

[CICD Setup](#)

[Jenkins](#)

[Install Jenkins and Setup](#)

[Create a Build JOB in Jenkins: Create a project](#)

[SonarQube](#)

[Install sonarQube server](#)

[Connect SonarQube to Jenkins](#)

[Jenkins & Bitbucket](#)

[Link your Jenkins project to your Source Code: Jenkins & Bitbucket](#)

[Adding Credentials](#)

[Trigger Build on commit](#)

[Android](#)

[Jenkins build for Android](#)

[SonarQube for Android](#)

[Jenkins and SonarQube for Android](#)

[Fastlane for Android](#)

[Install fastlane](#)

[Collect your Google credentials](#)

[Configure supply](#)

[Fetch your app metadata](#)

[Set up environment variables](#)

[Deploy to Google Play using fastlane](#)

[Building your app](#)

[Uploading your APK](#)

[Web](#)

[Jenkins build for Web](#)

[Jenkins and SonarQube for Web](#)

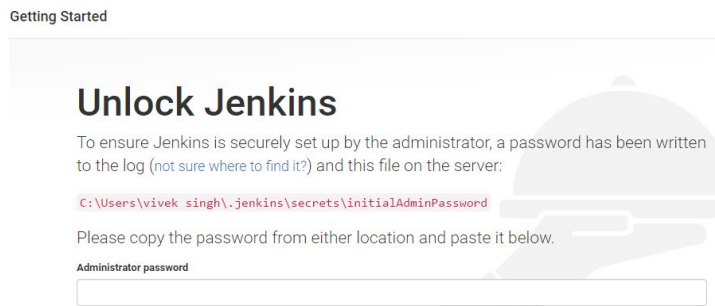
[Jenkins Deploy for web](#)

CICD Setup

Jenkins

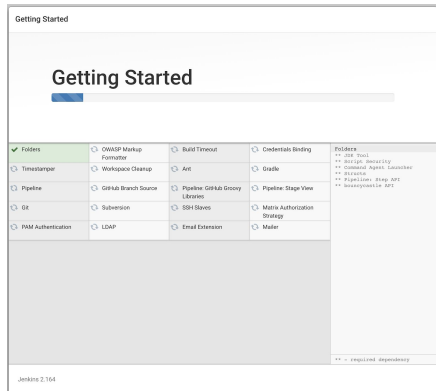
Install Jenkins and Setup

- 1) Install on macOS:
 - a) To install from the website, using a package:
 - b) [Download the latest package](#)
 - c) Open the package and follow the instructions
- 2) Jenkins can also be installed using brew:
 - a) Install the latest release version
 - i) `brew install jenkins`
 - b) Install the LTS version
 - i) `brew install jenkins-lts`
- 3) Unlock : After running jenkins and accessing the localhost:8080 page, if you see the unlock screen:



Follow the instructions here : https://www.youtube.com/watch?v=_7LaeqKAHvA

- a) Open the secrets folder shown in the screen
 - b) Right click on the Folder->Info
 - c) Add your user to the permission box
- 4) Follow the wizard, install suggested plugins...



Create a Build JOB in Jenkins: Create a project

1. Prerequisites:
 - a. Make sure Jenkins is started, if not run inside the Jenkins folder:
`java -jar jenkins.war --httpPort=9090`
 - b. Note, to stop Jenkins: In your browser go to : [Jenkins-URL]/exit
2. Create a Job:
 - a. Click on the create job button
 - b. Choose: Freestyle project
 - c. Name your project

SonarQube

Install sonarQube server

1. [Download](#) the SonarQube Community Edition
2. Unzip it, let's say in `C:\sonarqube` or `/opt/sonarqube`
3. Start the SonarQube Server:
 - a. `cd sonarqube-7.6/`
 - b. `./bin/macosx-universal-64/sonar.sh start`
4. Access it at : localhost:9000

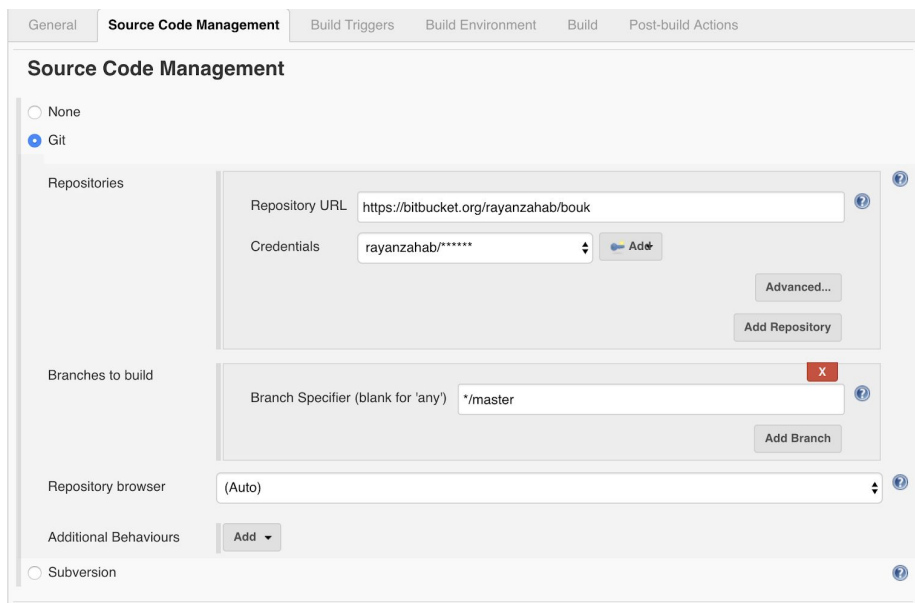
Connect SonarQube to Jenkins

- 1) Add the sonarqube plugin to Jenkins:
 - a) Jenkins home → Manage Jenkins → Manage plugins
 - b) Go to available tab, and search for Sonar
 - c) Locate plugin:
[SonarQube Scanner](#)
 - d) And download and restart

Jenkins & Bitbucket

Link your Jenkins project to your Source Code: Jenkins & Bitbucket

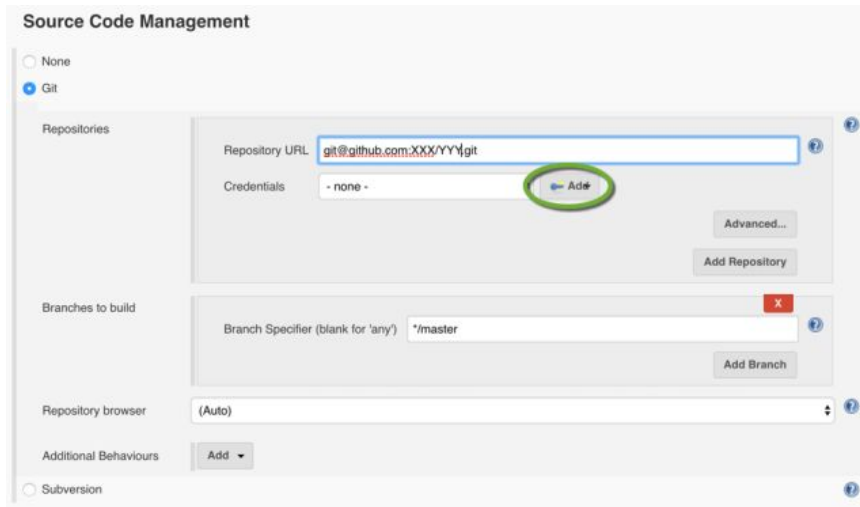
1. From your jenkins page go to the project page
2. Go to Configure
3. Click on the : Source Code Management tab
 - a. Choose Git
 - b. Enter your URL
 - c. Choose your credentials (Check next section to know how to add credentials)
 - d. Specify the branch



The screenshot shows the Jenkins 'Source Code Management' configuration page. At the top, there are tabs: 'General', 'Source Code Management' (selected), 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. Below the tabs, the 'Source Code Management' section is active. It features a left sidebar with radio buttons for 'None' and 'Git' (selected). The main area is divided into sections: 'Repositories' with fields for 'Repository URL' (https://bitbucket.org/rayanzahab/bouk) and 'Credentials' (rayanzahab/*****), and 'Branches to build' with a 'Branch Specifier (blank for 'any')' field containing */master. There are also buttons for 'Add Repository', 'Advanced...', 'Add Branch', and 'Repository browser' set to '(Auto)'. At the bottom, there is an 'Additional Behaviours' section with an 'Add' button and a 'Subversion' radio button.

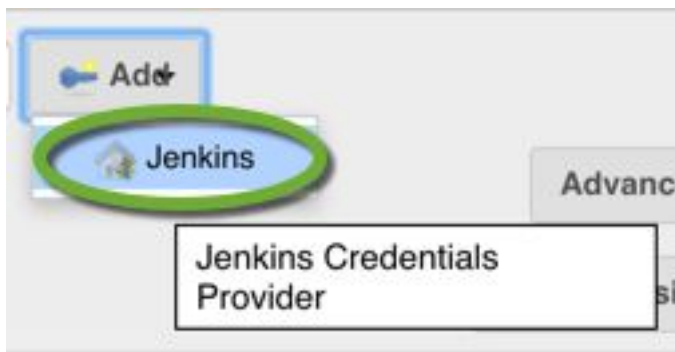
Adding Credentials

1. click the Add button to create the credential



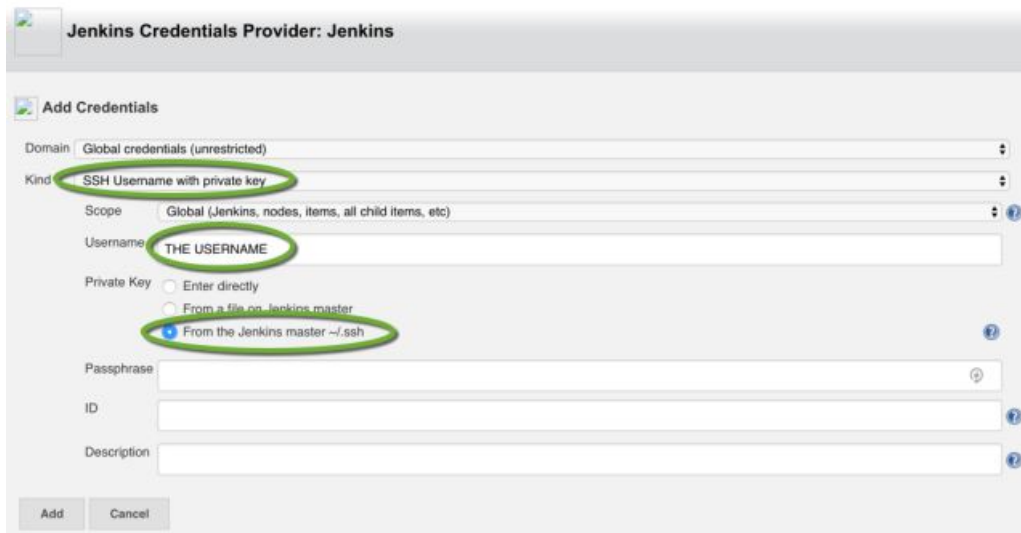
The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' radio button is selected. Under the 'Repositories' section, the 'Repository URL' is set to 'git@github.com:XXX:YY.git'. The 'Credentials' dropdown menu is open, showing '- none -' as the selected option. A green circle highlights the 'Add' button next to the dropdown. Other visible elements include the 'Advanced...' button, 'Add Repository' button, 'Branches to build' section with a 'Branch Specifier' of '*/master', 'Add Branch' button, 'Repository browser' set to '(Auto)', and 'Additional Behaviours' with an 'Add' button. The 'Subversion' radio button is unselected at the bottom.

2. Click Jenkins to select the credentials provider



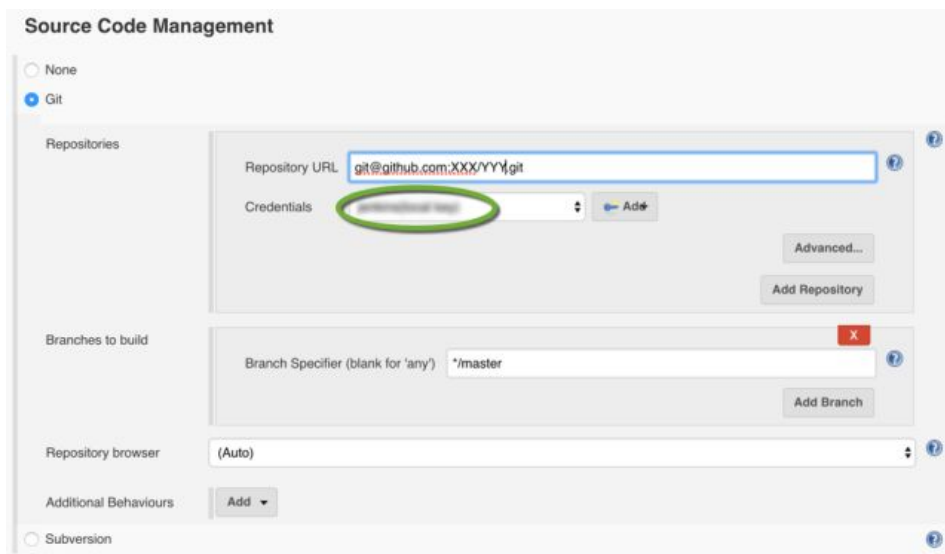
3. Select SSH Username with private key as the Kind
4. Enter the username you used when you created the SSH key for the Git repository

5. Select From the Jenkins master ~/.ssh as the Private Key



The screenshot shows the 'Add Credentials' form in Jenkins. The 'Kind' dropdown is set to 'SSH Username with private key'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'THE USERNAME'. Under the 'Private Key' section, the radio button 'From the Jenkins master ~/.ssh' is selected. The 'Passphrase', 'ID', and 'Description' fields are empty. The 'Add' button is at the bottom left.

6. Click the Add button
7. Now you will be able to see these credentials under the drop down.
8. In the Credentials drop down select the credential you have created (the Git user name)



The screenshot shows the 'Source Code Management' configuration page for Git. The 'Git' radio button is selected. Under 'Repositories', the 'Repository URL' is 'git@github.com:XXX/YYY.git'. The 'Credentials' dropdown is set to 'Jenkins Credentials Provider: Jenkins'. The 'Branches to build' section has a 'Branch Specifier (blank for 'any')' of '*/master'. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. The 'Subversion' radio button is unselected.

Trigger Build on commit

1. Go to the Build Triggers tab under the project:
2. Check Poll SCM to trigger build after commits to your chosen branch
3. Specify when to check for changes using the syntax of cron :

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Build periodically

☐ Deploy periodically

☐ GitHub hook trigger for GITScm polling

☒ Poll SCM

Schedule

Do you really mean "every minute" when you say "*****"? Perhaps you meant "H*****" to poll once per hour

Would last have run at Monday, February 25, 2019 11:55:08 AM CET; would next run at Monday, February 25, 2019 11:55:08 AM CET.

Ignore post-commit hooks

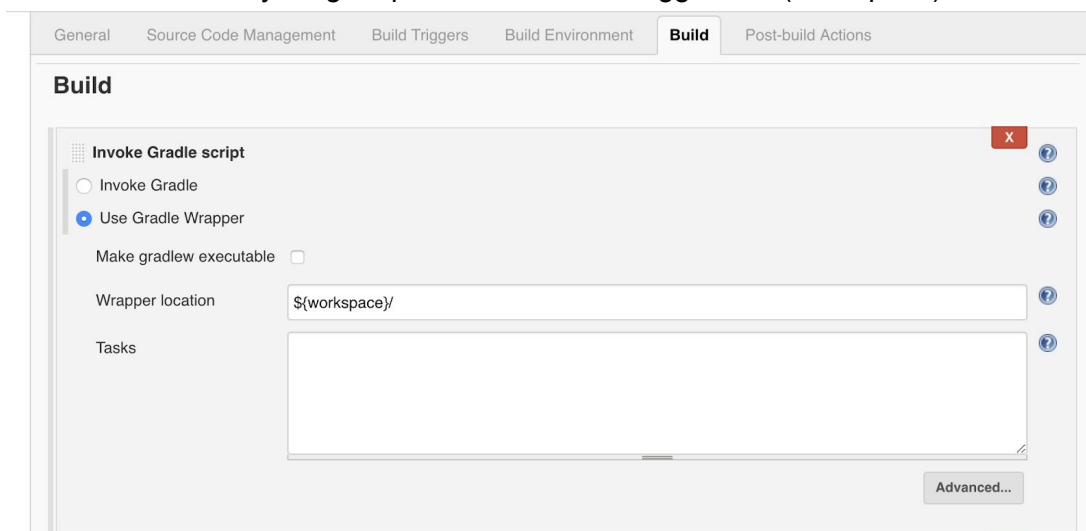
☐

Android

Jenkins build for Android

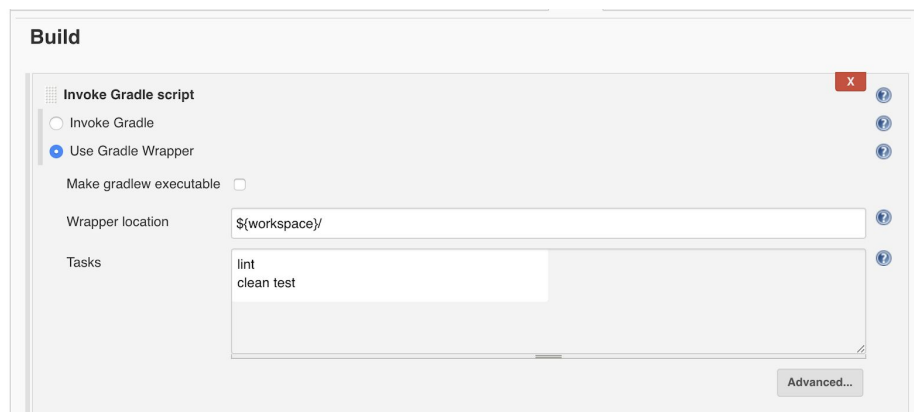
If your project is an android application:

1. Go to Build Tab
2. Check Gradle Wrapper (If you would like to use gradlew instead of gradle)
3. Specify your workspace location
 - a. Usually Jenkins has a defined variable for where it put the codes after pulling from your git repo when a build is triggered: `$(workspace)/`



The screenshot shows the Jenkins configuration page for a build job, specifically the 'Build' tab. The 'Invoke Gradle script' section is expanded, showing the 'Use Gradle Wrapper' option selected. The 'Wrapper location' field is set to `$(workspace)/`. The 'Tasks' field is empty. The 'Advanced...' button is visible at the bottom right of the configuration area.

If you have lint and unit tests setup in your project you can also ask the build to run them by adding the commands under tasks:



This screenshot shows the same Jenkins configuration page as the previous one, but with the 'Tasks' field populated with the commands `lint` and `clean test`. The 'Advanced...' button is still visible at the bottom right.

SonarQube for Android

To send the Android code to SonarQube you need to setup the prerequisites below.

- Ensure Android SDK Platform 27 is installed or update the [app/build.gradle](#) with the SDK version you have installed. To instal SDK platform 27 go to Android Studio -> Preferences -> Appearance & Behavior -> System Settings -> Android SDK and check the Android 8.1 (Oreo) then click Ok.
- Add the following to your project buildscript

```
buildscript {  
    dependencies {  
        classpath 'org.sonarsource.scanner.gradle:sonarqube-gradle-plugin:2.7'  
    }  
}
```

- Apply the plugin from your build.gradle:

```
apply plugin: 'org.sonarqube'
```

- Define the sonarqube rules and resources in your build.gradle:

Make sure you update the sonar.login : Go to your sonar link, Admin->security and generate a token.

```
sonarqube {  
    properties {  
        property "sonar.host.url", "http://localhost:9000"  
        property "sonar.login", "75eeb35bb22e4be73769d08598db477ada40c6ae"  
        def libraries = project.android.sdkDirectory.getPath() + "/platforms/android-27/android.jar,"  
+  
        "build/intermediates/classes/*"  
        property "sonar.projectVersion", System.getenv("BUILD_NUMBER")  
        property "sonar.sourceEncoding", "UTF-8"  
        property "sonar.sources", "src/main/java"  
        property "sonar.binaries", "build/intermediates/classes/*"  
        property "sonar.libraries", libraries  
        property "sonar.java.binaries", "build/intermediates/classes/*"  
        property "sonar.java.libraries", libraries  
        property "sonar.tests", "src/test/java" // where the tests are located  
        property "sonar.java.test.binaries", "build/intermediates/classes/*"  
        property "sonar.java.test.libraries", libraries  
        property "sonar.scm.provider", "git"
```

```
    property "sonar.jacoco.reportPaths", "build/jacoco/testDebugUnitTest.exec"
    property "sonar.jacoco.reportPaths", "build/jacoco/testDebugUnitTest.exec,
build/spoon/mock/debug/coverage/merged-coverage.ec"
    // path to coverage report
    property "sonar.java.coveragePlugin", "jacoco"
    property "sonar.junit.reportsPath", "build/test-results/testDebugUnitTest"
    // path to junit reports
    property "sonar.android.lint.report", "build/reports/*.xml"
    // path to lint reports
}
}
```

- Run the following command from the project root folder

```
./gradlew --info sonarqube
```

- Access the report from the link provided in the command line , or simply access your sonar dashboard on go to your project.

Jenkins and SonarQube for Android

- 2) Invoke running sonar after build:
 - a) Go to configure inside the project
 - b) Go to build tab
 - c) Under add build setup choose: Execute SonarQube Scanner
 - d) Provide the properties as shown below:

The screenshot shows the Jenkins configuration page for a build step named 'Execute SonarQube Scanner'. The 'Build' tab is selected, and the configuration is as follows:

- Task to run:** (Empty text field)
- JDK:** (Inherit From Job) [Dropdown menu]
- JDK to be used for this SonarQube analysis:** (Label)
- Path to project properties:** (Empty text field)
- Analysis properties:**

```
sonar.projectKey=ExperimentKey
sonar.host.url=http://localhost:9000
sonar.login=d9fbf585fe2c6a1a5ef2dbf4a73cdc0b07da0712
sonar.sources=app/src/main
```
- Additional arguments:** (Empty text field with a dropdown arrow)
- JVM Options:** (Empty text field with a dropdown arrow)

At the bottom, the 'Add build step' dropdown menu is open, showing two options: 'Execute SonarQube Scanner' and 'Execute Windows batch command'.

Now the sonarqube scanner will be invoked after every build.

Fastlane for Android

Install fastlane

- a) Make sure you have the latest version of the Xcode command line tools installed:
`xcode-select --install`
- b) Install fastlane using
`[sudo] gem install fastlane -NV`
- c) or alternatively using brew cask install fastlane
- 2) Setup: <https://docs.fastlane.tools/getting-started/android/setup/>
- 3) Test by running pre existing actions:
`fastlane env`
- 4) Deploy using fastlane: <https://docs.fastlane.tools/getting-started/android/setup/>

Collect your Google credentials

Tip: If you see Google Play Console or Google Developer Console in your local language, add &hl=en at the end of the URL (before any #...) to switch to English.

1. Open the [Google Play Console](#)
2. Click the Settings menu entry, followed by API access
3. Click the CREATE SERVICE ACCOUNT button
4. Follow the Google Developers Console link in the dialog, which opens a new tab/window:
 1. Click the CREATE SERVICE ACCOUNT button at the top of the Google Developers Console
 2. Provide a Service account name
 3. Click Select a role and choose Service Accounts > Service Account User
 4. Check the Furnish a new private key checkbox
 5. Make sure JSON is selected as the Key type
 6. Click SAVE to close the dialog

7. Make a note of the file name of the JSON file downloaded to your computer
5. Back on the Google Play Console, click DONE to close the dialog
6. Click on Grant Access for the newly added service account
7. Choose Release Manager from the Role dropdown
8. Click ADD USER to close the dialo

Configure supply

Edit your fastlane/Appfile and change the json_key_file line to have the path to your credentials file:

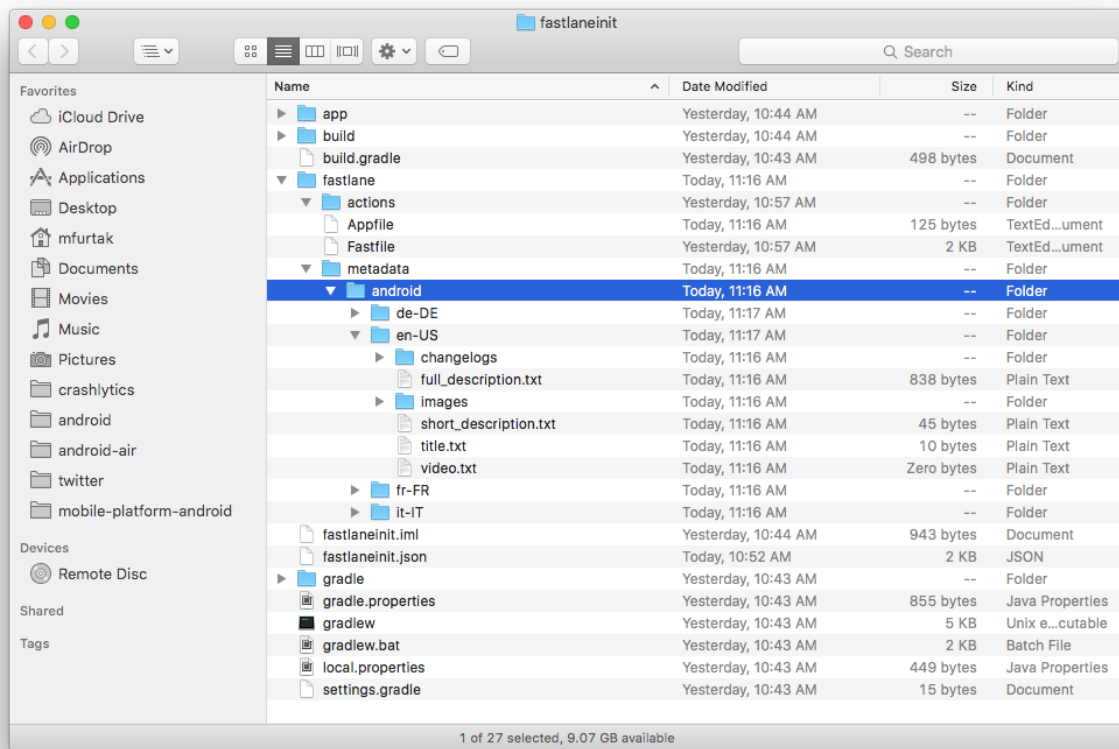
```
json_key_file "/path/to/your/downloaded/key.json"
```

Fetch your app metadata

If your app has been created on the Google Play developer console, you're ready to start using supply to manage it! Run:

```
fastlane supply init
```

and all of your current Google Play store metadata will be downloaded to fastlane/metadata/android.



Due to limitations of the Google Play API, supply can't download existing screenshots or videos.

Set up environment variables

fastlane requires some environment variables set up to run correctly. In particular, having your locale not set to a UTF-8 locale will cause issues with building and uploading your build. In your shell profile add the following lines:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

You can find your shell profile at ~/.bashrc, ~/.bash_profile, ~/.profile or ~/.zshrc depending on your system.

Deploy to Google Play using fastlane

Building your app

fastlane takes care of building your app by delegating to your existing Gradle build. Just add the following to your `Fastfile`:

```
lane :playstore do
  gradle(
    task: 'assemble',
    build_type: 'Release'
  )
end
```

Try running the lane with:

```
fastlane playstore
```

When that completes you should have the appropriate APK ready to go in the standard output directory. To get a list of all available parameters for the `gradle` action, run:

```
fastlane action gradle
```

Uploading your APK

To upload your binary to Google Play, *fastlane* uses a tool called *supply*. Because *supply* needs authentication information from Google, if you haven't yet done the [supply setup steps](#), please do those now!

With that done, simply add a call to *supply* to the lane you set up above:


```

lane :playstore do
  gradle(
    task: 'assemble',
    build_type: 'Release'
  )
  upload_to_play_store # Uploads the APK built in the gradle step above
                        and releases it to all production users
end

```

This will also: - Upload app metadata from `fastlane/metadata/android` if you previously ran `fastlane supply init` - Upload expansion files (obbs) found under the same directory as your APK as long as: - They are identified by type as **main** or **patch** by containing `main` or `patch` in their file names - There is at most one of each type - Upload screenshots from `fastlane/metadata/android` if you previously ran `screengrab` - Create a new production build - Release the production build to all users

If you would like to capture and upload screenshots automatically as part of your deployment process, check out the [fastlane screenshots for Android](#) guide to get started!

To gradually roll out a new build you can use:

```

lane :playstore do
  # ...
  upload_to_play_store(
    track: 'rollout',
    rollout: '0.5'
  )
end

```

To get a list of all available parameters for the *upload_to_play_store* action, run:

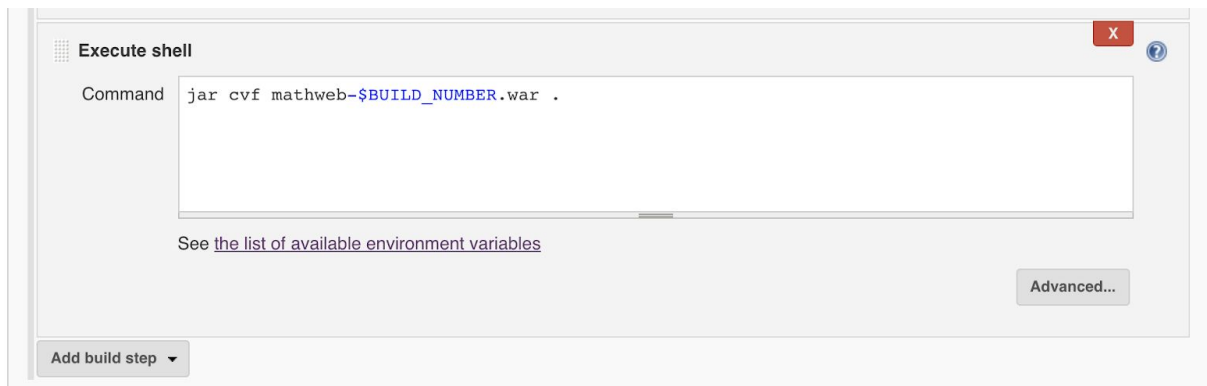
```
fastlane action upload_to_play_store
```

Web

Jenkins build for Web

If your project is an android application:

4. Go to Build Tab
5. Choose: Execute shell
6. Build the war file : `jar cvf mathweb-$BUILD_NUMBER.war .`



Jenkins and SonarQube for Web

Invoke running sonar after/before build:

- e) Go to configure inside the project
- f) Go to build tab
- g) Under add build setup choose: Execute SonarQube Scanner
- h) Provide the properties as shown below:

The screenshot shows the Jenkins configuration page for a build job, specifically the 'Build' tab. The 'Execute SonarQube Scanner' build step is selected. The configuration fields are as follows:

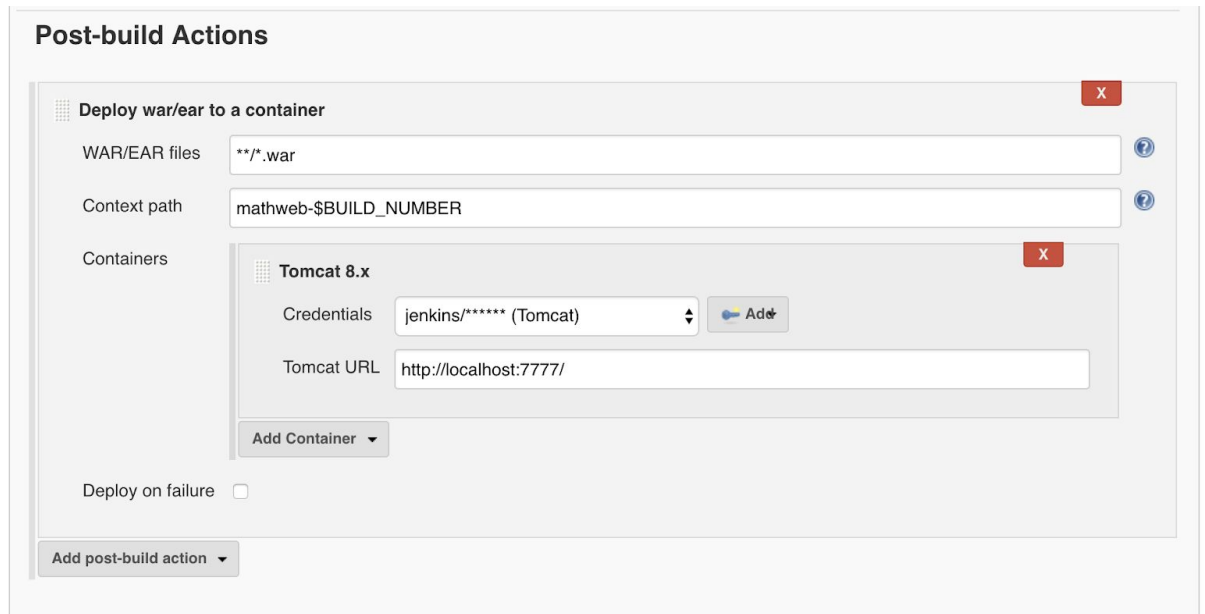
- Task to run:** (Empty text field)
- JDK:** (Inherit From Job)
- JDK to be used for this SonarQube analysis:** (Label for the JDK field)
- Path to project properties:** (Empty text field)
- Analysis properties:** (Text area containing:
sonar.projectKey=MathWeb
sonar.sources=.
sonar.host.url=http://localhost:9000
sonar.login=b4422eaf93c6ca3f5a6c293a0aa9bac4ea4aa504)
- Additional arguments:** (Empty text field)
- JVM Options:** (Empty text field)

Now the sonarqube scanner will be invoked after every build.

Jenkins Deploy for web

Deploy the web application to a Tomcat server after build:

1. Go to the Post-build Actions tab inside the project's configure page
2. From the Add post-build action drop down choose: Deploy war/ear to a container
3. Add the details as below:
4. Make sure to add the tomcat credentials to allow deploying



The screenshot shows the 'Post-build Actions' configuration page in Jenkins. The main section is titled 'Deploy war/ear to a container' and contains the following fields:

- WAR/EAR files:** A text input field containing the pattern `**/*.war`.
- Context path:** A text input field containing the value `mathweb-$BUILD_NUMBER`.
- Containers:** A section for configuring deployment targets.
 - Tomcat 8.x:** A sub-section for Tomcat 8.x configuration.
 - Credentials:** A dropdown menu showing 'jenkins/****** (Tomcat)' with an 'Add' button next to it.
 - Tomcat URL:** A text input field containing the value `http://localhost:7777/`.
 - Add Container:** A button with a dropdown arrow to add more containers.
- Deploy on failure:** A checkbox that is currently unchecked.

At the bottom of the configuration area, there is a button labeled 'Add post-build action' with a dropdown arrow.