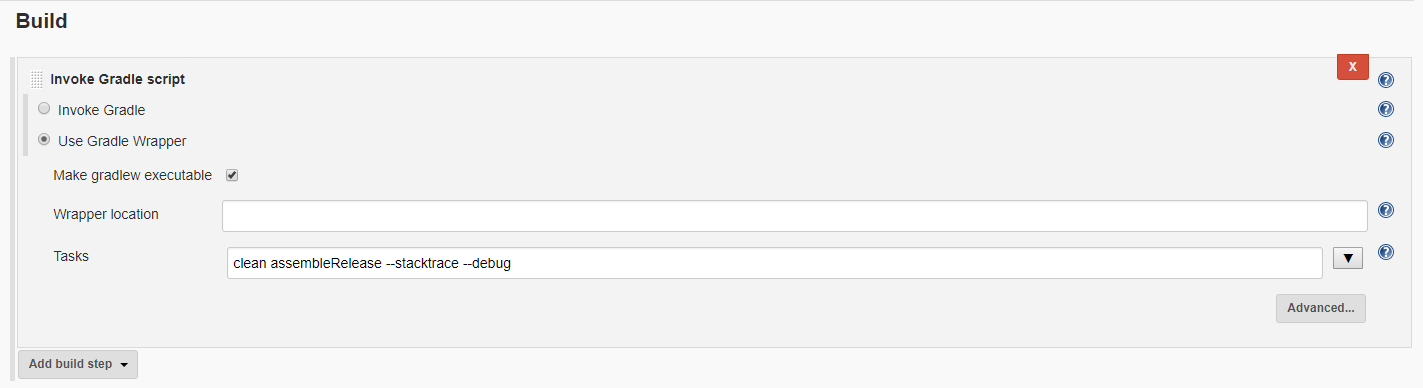
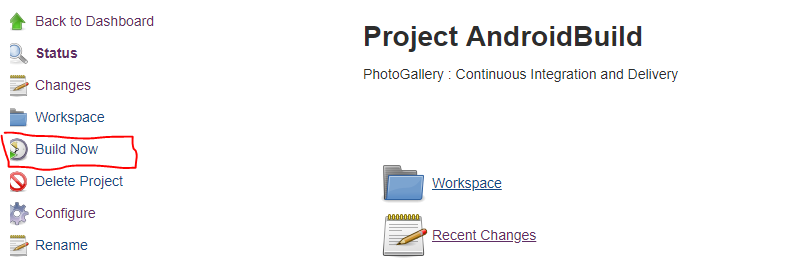


Figure 7 Config Item Gradle

# Step 4 Build Project

Go item page, select “Build Now”



So far, the software setup is complete. By create different item and set different Gradle tasks, we can use Jenkins to execute various processes. Here are some examples of tasks:

## Tasks: Use Jenkins for a debug/release build

# Build debug package: set the Gradle tasks in Items as following:

clean assembleDebug

# Build release package: set the Gradle tasks in Items as following:

clean assembleRelease

# Publish App to Google Play

Jenkins can also be used to publish android application to Google Play. To do that, we need to install a Jenkins Plugin: “**Google Play Android Publisher Plugin**”

The following steps for setting up the publisher plugin comes from its [wiki tutorial](https://wiki.jenkins.io/display/JENKINS/Google+Play+Android+Publisher+Plugin):

## Install plugin

Install this plugin via the Jenkins plugin manager.  
Or if installing the plugin via other means, ensure that the prerequisite [Google OAuth Plugin](https://wiki.jenkins.io/display/JENKINS/Google+OAuth+Plugin), [Token Macro Plugin](https://wiki.jenkins.io/display/JENKINS/Token+Macro+Plugin) and their dependencies are also installed.

## Create Google service account

To enable automated access to your Google Play account, you must create a service account:

1. Sign in to the [Google Play developer console](https://play.google.com/apps/publish/) as the account owner
2. Select Settings → Developer account → API access
3. Under Service Accounts, click "Create Service Account"
4. Follow the link to the Google API Console
5. Click the "Create service account" button
6. Give the service account any name you like, e.g. "Jenkins"
7. Choose Service Accounts > Service Account User for the "Role" field
8. Enable "Furnish a new private key"
9. Choose "JSON" as the key type (P12 works as well, but JSON is a little simpler)
10. Click the "Save" button
11. Note that a .json file is downloaded, named something like "api-xxxxxxxxx-xxxxx-xxxx.json"
12. Close the dialog that appears
13. Copy the email address of the new user (something like "jenkins@api-xxxxxxxxx-xxxxx-xxxx.iam.gserviceaccount.com")
14. You can now close the page

## Assign permissions to the service account

1. Return to the Google Play developer console page
2. Click "Done" on the dialog
3. Note that the service account has associated with the Google Play publisher account
   1. If it hasn't, follow these additional steps before continuing:
   2. Click "Users & permissions" in the menu
   3. Click "Invite new user"
   4. Paste in the email address you copied above
   5. Continue from step 5
4. Click the "Grant access" button for the account (e.g. "jenkins@api-xxxxxxxxx-xxxxx-xxxx.iam.gserviceaccount.com")
5. Ensure that at least the following permissions are enabled:
   1. **View app information** — this is always required
   2. **Manage production releases** — if you want to upload APKs to production, or a staged rollout
   3. **Manage testing track releases** — if you want to upload APKs to alpha, beta, or internal
6. Click "Add user" (or "Send invitation", as appropriate)
7. You can now log out of the Google Play publisher account

## Add the service account credentials to Jenkins:

1. Navigate to your Jenkins instance
2. Select "Credentials" from the Jenkins sidebar
3. Choose a credentials domain and click "Add Credentials"
4. From the "Kind" drop-down, choose "Google Service Account from private key"
5. Enter a name for the credential — the actual value is not important
6. Choose the "JSON key" type
7. Upload the .json file that was downloaded by the Google API Console
8. Click "OK" to create the credential

Jenkins now has the required credentials and permissions in order to publish to Google Play.

Once you've set up a job (see the next section) and confirmed that uploading works, either delete the downloaded JSON file or ensure that it's stored somewhere secure.

## Freestyle job configuration

**Uploading an APK**

1. Create a new free-style software project
2. Ensure, via whatever build steps you need, that the APK(s) you want to upload will be available in the build's workspace
3. Add the "Upload Android APK to Google Play" post-build action
4. Select the credential name from the drop-down list
5. The credential must belong to the Google Play account which owns the app to be uploaded
6. Enter paths and/or wildcards pointing to the APK or APKs to be uploaded
7. This can be an Ant-style \*\*/\*-release.apk pattern, or a comma-separated list of filenames, relative to the root of the workspace
8. Choose the track to which the APKs should be deployed
9. If you're deploying a production release, you can choose a [rollout percentage](https://support.google.com/googleplay/android-developer/answer/3131213)
10. Optionally choose "Add language" to associate release notes with the uploaded APK(s)
11. You add entries for as many or as few of your supported language as you wish, but each language must already have been added to your app, under the "Store Listing" section in the Google Play Developer Console.

**APK expansion files**

You can optionally add up to two [expansion files](https://developer.android.com/google/play/expansion-files.html) for each APK being uploaded.

A list of expansion files can be specified in the same way as APKs, though note that they must be named in the format [main|patch].<expansion-version>.<package-name>.obb.

See the inline help for more details.

**Moving existing APK(s) to another release track**

If you have already uploaded an app to the alpha track (for example), you can later use Jenkins to re-assign that version to the beta or production release track.  
Under the "Build" section of the job configuration, add the "Move Android APKs to a different release track" build step and configure the new release track.

You can tell Jenkins **which** APKs to be moved by either entering the APK version codes directly, or by providing the APK files, from which the plugin will read the application ID and version codes for you.

## Pipeline job configuration

As of version 1.5, this plugin supports the [Pipeline Plugin](https://wiki.jenkins.io/display/JENKINS/Pipeline+Plugin) syntax. You can generate the required Pipeline syntax via the [Snippet Generator](https://jenkins.io/blog/2016/05/31/pipeline-snippetizer/), but some examples follow.

Note that you should avoid using these steps in a parallel block, as the Google Play API only allows one concurrent "edit session" to be open at a time.

**Uploading an APK**

The androidApkUpload step requires at least the Google Play credential ID, a list of APK(s) to upload, and the track to assign them to:

|  |
| --- |
| androidApkUpload googleCredentialsId: 'My Google Play account', apkFilesPattern:  '\*\*/\*.apk', trackName: 'production' |

Uploading the ProGuard mapping file(s) to be associated with the APK(s) just requires one more more patterns in addition:

|  |
| --- |
| androidApkUpload googleCredentialsId: 'My Google Play account', apkFilesPattern:  '\*\*/\*.apk',  deobfuscationFilesPattern: '\*\*/[mapping.txt'](https://wiki.jenkins.io/display/JENKINS/mapping.txt'), trackName: 'production' |

Performing a staged rollout requires the rollout percentage as a string; the percentage sign is optional:

|  |
| --- |
| androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '\*\*/\*.apk',  trackName: 'production', rolloutPercentage: '50%' |

Adding "recent changes" text requires specifying a list:

|  |
| --- |
| androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '\*\*/\*.apk',  trackName: 'alpha',  recentChangeList: [  [language: 'en-GB', text: "Please test the changes from Jenkins build  ${[env.BUILD\_NUMBER](https://wiki.jenkins.io/display/JENKINS/env.BUILD_NUMBER)}."],  [language: 'de-DE', text: "Bitte die Änderungen vom Jenkins Build  ${[env.BUILD\_NUMBER](https://wiki.jenkins.io/display/JENKINS/env.BUILD_NUMBER)} testen."]  ] |

To upload expansion files, reusing those from the previous upload where possible:

|  |
| --- |
| androidApkUpload googleCredentialsId: 'GP', apkFilesPattern: '\*\*/\*.apk',  trackName: 'production',  expansionFilesPattern: '\*\*/[patch.obb'](https://wiki.jenkins.io/display/JENKINS/patch.obb'),  usePreviousExpansionFilesIfMissing: true |
|  |

**Moving existing APK(s) to another release track**

The androidApkMove step requires at least the Google Play credential ID and the track to move APK(s) to, plus either an application ID and version code(s), or APK file(s) to read this information from.

Moving APKs from alpha to beta, specifying the application ID and version codes:

|  |
| --- |
| androidApkMove googleCredentialsId: 'My Google Play account',                 applicationId: '[com.example.app'](https://wiki.jenkins.io/display/JENKINS/com.example.app'),                 versionCodes: '1050, 2050, 3050',                 trackName: 'beta' |

Moving APKs from beta to staged rollout, specifying the application ID and version codes via the uploaded APKs:

|  |
| --- |
| androidApkMove googleCredentialsId: 'My Google Play account',                 fromVersionCode: false,                 apkFilesPattern: '\*\*/\*.apk',                 trackName: 'production',                 rolloutPercentage: '5' |