CI/CD using Jenkins and Docker

JUNE 10, 2018 NIDHI GUPTA LEAVE A COMMENT

This blog will help you to setup a CI/CD pipeline using Jenkins and Docker. It includes automation using Jenkins Pipeline/Groovy scripting language, it uses sonar for code quality and artifactory for artifactory management

Tools:

Jenkins- CI/CD

Git/GitHub—Source Control Management

Docker—Container

JaCOCO—Code Coverage Tool

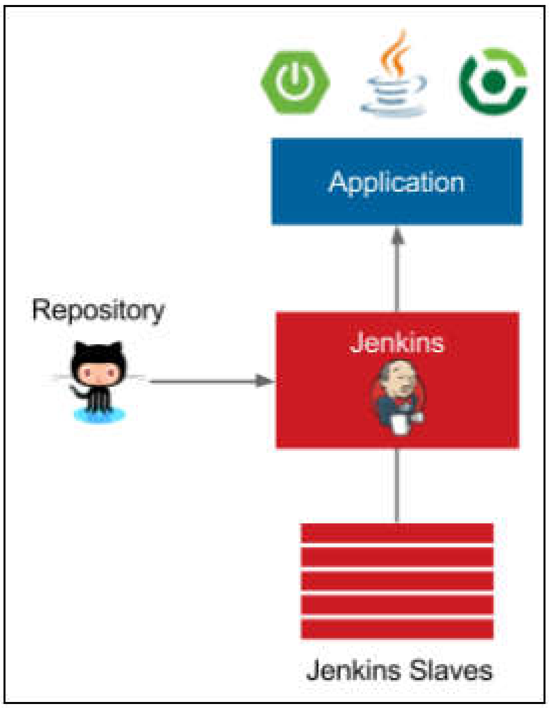
Gradle—Build tool

Ansible—Configuration Management

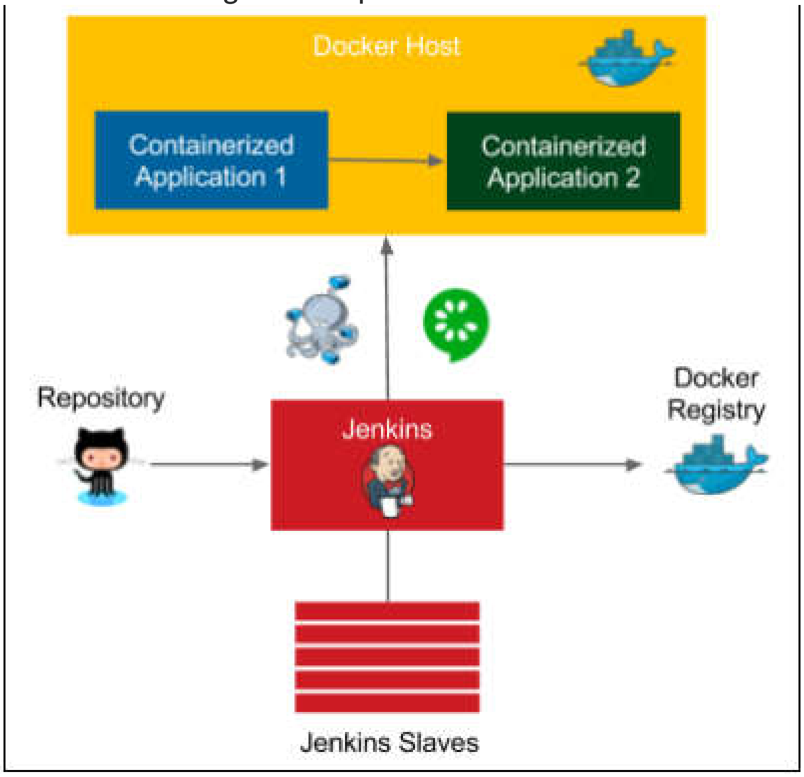
Ansible/Github/Docker/Cucumber/

Continuous Integration Pipeline

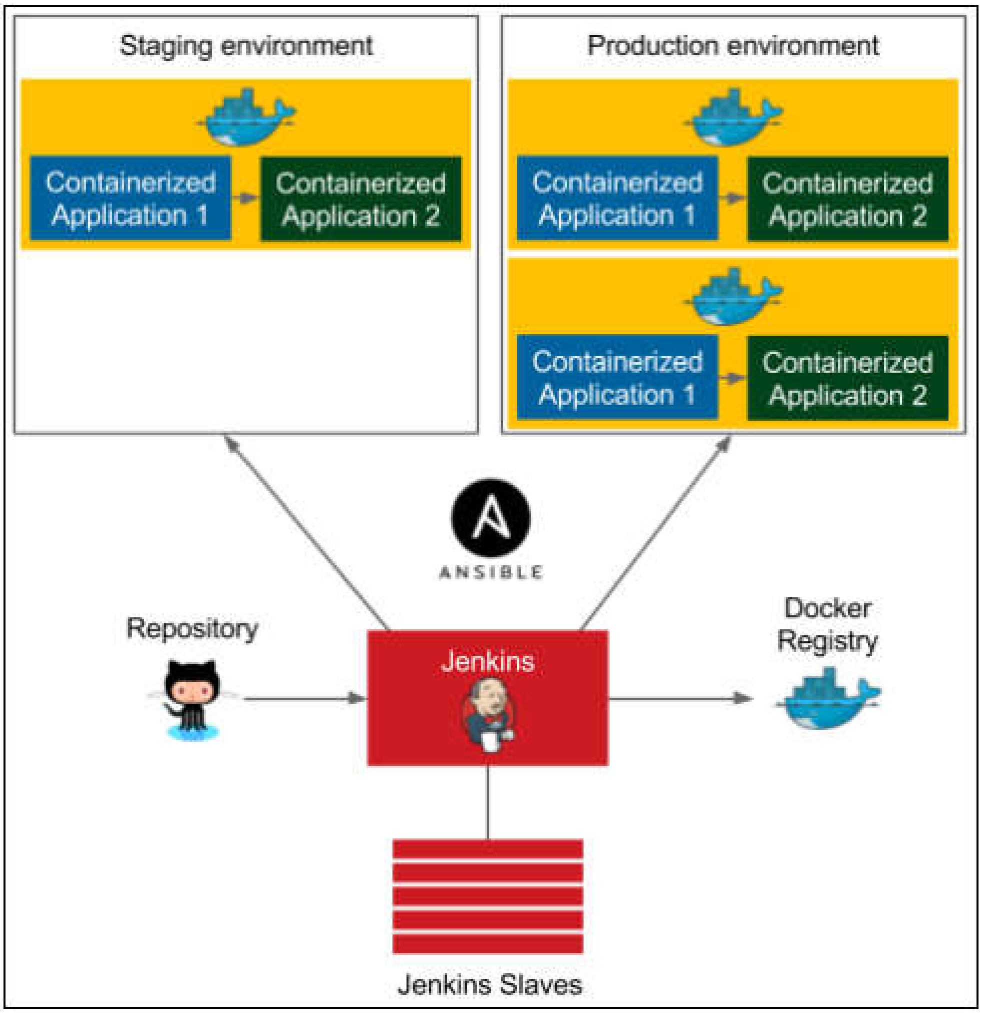
The application is a simple web service written in Java with the Spring Boot framework. Gradle is used as a build tool and GitHub as the source code repository. Every commit to GitHub automatically triggers the Jenkins build, which uses Gradle to compile Java code, run unit tests, and perform additional checks (code coverage, static code analysis, and so on). After the Jenkins build is completed, a notification is sent to the developers.



Continuous Integration Pipeline



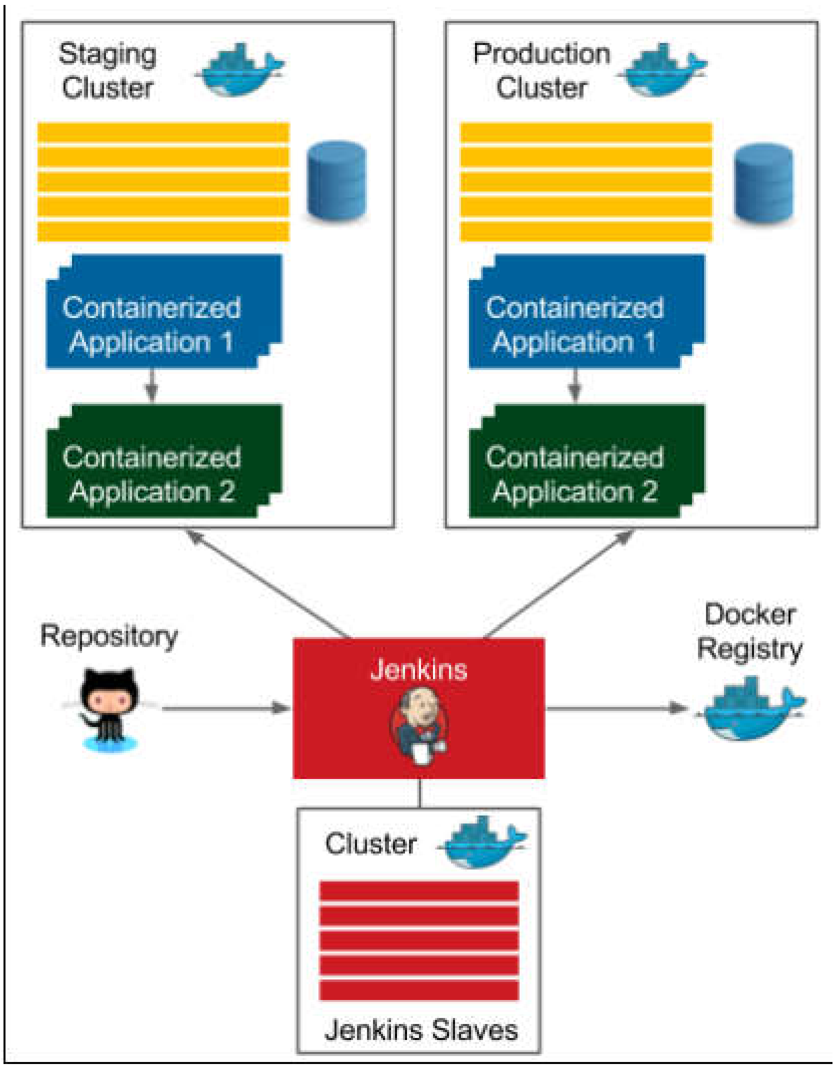
Configuration Management with Ansible



Staging environment
Containerized Application 1
Containerized Application 2
Production environment
Containerized Application 1
Containerized Application 2
Containerized Application 1
Containerized Application 2
ANSIBLE
Repository
Jenkins
Docker Registry
Jenkins Slaves

Continuous Delivery Pipeline

Clustering with Docker Swarm/Advanced Continuous Delivery


Staging Cluster
Containerized Application 1
Containerized Application 2
Production Cluster
Containerized Application 1
Containerized Application 2
Repository
Jenkins
Docker Registry
Cluster
Jenkins Slaves

Docker Installation: Click Here

Jenkins setup on AWS

Java 8 should be present,if not please use the below command

Install JAVA 8

yum install wget (in case wget is not found)

```
$ wget --header "Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u161-
b12/2f38c3b165be4555a1fa6e98c45e0808/jdk-8u161-linux-x64.rpm
```

```
$ sudo yum localinstall jdk-8u161-linux-x64.rpm
```

```
export JAVA_HOME=/usr/java/jdk1.8.0_161/
export JRE_HOME=/usr/java/jdk1.8.0_161/jre
```

```
PATH=$PATH:$HOME/bin:$JAVA_HOME/bin

export PATH
sudo alternatives --config java
```

Install Jenkins

```
sudo yum update
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-
stable/jenkins.repo
sudo rpm --import http://pkg.jenkins-ci.org/redhat-stable/jenkins-ci.org.key
sudo yum install jenkins
sudo service jenkins start
```

Install Git on Jenkins server

yum install git

Open the url in the browser. Default port is 8080

http://localhost:8080/

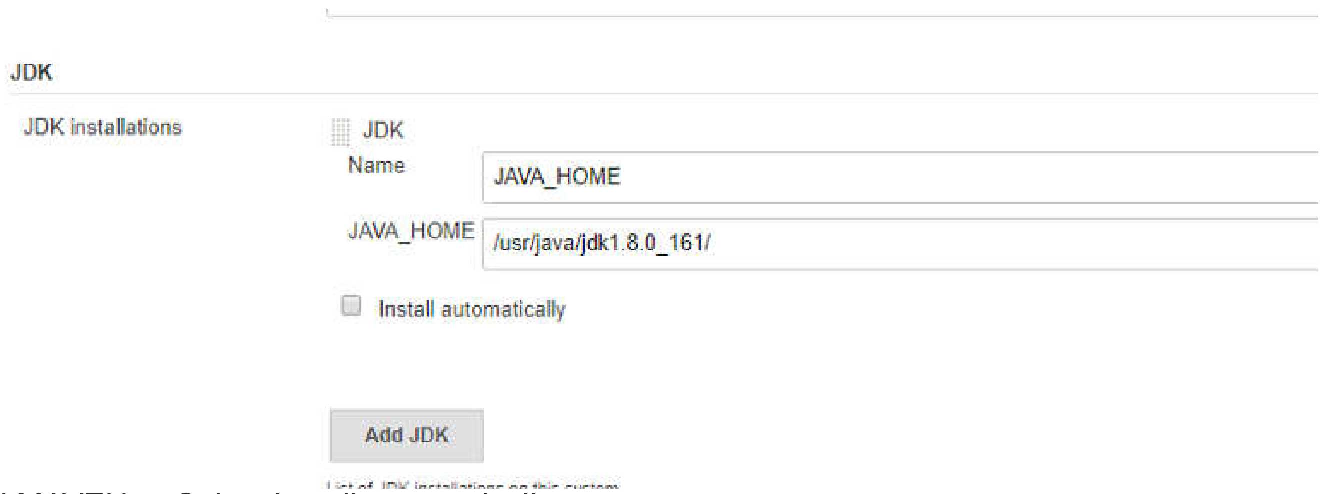
It will ask for the initial password, please run the below command

cat /var/lib/jenkins/secrets/initialAdminPassword

Configure Jenkins with JAVA_HOME,MAVEN_HOME

Go to Manage Jenkins ->Global Tool Configuration

Add JDK -> JAVA_HOME -> put the java_home path of the machine


JDK

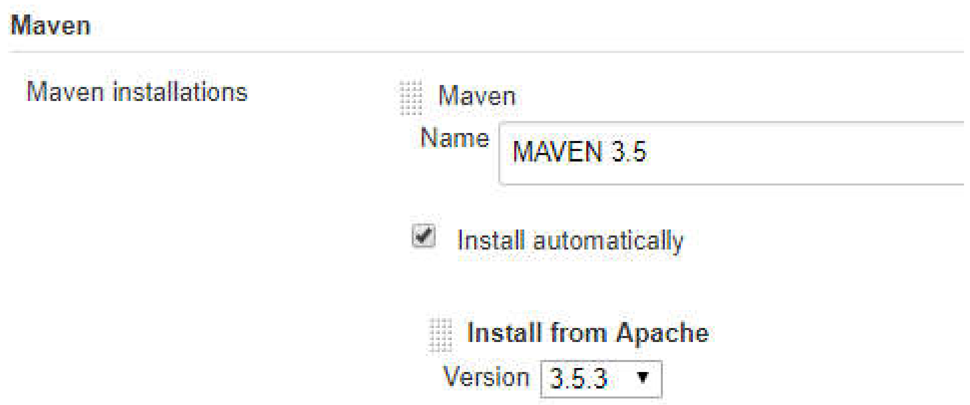
JDK installations

JDK

Name: JAVA_HOME

JAVA_HOME: /usr/java/jdk1.8.0_161/

Add MAVEN -> Select Install automatically.



Maven

Maven installations

Maven

Name: MAVEN 3.5

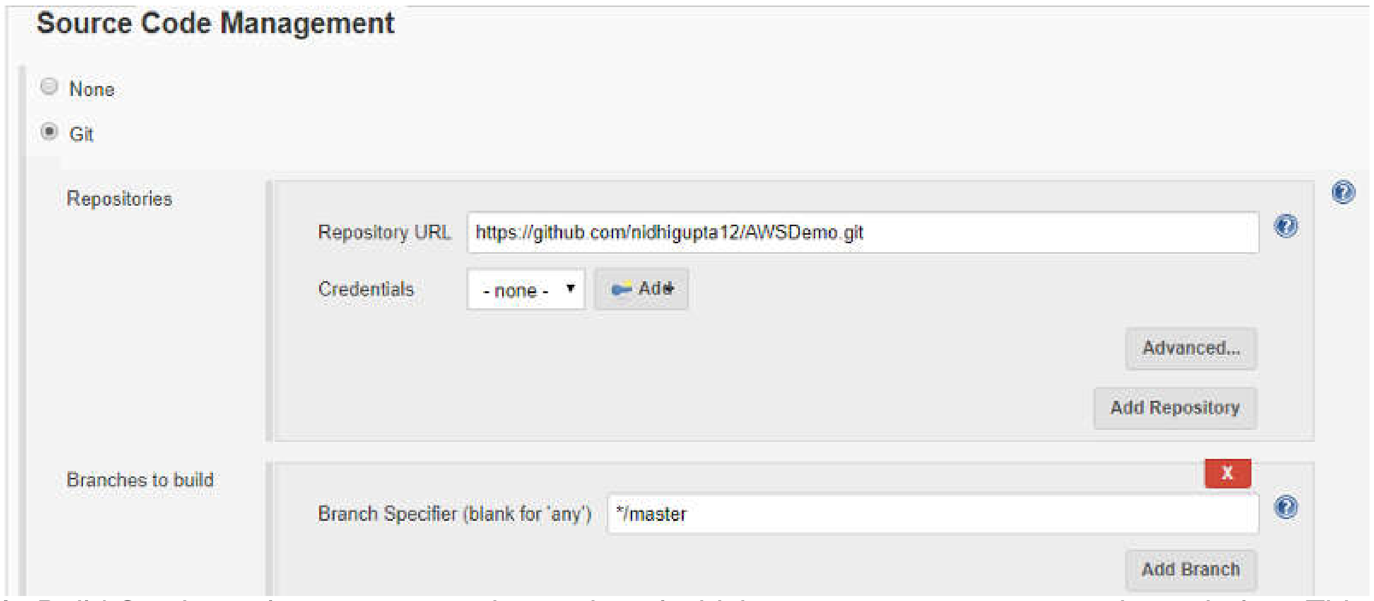
Install automatically [✓]

Install from Apache

Version: 3.5.3

Create a freestyle project in Jenkins

In source code management put this url as shown below
“https://github.com/nidhigupta12/AWSDemo.git”



Source Code Management

None

Git [✓]

Repositories

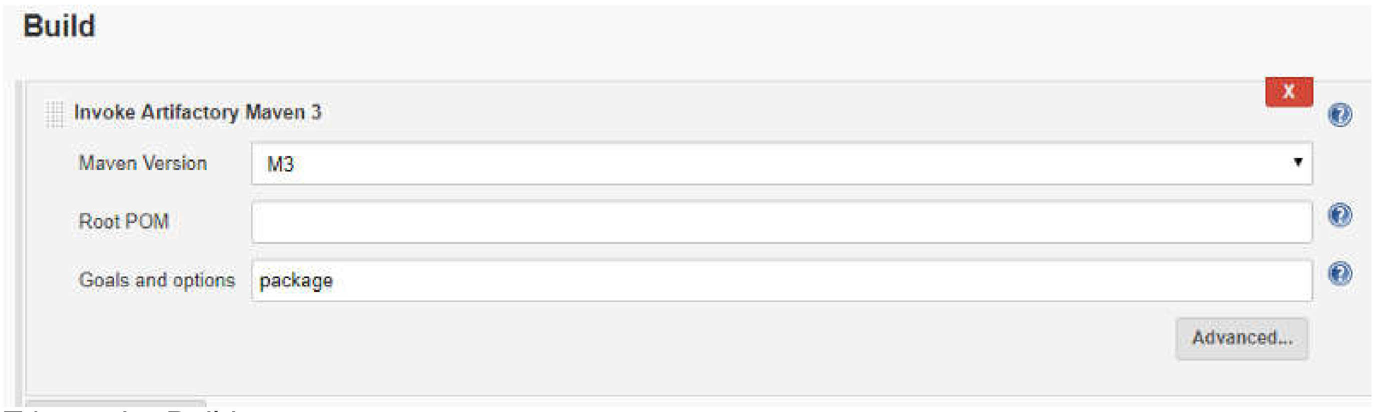
Repository URL: https://github.com/nidhigupta12/AWSDemo.git

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

In Build Section,select maven version and goal which you want to execute as shown below. This is very important, you have to select invoke Artifactory maven 3 only not the invoke maven top level targets.



Build

Invoke Artifactory Maven 3

Maven Version: M3

Root POM:

Goals and options: package

Trigger the Build

Output of this job

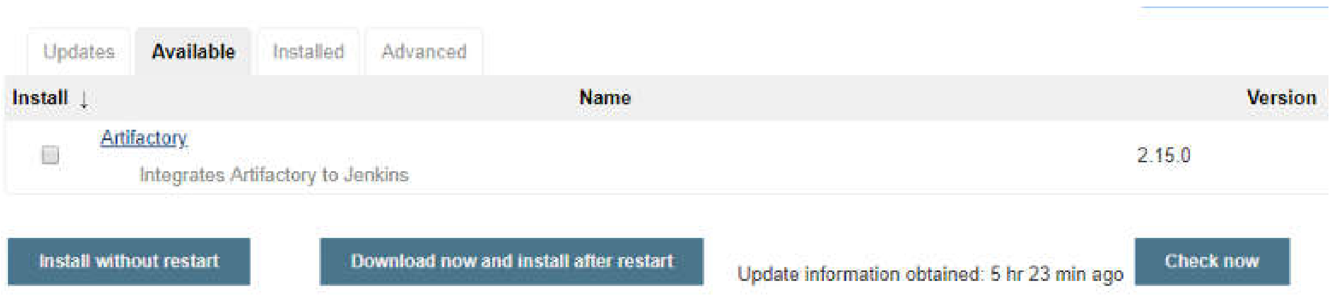
```
[INFO] Packaging webapp
[INFO] Assembling webapp [LoginWebApp] in
[/var/lib/jenkins/workspace/Test_Maven/target/LoginWebApp]
[INFO] Processing war project
[INFO] Copying webapp resources
[/var/lib/jenkins/workspace/Test_Maven/src/main/webapp]
[INFO] Webapp assembled in [732 msecs]
[INFO] Building war:
/var/lib/jenkins/workspace/Test_Maven/target/LoginWebApp.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 01:36 min
[INFO] Finished at: 2018-03-27T20:59:52Z
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

Install Jfrog Artifactory. Click Here

Now storing the artifacts to the jfrog artifactory

Integrate Artifactory with Jenkins

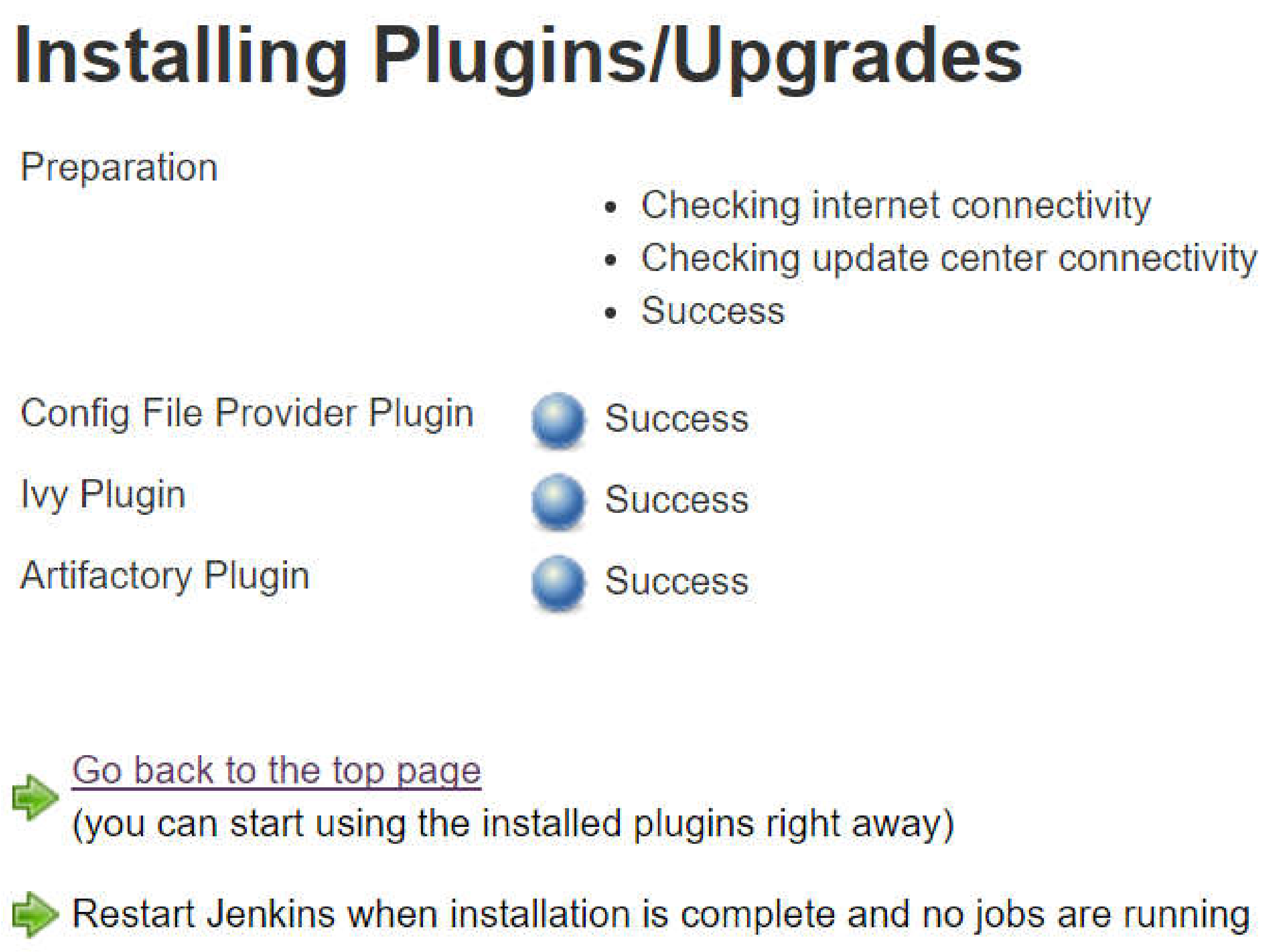
Go to Manage Jenkins -> Manage Plugins



Install ↓	Name	Version
☐	Artifactory Integrates Artifactory to Jenkins	2.15.0

Update information obtained: 5 hr 23 min ago

Once plugin installation is successful, you can configure Artifactory-related settings in Jenkins:



Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

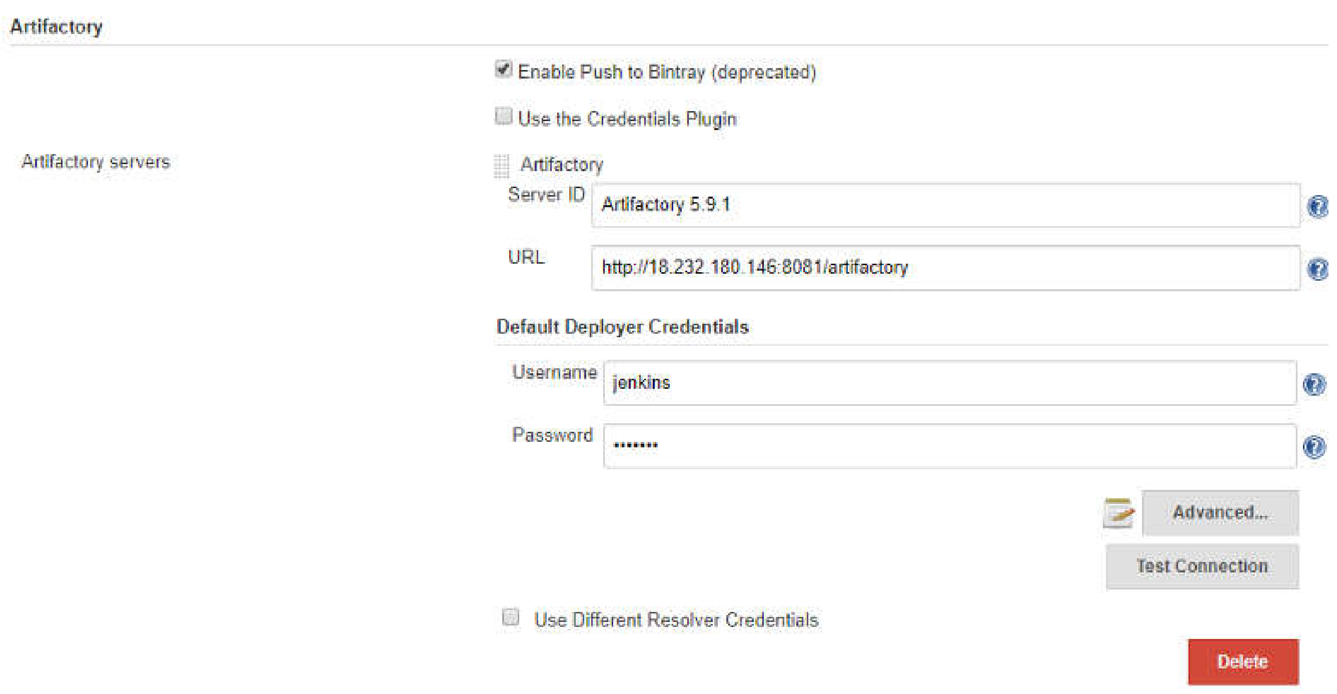
Config File Provider Plugin	● Success
Ivy Plugin	● Success
Artifactory Plugin	● Success

Go back to the top page
(you can start using the installed plugins right away)

Restart Jenkins when installation is complete and no jobs are running

Configure Artifactory in Jenkins:

Go to Manage Jenkins ->Configure System



Artifactory

[✓] Enable Push to Bintray (deprecated)

[ ] Use the Credentials Plugin

Artifactory servers

Artifactory

Server ID	Artifactory 5.9.1
URL	http://18.232.180.146:8081/artifactory

Default Deployer Credentials

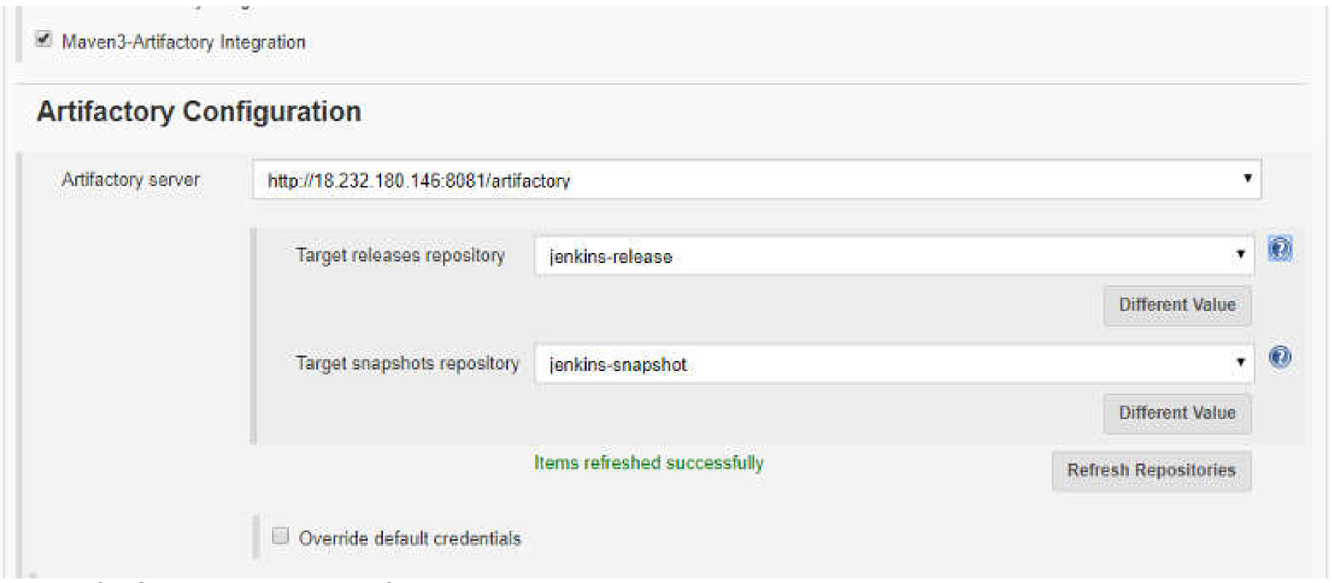
Username	jenkins
Password	•••••••

[ ] Use Different Resolver Credentials

1. Go to Section “Build Environment”

3. Select Maven3-Artifactory Integration

4. Click on Refresh Repositories and select the repository in the release and snapshot field from the lists:


☑ Maven3-Artifactory Integration

Artifactory Configuration

Artifactory server: http://18.232.180.146:8081/artifactory

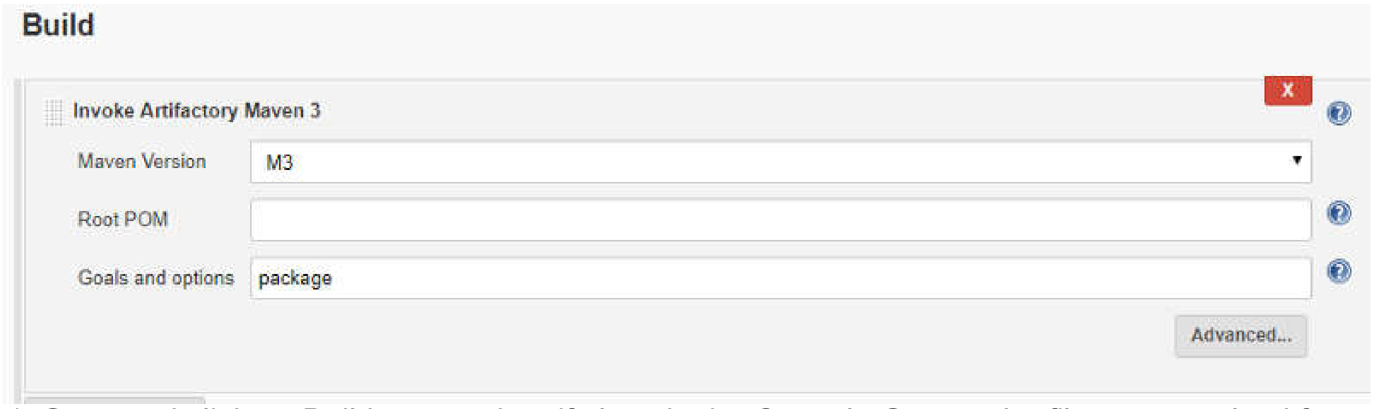
Target releases repository: jenkins-release

Target snapshots repository: jenkins-snapshot

Items refreshed successfully

☐ Override default credentials

Add Build Step as shown below


Build

Invoke Artifactory Maven 3

Maven Version: M3

Root POM:

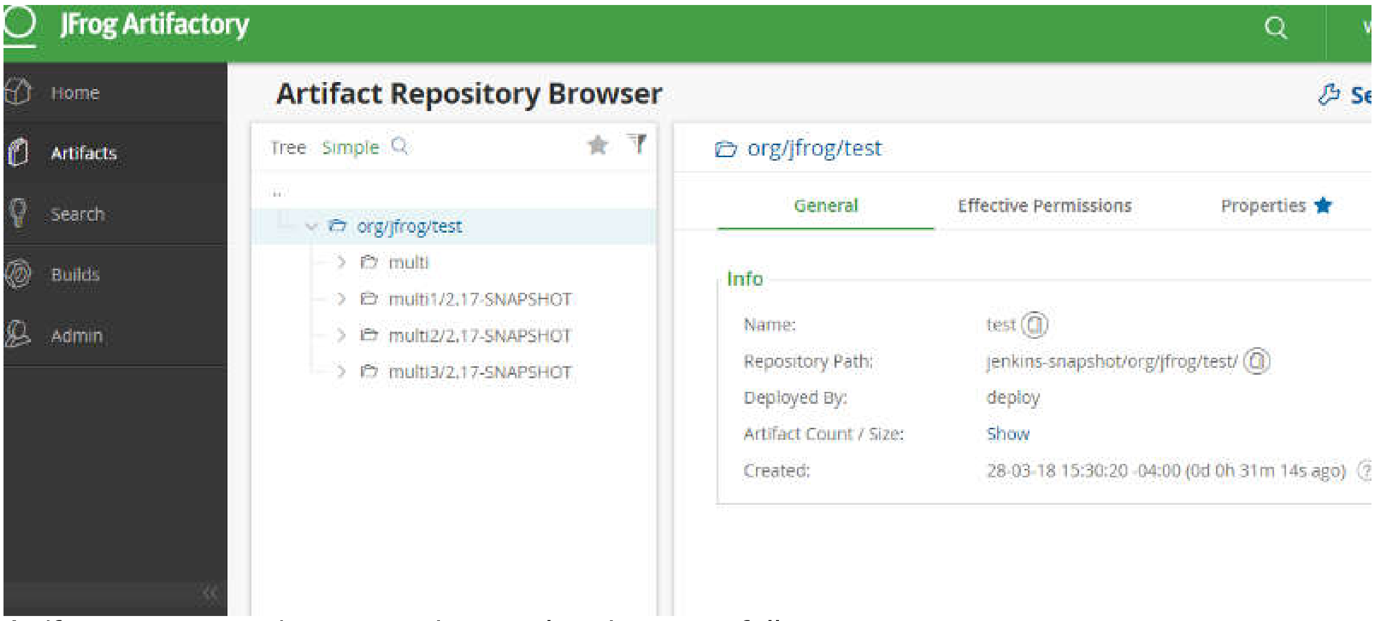
Goals and options: package

1. Save and click on Build now and verify logs in the Console Output. Jar files are resolved from the local repository or Artifactory:
2. Once the package is created, it is stored in Artifactory too:

```
[main] INFO org.apache.maven.plugin.war.WarMojo - Webapp assembled in [85
msecs]
[main] INFO org.codehaus.plexus.archiver.war.WarArchiver - Building war:
C:\Program Files
(x86)\Jenkins\workspace\Test_Maven_MyProject\target\LoginWebApp-1.0-
SNAPSHOT.war
[main] INFO org.jfrog.build.extractor.maven.BuildDeploymentHelper - Artifactory
Build Info Recorder: Saving Build Info to 'C:\Program Files
(x86)\Jenkins\workspace\Test_Maven_MyProject\target\build-info.json'
[main] INFO org.jfrog.build.extractor.maven.BuildInfoClientBuilder - Deploying
artifact: http://localhost:8081/artifactory/jenkins-
snapshot/com/javawebtutor/LoginWebApp/1.0-SNAPSHOT/LoginWebApp-1.0-SNAPSHOT.war
[main] INFO org.jfrog.build.extractor.maven.BuildDeploymentHelper - Artifactory
Build Info Recorder: Deploying build info ...
[main] INFO org.jfrog.build.extractor.maven.BuildInfoClientBuilder - Deploying
```

```
build descriptor to: http://localhost:8081/artifactory/api/build
[main] INFO org.jfrog.build.extractor.maven.BuildInfoClientBuilder - Build
successfully deployed. Browse it in Artifactory under
http://localhost:8081/artifactory/webapp/builds/Test_Maven_MyProject/10
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - ----------------
-------------------------------------------------------
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - BUILD SUCCESS
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - ----------------
-------------------------------------------------------
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Total time: 5.419
s
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Finished at:
2018-03-30T16:55:43-04:00
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - ----------------
-------------------------------------------------------
Finished: SUCCESS
```

1. Go to Artifactory and verify the package:



Name: test
Repository Path: jenkins-snapshot/org/jfrog/test/
Deployed By: deploy
Artifact Count / Size: Show
Created: 28-03-18 15:30:20 -04:00 (0d 0h 31m 14s ago)

Artifactory setup using maven is completed successfully

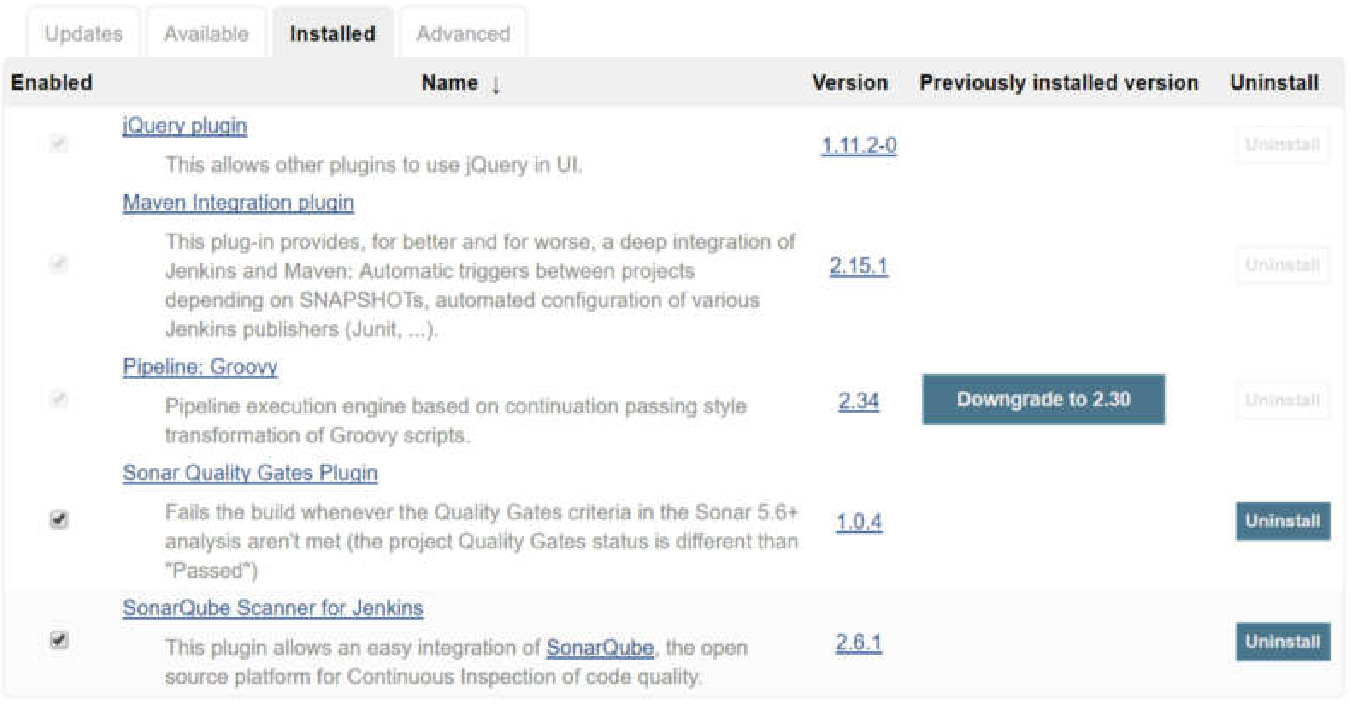
Sonarqube Setup

1. Download SonarQube from https://www.sonarqube.org/downloads/ and extract it in the system:
2. Execute StartSonar.bat/.sh as per OS

3. Once SonarQube is up and running, open the browser at http://localhost:9000 to visit the SonarQube dashboard

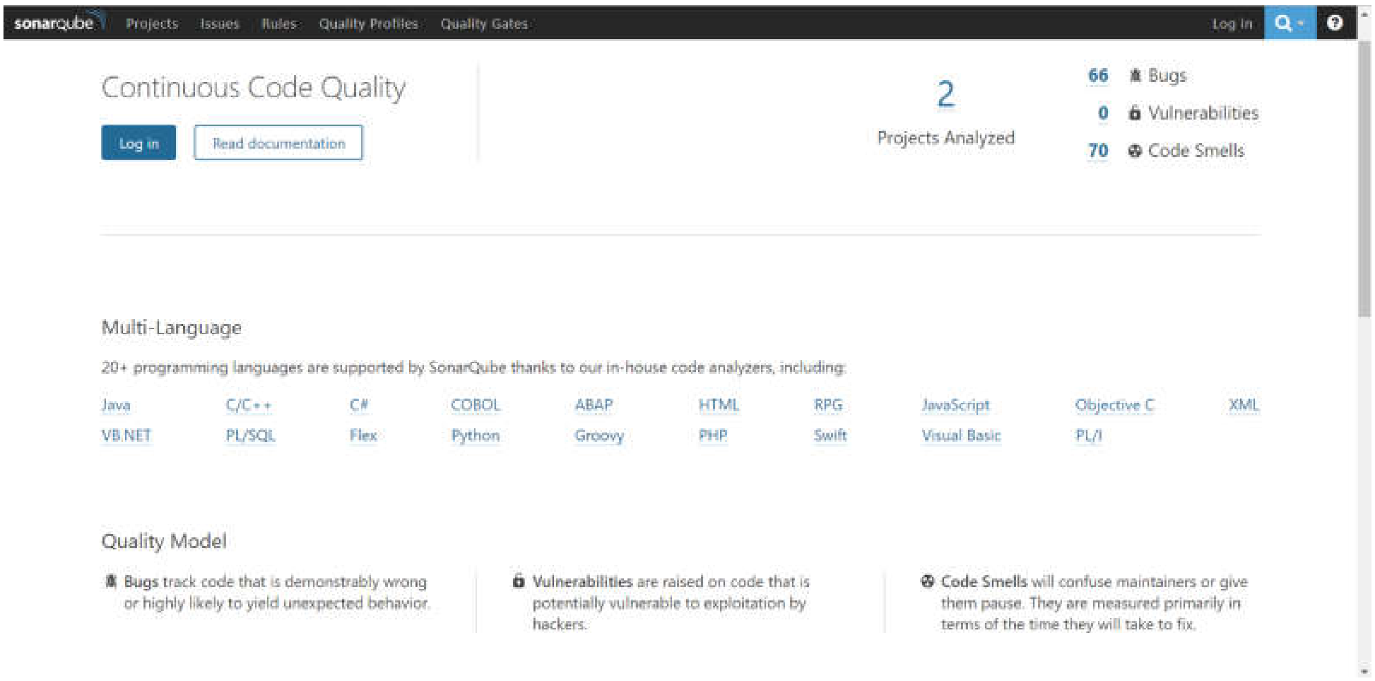
Integrate Jenkins with sonar

1. Go to the Jenkins dashboard and click on Manage Jenkins. Go to Manage Plugins and in the Available tab find the SonarQube plugin.
2. Click on Install without restart:



Enabled	Name ↓	Version	Previously installed version	Uninstall
✓	jQuery plugin This allows other plugins to use jQuery in UI.	1.11.2-0		Uninstall
✓	Maven Integration plugin This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTs, automated configuration of various Jenkins publishers (Junit, ...).	2.15.1		Uninstall
✓	Pipeline: Groovy Pipeline execution engine based on continuation passing style transformation of Groovy scripts.	2.34	Downgrade to 2.30	Uninstall
✓	Sonar Quality Gates Plugin Fails the build whenever the Quality Gates criteria in the Sonar 5.6+ analysis aren't met (the project Quality Gates status is different than "Passed")	1.0.4		Uninstall
✓	SonarQube Scanner for Jenkins This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	2.6.1		Uninstall

1. Go to the Jenkins dashboard and click on Manage Jenkins.
2. Click on Configure system and find the SonarQube section.
3. Now, let’s go to SonarQube to get the token to integrate Jenkins and SonarQube.
4. Once SonarQube is up and running, open the browser at http://localhost:9000 to visit the SonarQube dashboard:



Continuous Code Quality

2 Projects Analyzed

66 Bugs
0 Vulnerabilities
70 Code Smells

Multi-Language

20+ programming languages are supported by SonarQube thanks to our in-house code analyzers, including:

Java, C/C++, C#, COBOL, ABAP, HTML, RPG, JavaScript, Objective C, XML, VB.NET, PL/SQL, Flex, Python, Groovy, PHP, Swift, Visual Basic, PL/I

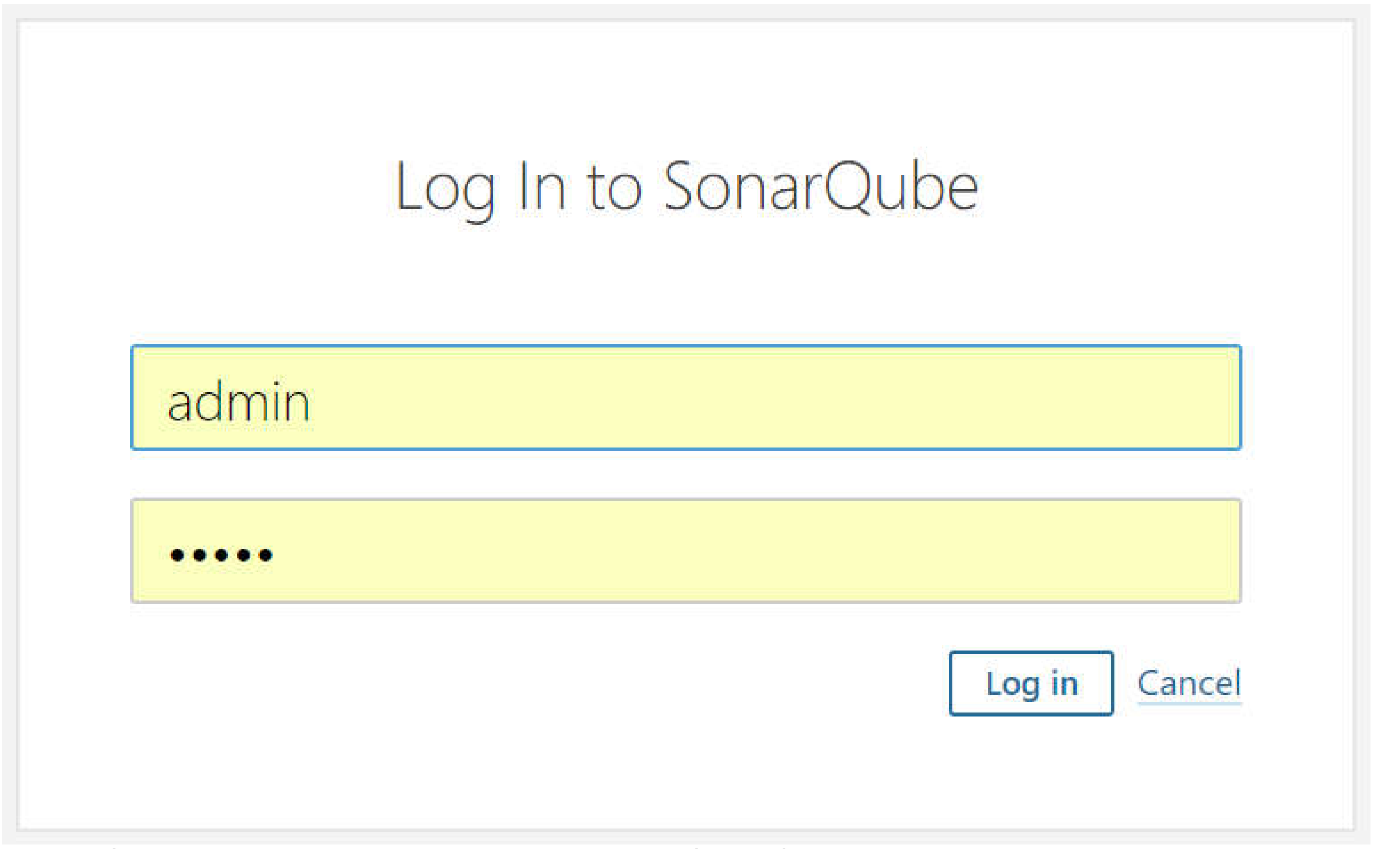
Quality Model

Bugs track code that is demonstrably wrong or highly likely to yield unexpected behavior.

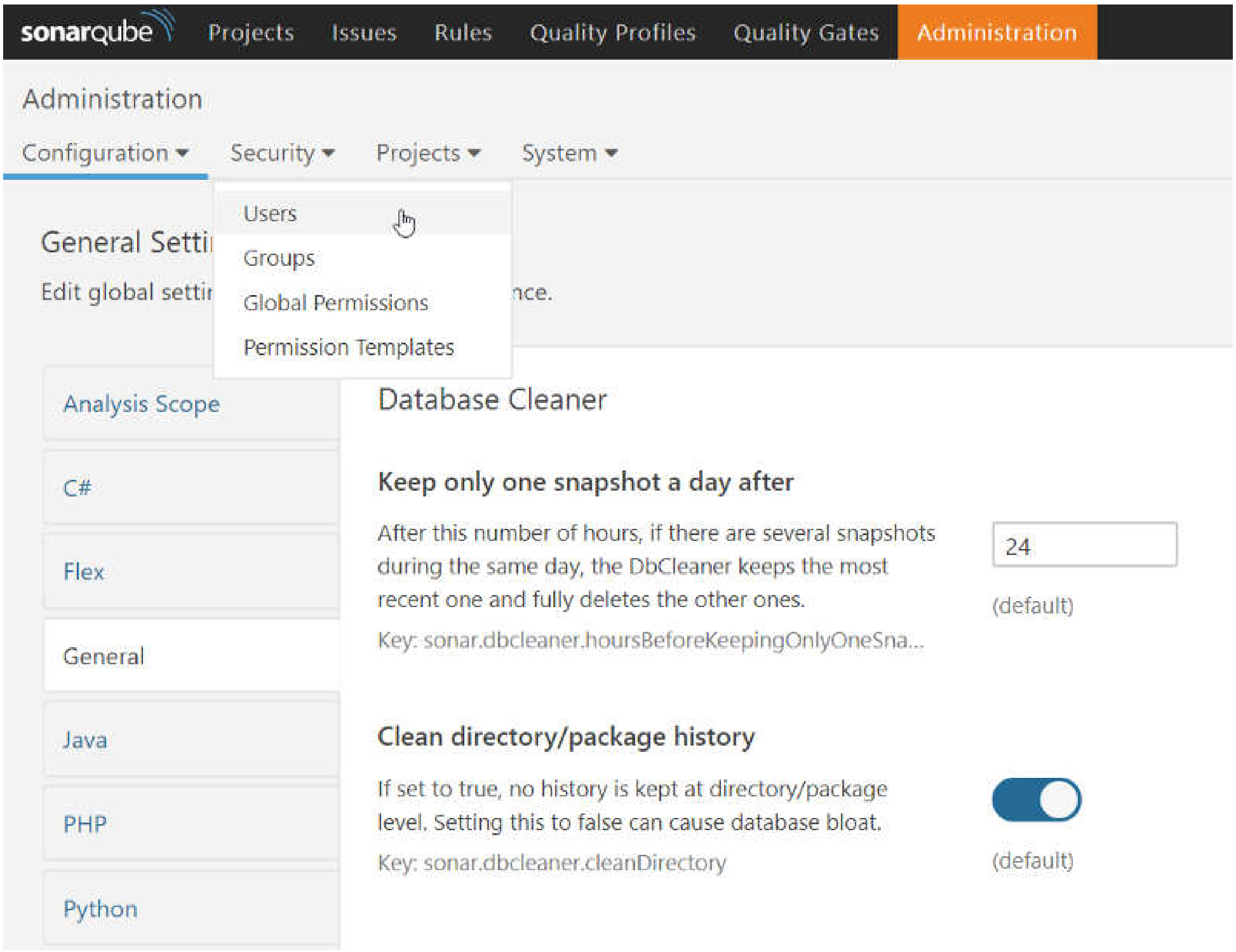
Vulnerabilities are raised on code that is potentially vulnerable to exploitation by hackers.

Code Smells will confuse maintainers or give them pause. They are measured primarily in terms of the time they will take to fix.

1. Click on Login and give the default username and password as admin and default to log in as an administrator.
2. Click on Login:



1. As of now, there is no project available in the SonarQube dashboard.
2. Click on the Administration tab and in the Security menu click on Users:



Database Cleaner

Keep only one snapshot a day after

After this number of hours, if there are several snapshots during the same day, the DbCleaner keeps the most recent one and fully deletes the other ones.

Key: sonar.dbcleaner.hoursBeforeKeepingOnlyOneSna...

24 (default)

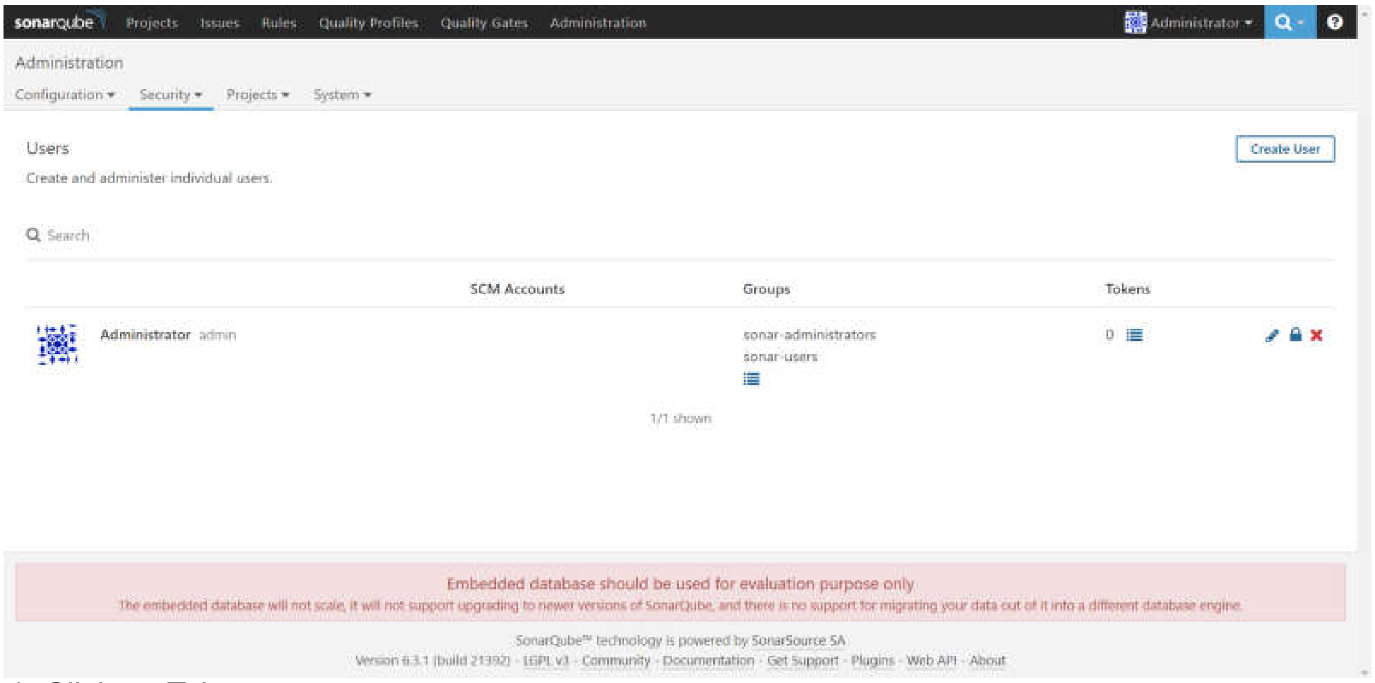
Clean directory/package history

If set to true, no history is kept at directory/package level. Setting this to false can cause database bloat.

Key: sonar.dbcleaner.cleanDirectory

(default)

1. Initially, there are no tokens issued; there is a 0 token for Administrator:



Administration

Users

Create and administer individual users.

	SCM Accounts	Groups	Tokens
Administrator admin		sonar-administrators sonar-users	0

1/1 shown

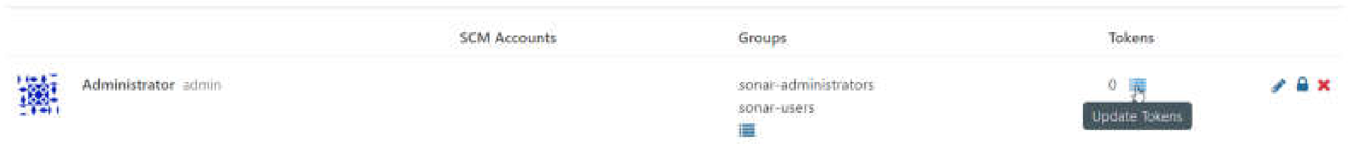
Embedded database should be used for evaluation purpose only

The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA

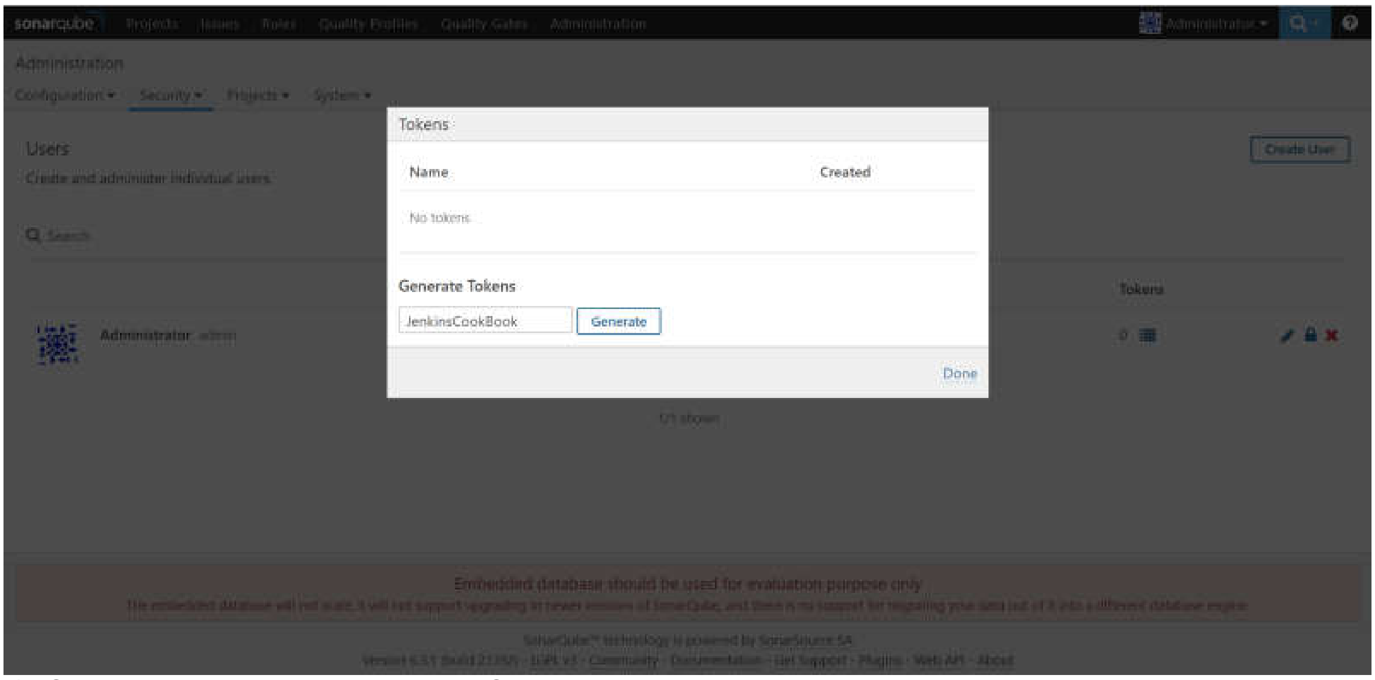
Version 6.3.1 (build 21392) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

1. Click on Tokens:



	SCM Accounts	Groups	Tokens
Administrator admin		sonar-administrators sonar-users	0

1. Give a name in the Generate Tokens section and click on Generate:

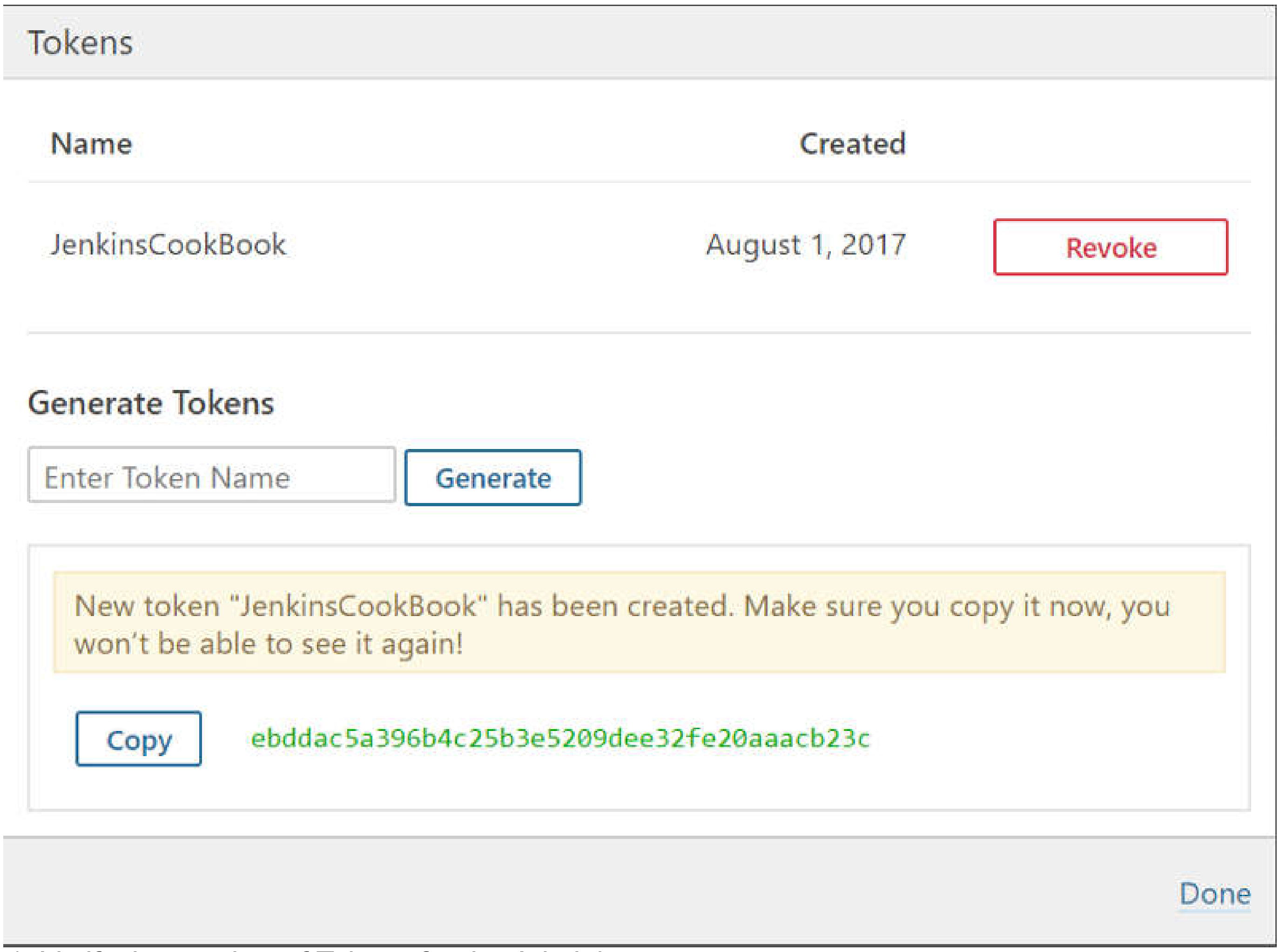


Tokens

Name	Created
No tokens	

Generate Tokens

1. Copy the newly created token. Click on Done:


Tokens

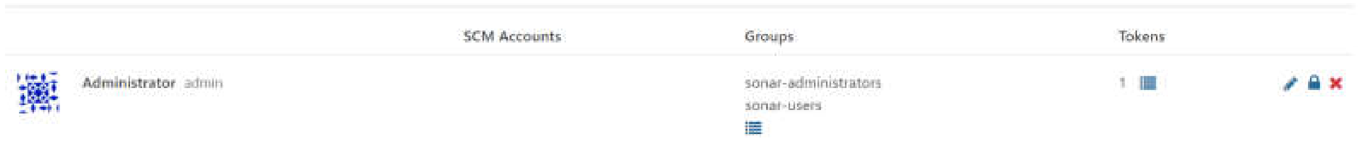
Name	Created	
JenkinsCookBook	August 1, 2017	Revoke

Generate Tokens

New token "JenkinsCookBook" has been created. Make sure you copy it now, you won't be able to see it again!

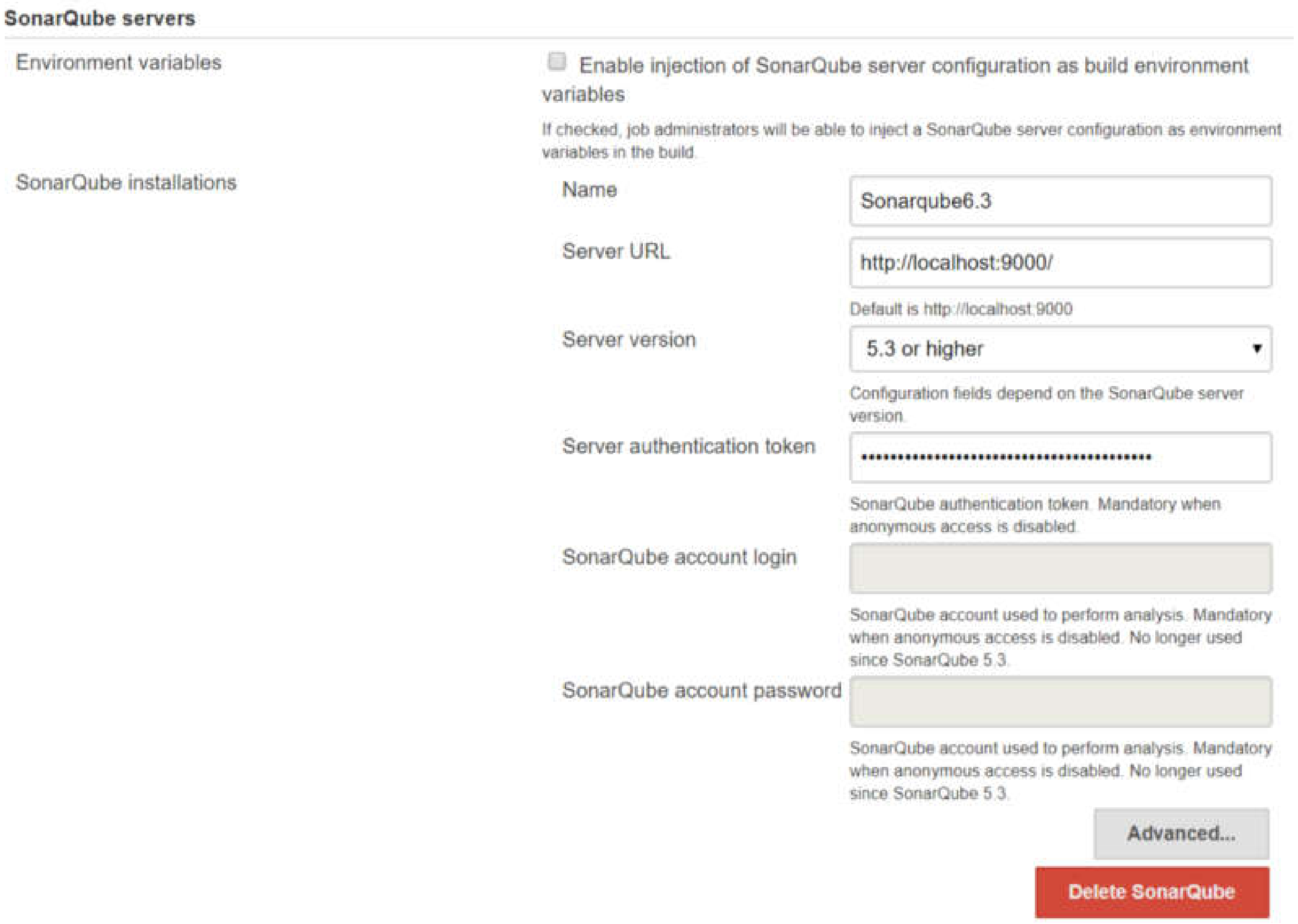
ebddac5a396b4c25b3e5209dee32fe20aaacb23c

1. Verify the number of Tokens for the Administrator user:



	SCM Accounts	Groups	Tokens
Administrator admin		sonar-administrators sonar-users	1

1. Now we have all the required parameters to integrate Jenkins and SonarQube:
2. Go to the Jenkins dashboard and click on Manage Jenkins.
3. Click on Configure system and find the SonarQube section.
4. Click on Add SonarQube.
5. Provide the Name, Server URL, and Server version.
6. Paste the token value in Jenkins and save it:


SonarQube servers

Environment variables

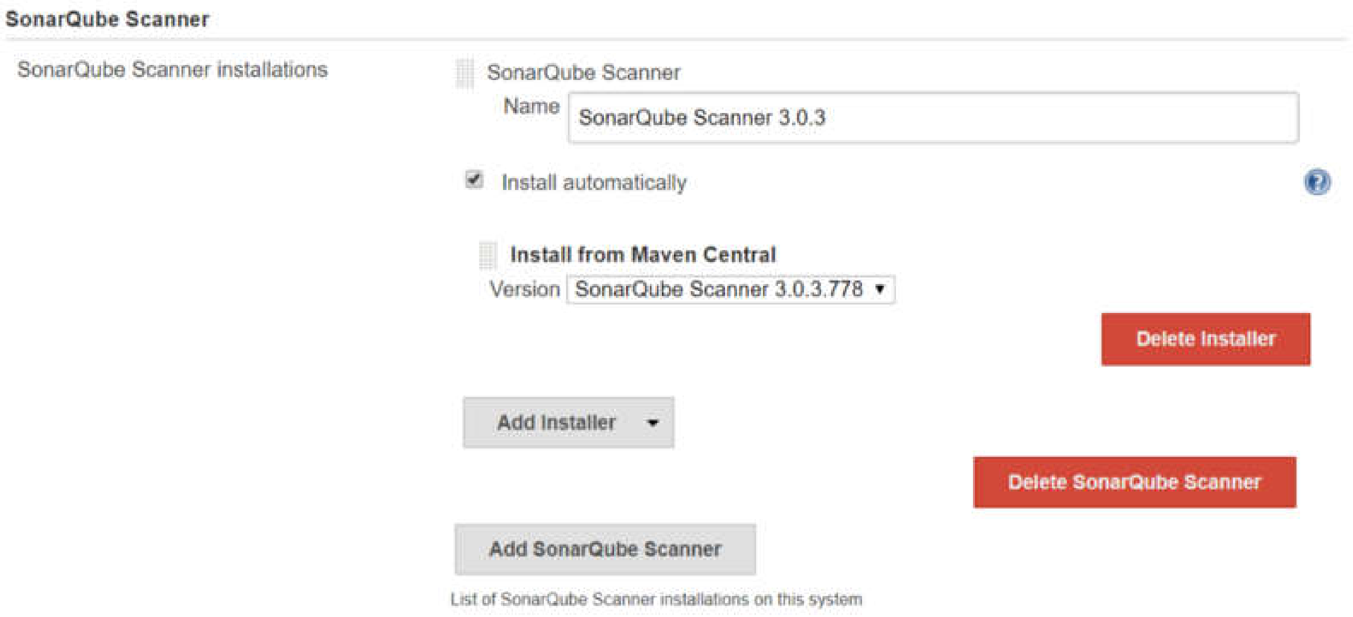
☐ Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name	Sonarqube6.3
Server URL	http://localhost:9000/
	Default is http://localhost:9000
Server version	5.3 or higher
	Configuration fields depend on the SonarQube server version.
Server authentication token	••
	SonarQube authentication token. Mandatory when anonymous access is disabled.
SonarQube account login	
	SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.
SonarQube account password	
	SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.

1. Go to Global Tool Configuration and configure Add SonarQube Scanner:


SonarQube Scanner

SonarQube Scanner installations

SonarQube Scanner

Name: SonarQube Scanner 3.0.3

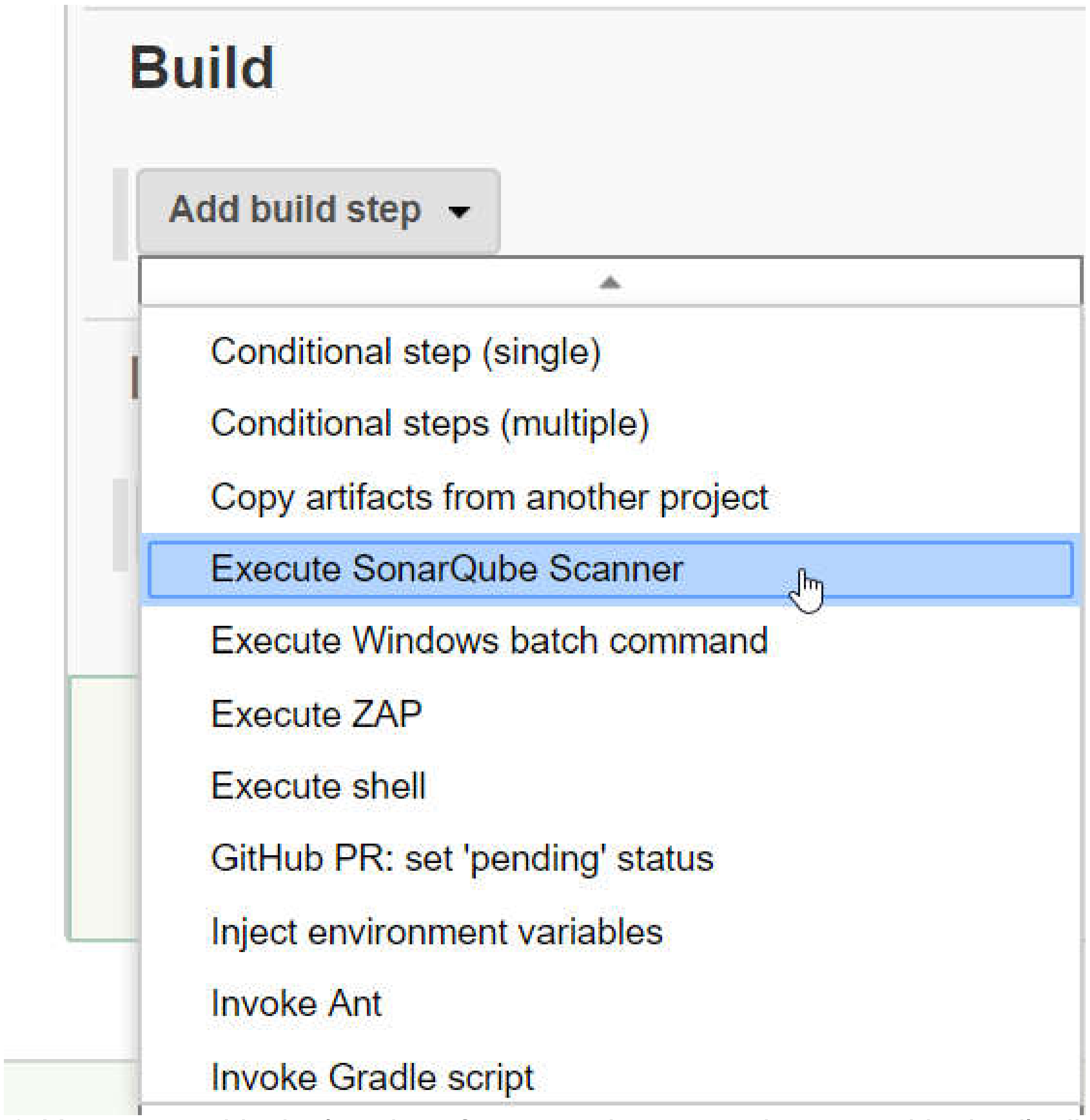
✓ Install automatically

Install from Maven Central

Version: SonarQube Scanner 3.0.3.778

List of SonarQube Scanner installations on this system

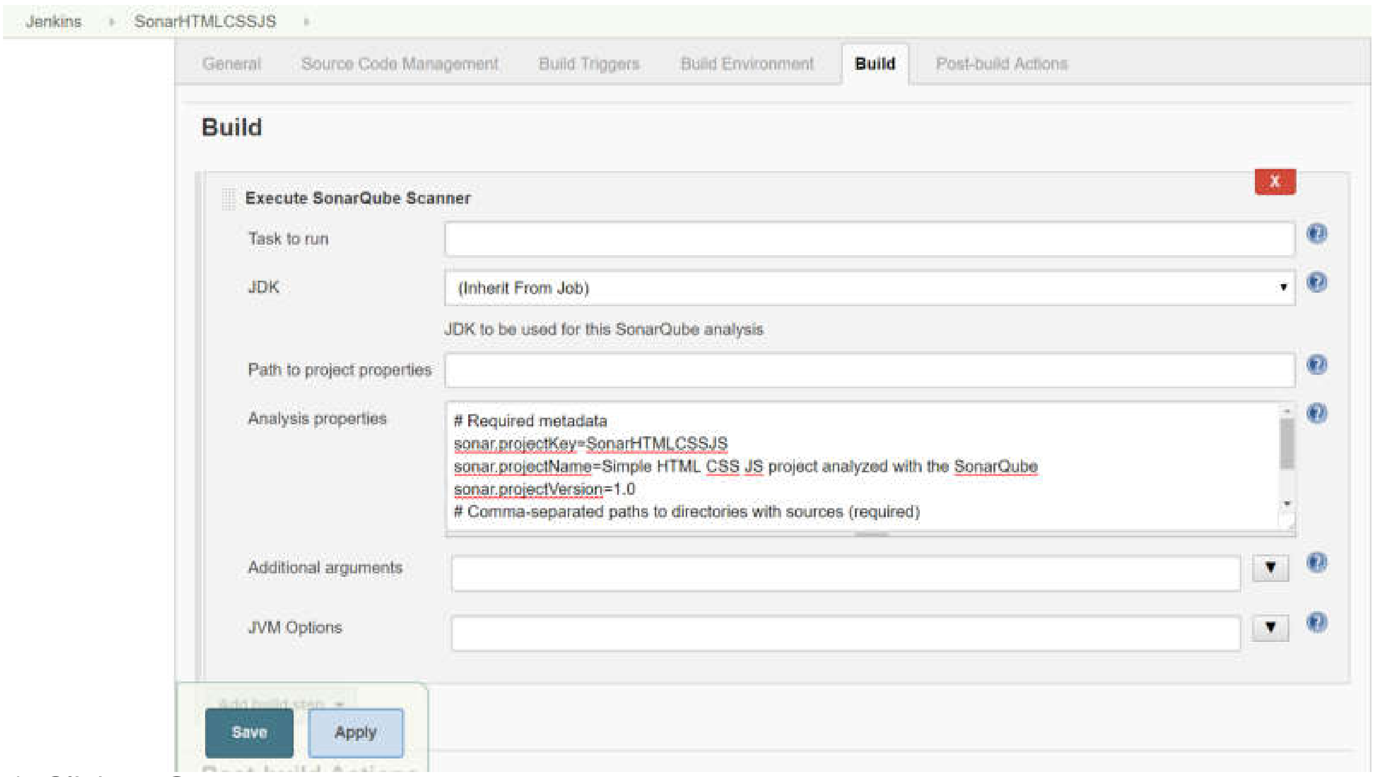
1. Now, you are ready for the static code analysis of the project.
2. Go to the Build section and select Execute SonarQube Scanner:


Build
Add build step
Conditional step (single)
Conditional steps (multiple)
Copy artifacts from another project
Execute SonarQube Scanner
Execute Windows batch command
Execute ZAP
Execute shell
GitHub PR: set 'pending' status
Inject environment variables
Invoke Ant
Invoke Gradle script

1. You can provide the location of sonar-project.properties or provide details directly for static code analysis.

```
# Required metadata
sonar.projectKey=SonarHTMLCSSJS
sonar.projectName=Simple HTML CSS JS project analyzed with the SonarQube
sonar.projectVersion=1.0
# Comma-separated paths to directories with sources (required)
sonar.sources=.
# Encoding of the source files
sonar.sourceEncoding=UTF-8
  sonar.java.binaries=.
```

1. sonar.sources is the main property for static code analysis. With this property, you inform SonarQube which directory needs to be analyzed:



Build

Execute SonarQube Scanner

Task to run:

JDK: (Inherit From Job)

JDK to be used for this SonarQube analysis

Path to project properties:

Analysis properties:

```
# Required metadata
sonar.projectKey=SonarHTMLCSSJS
sonar.projectName=Simple HTML CSS JS project analyzed with the SonarQube
sonar.projectVersion=1.0
# Comma-separated paths to directories with sources (required)
```

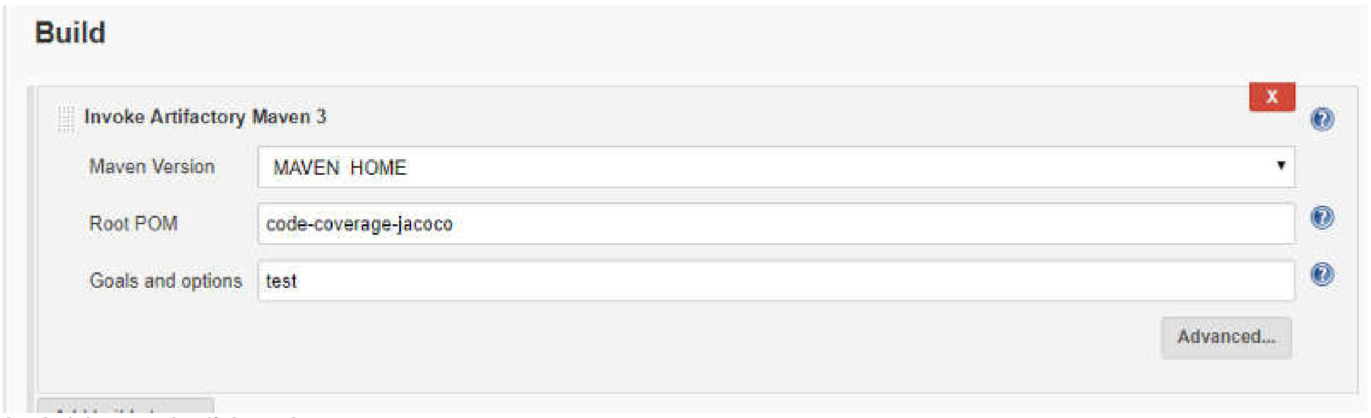
Additional arguments:

JVM Options:

1. Click on Save.
2. Go to Jenkins Project and click on Build now.
3. Go to Console output to check the logs.

Integrate Jacoco plugin with Maven

1. Install Jacoco plugin
2. Manage Jenkins -> Manage Plugin -> Search for Jacoco
3. Create a freestyle project in Jenkins
4. Use this link in source control amangement “https://github.com/pkainulainen/maven-examples.git”
5. You have to add all configuration in pom.xml as present in the code



Build

Invoke Artifactory Maven 3

Maven Version: MAVEN_HOME

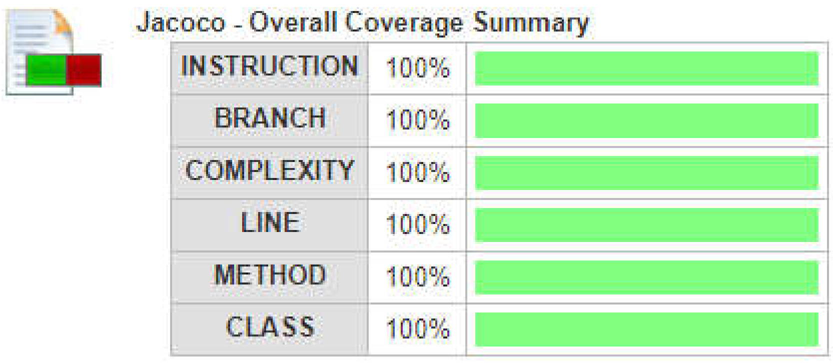
Root POM: code-coverage-jacoco

Goals and options: test

6. Add post build actions

7 Select “Record Jacoco Coverage report”

8. Jenkins build logs as shown below


Jacoco - Overall Coverage Summary

INSTRUCTION	100%	🟩
BRANCH	100%	🟩
COMPLEXITY	100%	🟩
LINE	100%	🟩
METHOD	100%	🟩
CLASS	100%	🟩

Configure Jenkins with sample spring boot project using Gradle/Maven

Please refer this link to generate a sample project http://start.spring.io/

All executable(.sh) file should have this permission.

For Gradle, any .sh file should have the below permission else while executing permission denied error comes

git update-index—chmod=+x gradlew

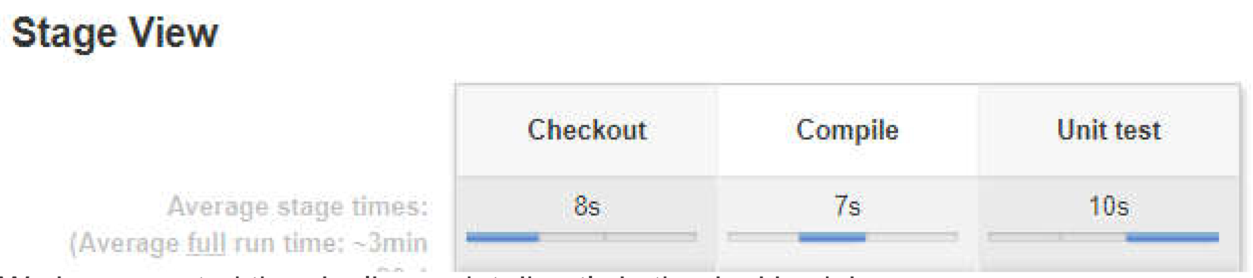
Pipeline Project Using Gradle

Create a Pipeline project in Jenkins and put the below code in the pipeline script and trigger the build

```
pipeline {
agent any
stages {
stage(“Checkout”) {
steps {
git url: ‘https://github.com/nidhigupta12/calculator.git’
}
}
stage(“Compile”) {
steps {
sh “./gradlew compileJava”
}
}
stage(“Unit test”) {
steps {
sh “./gradlew test”
}
}
}
}
```

After build is successful, you could see the below output view

```
./gradlew bootRun
```


Stage View

	Checkout	Compile	Unit test
Average stage times: (Average full run time: ~3min	8s	7s	10s

We have created the pipeline script directly in the Jenkins job.

Now we will see how to create the Jenkinsfile and commit it with the source code into the git repository.

Jenkinsfile

Let’s create a file called Jenkinsfile in the root directory of our project

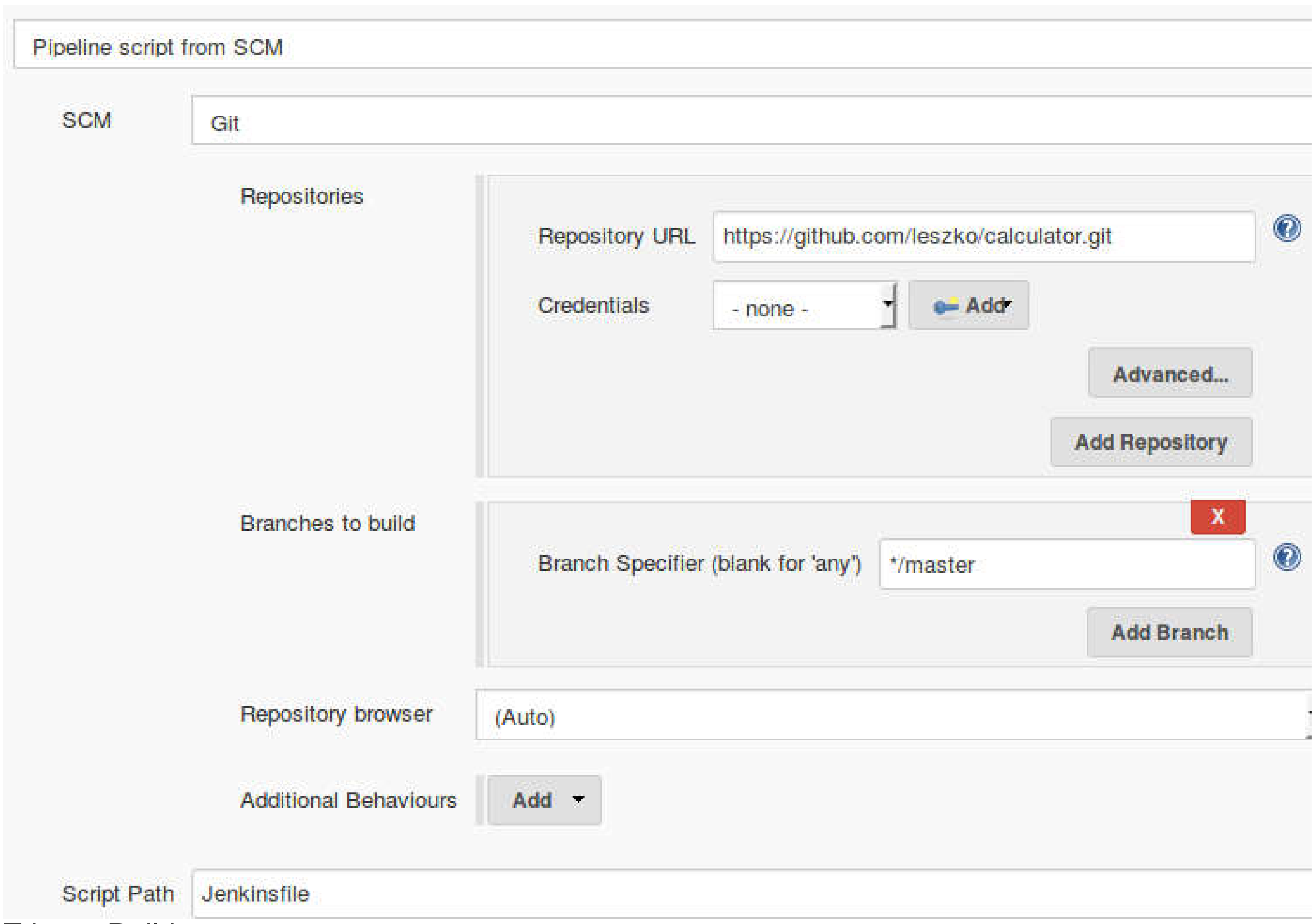
```
pipeline {
agent any
stages {
stage(“Compile”) {
steps {
sh “./gradlew compileJava”
}
}
stage(“Unit test”) {
steps {
sh “./gradlew test”
}
}
}
}
```

```
$ git add .
$ git commit -m "Add sum Jenkinsfile"
$ git push
```

Running pipeline from Jenkinsfile

When Jenkinsfile is in the repository, then all we have to do is to open the pipeline configuration and in the Pipeline section:

- Change Definition from Pipeline script to Pipeline script from SCM
- Select Git in SCM
- Put https://github.com/nidhigupta12/calculator.git in Repository URL


Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/leszko/calculator.git

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Trigger Build.

Code Coverage

Code coverage is a tool that runs all tests and verifies which parts of the code have been executed. Then, it creates a report showing not-tested sections. Moreover, we can make the build fail when there is too much untested code.

JACOCO

1. Add JaCoCo to the Gradle configuration.
2. Add the code coverage stage to the pipeline.
3. Optionally, publish JaCoCo reports in Jenkins.

In order to run JaCoCo from Gradle, we need to add the jacoco plugin to the build.gradle file by adding the following line in the plugin section:

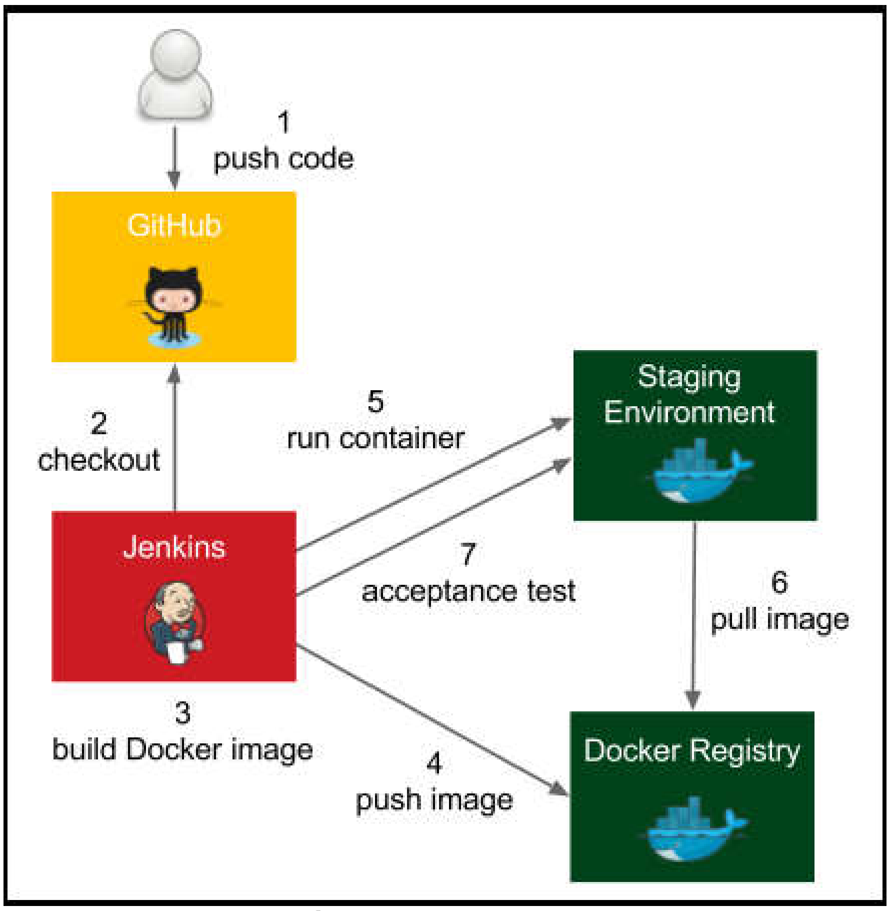
```
apply plugin: "jacoco"
```

Publishing report directly on Jenkins is not working.

```
No such DSL method 'publishHTML'
```

```
stage("Code coverage") {
     steps {
          sh "./gradlew jacocoTestReport"
          publishHTML (target: [
               reportDir: 'build/reports/jacoco/test/html',
               reportFiles: 'index.html',
               reportName: "JaCoCo Report"
          ])
          sh "./gradlew jacocoTestCoverageVerification"
     }
}
```

Acceptance test in pipeline



The process goes as follows:

1. The developer pushes a code change to GitHub.
2. Jenkins detects the change, triggers the build, and checks out the current code.
3. Jenkins executes the commit phase and builds the Docker image.
4. Jenkins pushes the image to Docker registry.
5. Jenkins runs the Docker container in the staging environment.
6. Staging the Docker host needs to pull the image from the Docker registry.
7. Jenkins runs the acceptance test suite against the application running in the staging environment.

Adding a Dockerfile and commit in Git and and add docker build/push to the jenkins pipeline(Jenkinsfile)

In the root directory of the project, let’s create the acceptance_test.sh file:

```
#!/bin/bash
test $(curl localhost:8765/sum?a=1\&b=2) -eq 3
```

Jenkinsfile

```
pipeline {
     agent any
     stages {
          stage("Compile") {
               steps {
                    sh "./gradlew compileJava"
               }
          }
          stage("Unit test") {
               steps {
                    sh "./gradlew test"
               }
          }


stage("Package") {
     steps {
          sh "./gradlew build"
     }
}
```

```
stage("Docker build") {
     steps {

          sh "docker build -t nikhilnidhi/calculator_1 ."
     }
}
```

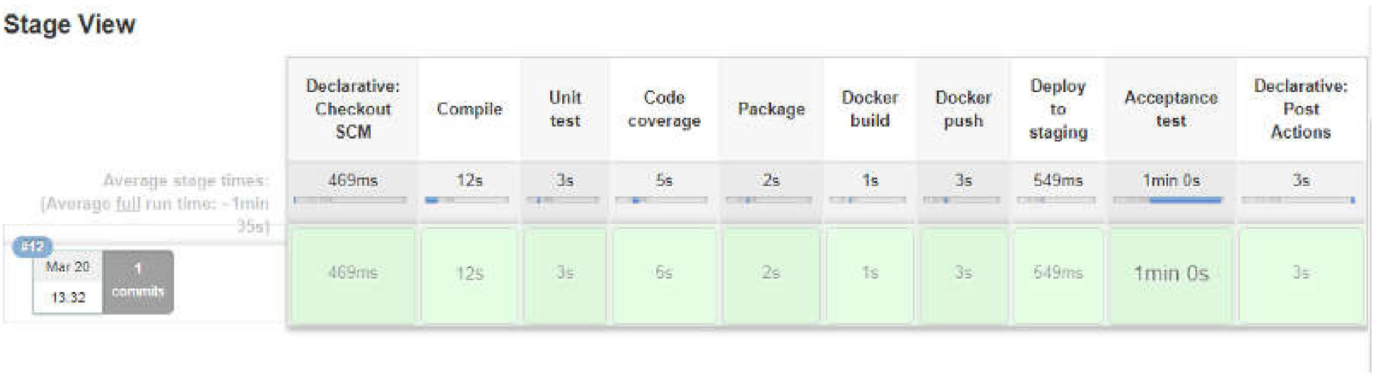
```
stage("Docker push") {
     steps {
```

```
    sh "docker login -u username -p password"
```

```
sh "docker push nikhilnidhi/calculator_1"
     }
}
stage("Deploy to staging") {
     steps {

          sh "docker run -d --rm -p 8765:8080 --name calculator_1
nikhilnidhi/calculator_1"
     }
}
```

```
stage("Acceptance test") {
     steps {
          sleep 60
          sh "./acceptance_test.sh"
     }
}
     }
  post {
     always {
          sh "docker stop calculator_1"
     }
}
}
```



Stage View

	Declarative: Checkout SCM	Compile	Unit test	Code coverage	Package	Docker build	Docker push	Deploy to staging	Acceptance test	Declarative: Post Actions
Average stage times: (Average full run time: ~1min 35s)	469ms	12s	3s	5s	2s	1s	3s	549ms	1min 0s	3s
#12 Mar 20 13:32 1 commits	469ms	12s	3s	6s	2s	1s	3s	549ms	1min 0s	3s

For Gradle

Create a freestyle project and use gradle

Configure Gradle -> Manage Jenkins->Global tools configuration->Gradle

and then in the job select Gradle version instead of default.

Setup using Docker-compose

Let’s start with an example and imagine that our calculator project uses the Redis server for caching. In this case, we need an environment with two containers, calculator and redis. let’s create the docker-compose.yml file at the same location.

```
version: "3"
services:
     calculator:
          image: calculator:latest
          ports:
               - 8080
     redis:
          image: redis:latest
```

References

Continuous Delivery with Docker and Jenkins. Click Here