

CI/CD using Jenkins and Docker

This Project 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

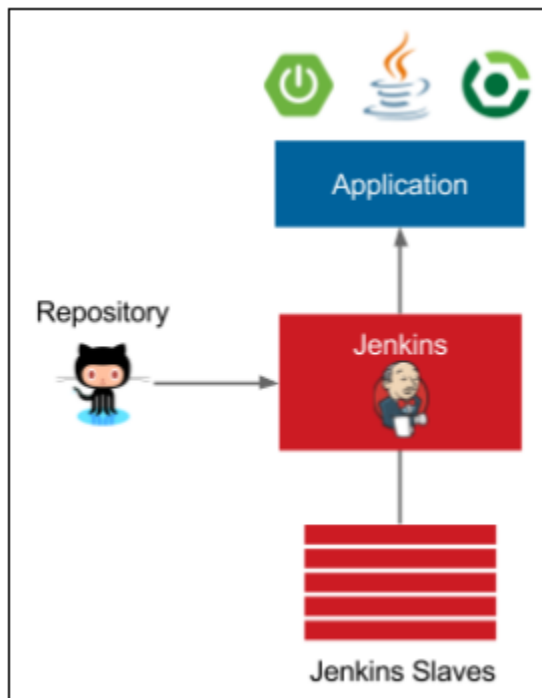
Docker—Container

JaCOCO—Code Coverage Tool

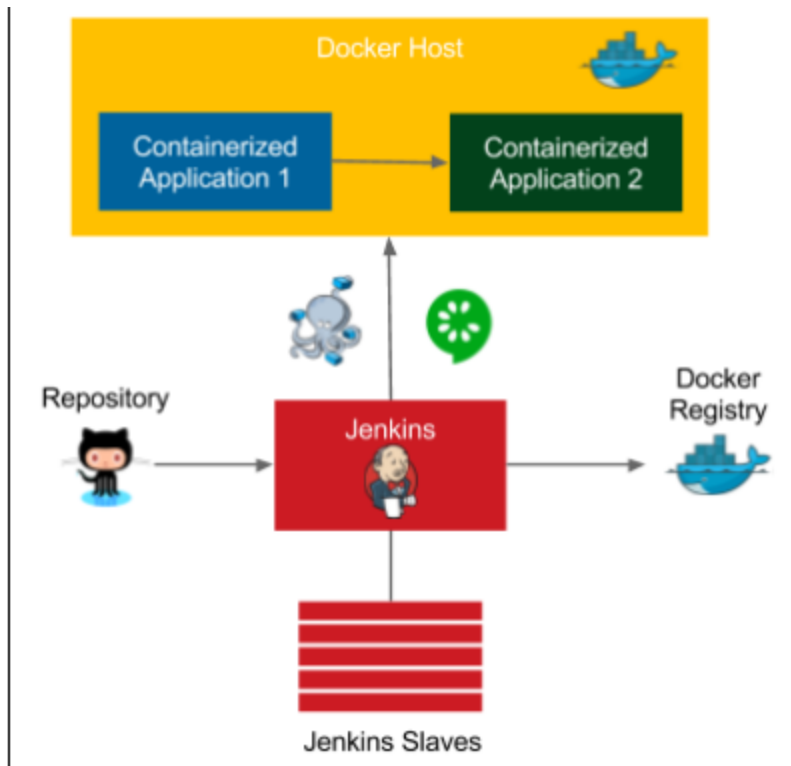
Gradle—Build tool

Ansible—Configuration Management

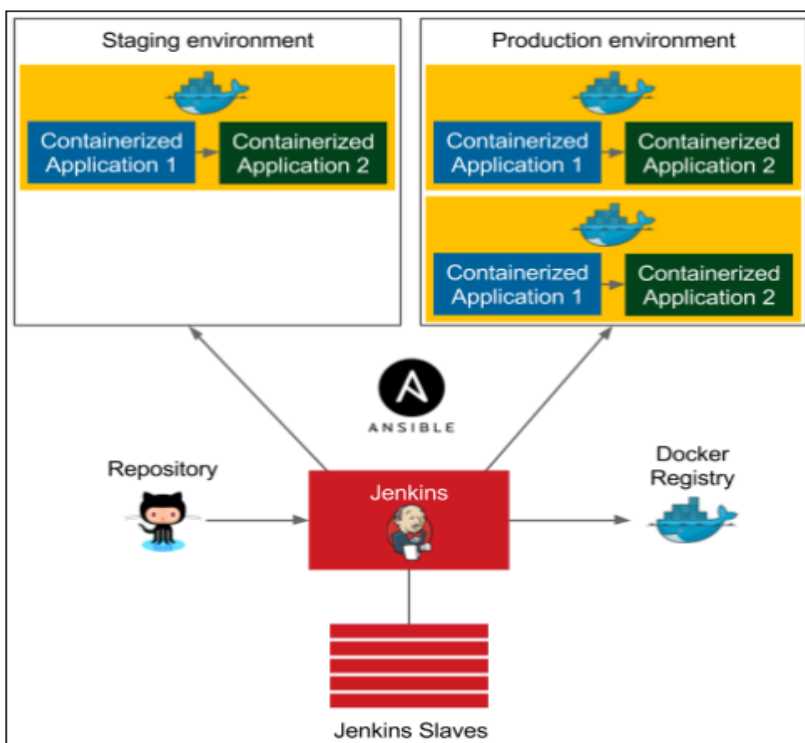
Ansible/Github/Docker/Cucumber/



Continuous Integration Pipeline

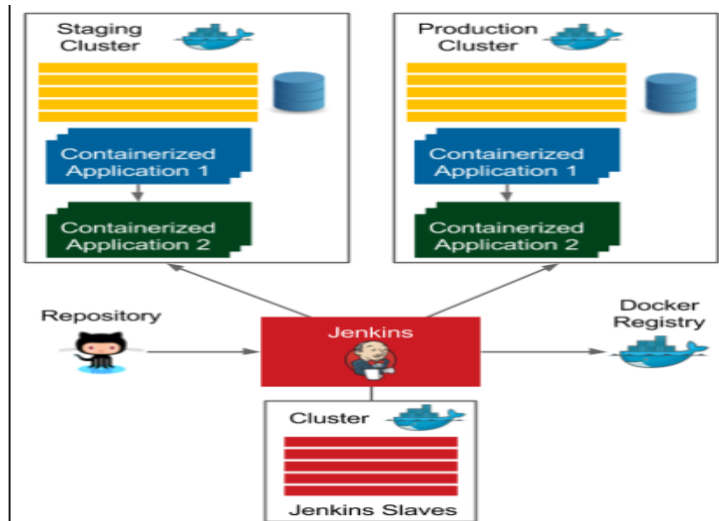


Configuration Management with Ansible



Continuous Delivery Pipeline

Clustering with Docker Swarm/Advanced Continuous Delivery



How To Install Jenkins In AWS EC2 Instance

How to install Jenkins in AWS EC2 instance

Hello Everyone

Welcome to [simplilearn](https://www.simplilearn.com) .

In this project we will explore one of the most popular CI/CD tool Jenkins. We will try to cover each aspect of Jenkins with a demo.

What is Jenkins?

Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Components of Jenkins:

- **Master Node:** The system where you install and run Jenkins.
- **Managed Node:** Target systems managed by Jenkins.
- **Repository:** Version controlled system where you keep your code.
- **UI:** User interface to manage and configure Jenkins.
- **Pipeline:** Means of continuous deployment of your code to target systems from the version control system.
- **Plugins:** Means of enhancing the functionality of a Jenkins.

Next, we are going to learn how to install Jenkins.

Jenkins Installation:

Prerequisite:

- One EC2 Amazon Linux 2 instance with internet access
- Java 8 runtime environments

```
1 #####
2 ## Jenkins installation on AWS EC2 ##
3 #####
4 ## Create an EC2 instance with Amazon Linux 2 AMI
5 ## Amazon Linux 2 AMI (HVM), SSD Volume Type
6 ## Connect to your EC2 instance
7
8 ## Update all packages
9 sudo yum update -y
10
11 ## Install Java
12 sudo yum install java-1.8.0-openjdk-devel
```

There are several ways you can install Jenkins in your master node (EC2 instance in our case).

- Using Docker
- Using Yum
- Using a War file

I would prefer the 1st method, using docker which comes with blue ocean plugin preinstalled and this will be used in rest of the Jenkins blog series.

Using docker:

```
1 #####
2 ## Jenkins installation on AWS EC2 using docker blue ocean ##
3 #####
4
5 ## Install Docker.
6 sudo yum install docker
7
8 ## Add the ec2-user to the docker group so you can execute Docker commands without using sudo.
9 ## Exit the terminal and re-login to make the change effective
10 sudo usermod -a -G docker ec2-user
11 exit
12
13 ## Enable docker service
14 sudo systemctl enable docker
15
16 ## Start docker service
17 sudo systemctl start docker
18
19 ## Check the Docker service.
20 sudo systemctl status docker
```

```

21
22 ## Run docker jenkins blueocean container
23 docker run \
24 -u root \
25 --rm \
26 -d \
27 -p 8080:8080 \
28 -p 50000:50000 \
29 --name myjenkin \
30 -v jenkins-data:/var/jenkins_home \
31 -v /var/run/docker.sock:/var/run/docker.sock \
32 jenkinsci/blueocean
33
34 ## get the administrator password
35 docker exec -it myjenkin bash
36 cat /var/jenkins_home/secrets/initialAdminPassword
37 exit
38
39 #or from docker logs
40
41 docker logs myjenkin

```

Using Yum:

```

1 #####
2 ## Jenkins installation on AWS EC2 using YUM ##
3 #####
4
5 ## Add Jenkins repo to your yum repository
6 sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat/jenkins.repo
7
8 ## Import a key file from Jenkins-CI to enable installation from the package
9 sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key
10
11 ## Install Jenkins
12 sudo yum install jenkins -y
13
14 ## Start and enable Jenkins service
15 sudo systemctl start jenkins
16 sudo systemctl enable jenkins
17 sudo systemctl status jenkins
18
19 ## Get the initial administrative password
20 sudo cat /var/lib/jenkins/secrets/initialAdminPassword

```

Using a WAR file:

```

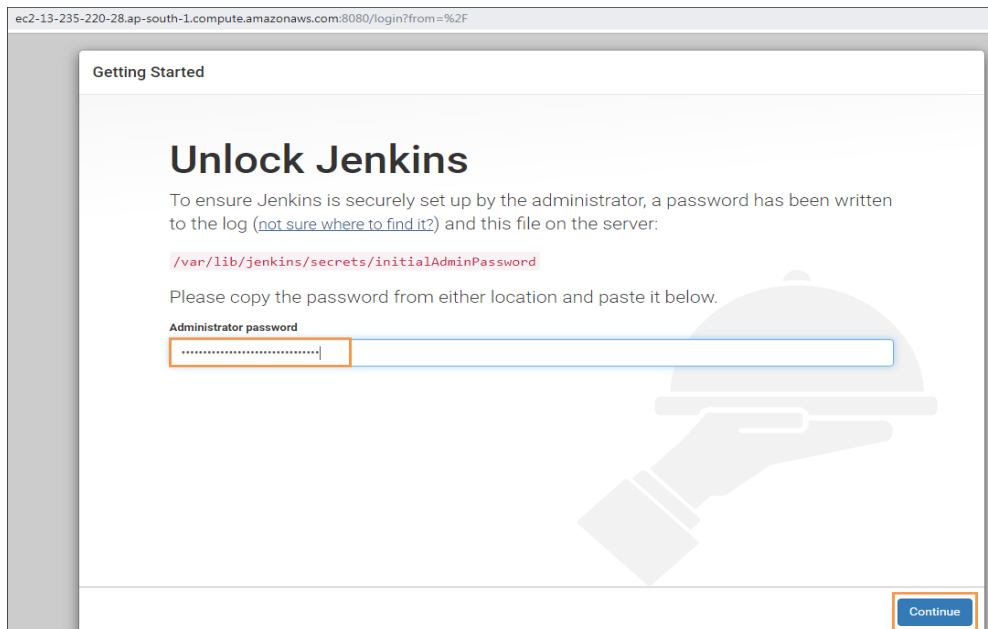
1 #####
2 ## Jenkins installation on AWS EC2 using war file##
3 #####
4
5 ## Get the latest jenkins war file
6 wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war
7
8 ## Install jenkins using war file
9 java -jar jenkins.war
10

```

11 ## Get the initial administrative password from the output of the above command
12 # Jenkins initial setup is required. An admin user has been created and a password generated.
13 # Please use the following password to proceed to installation:
14 # 2871112a4e704e3bbc9c2e0b2963c8c0
15 # This may also be found at: /home/ec2-user/.jenkins/secrets/initialAdminPassword

Jenkins Configuration:

Step 1: Open your EC2 instance public DNS or public IP (http://<PUBLIC_DNS/PUBLIC_IP>:8080/) along with port 8080 in your favorite browser. And provide the administrative password obtained during the installation.



ec2-13-235-220-28.ap-south-1.compute.amazonaws.com:8080/login?from=%2F

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

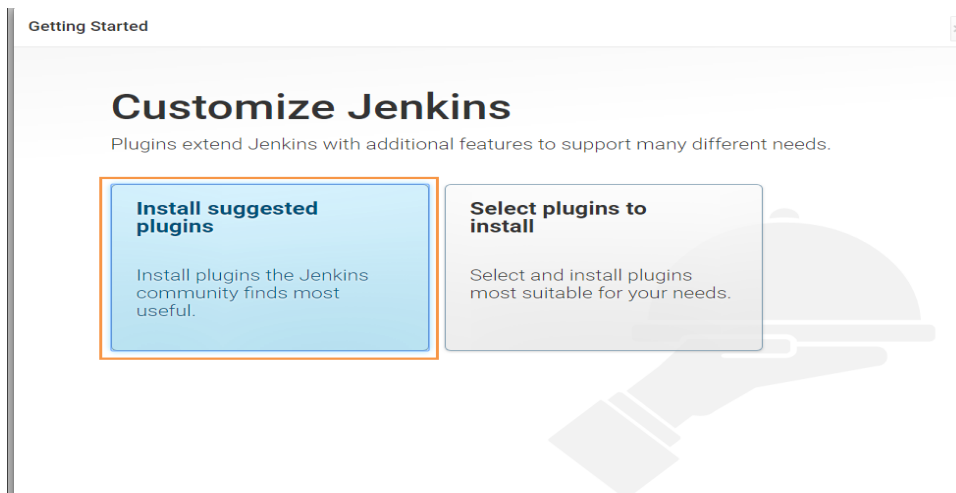
`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Continue

Step 2: Click 'Install suggested plugins'.



Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

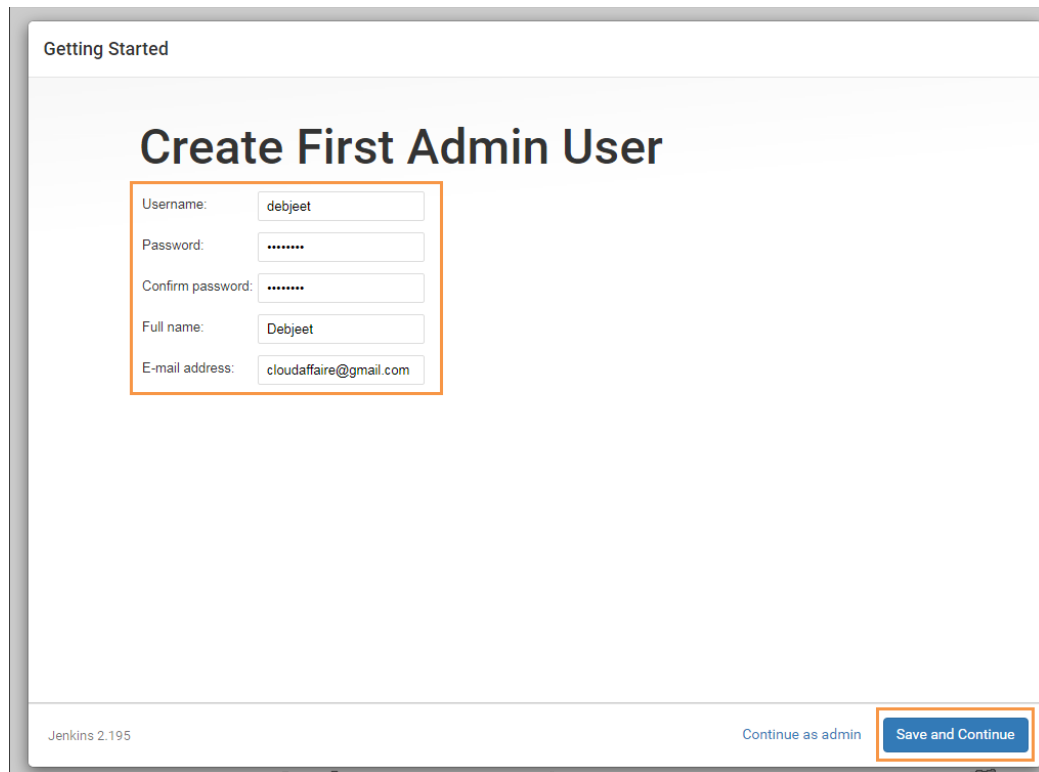
Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

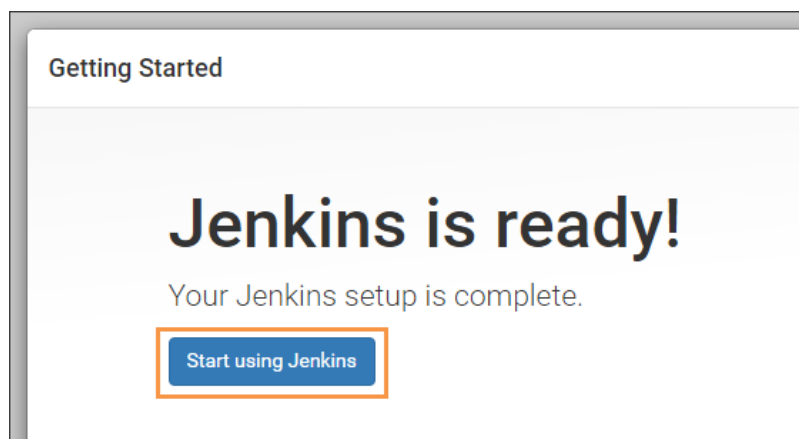
Continue

Step 3: Create an administrative user.

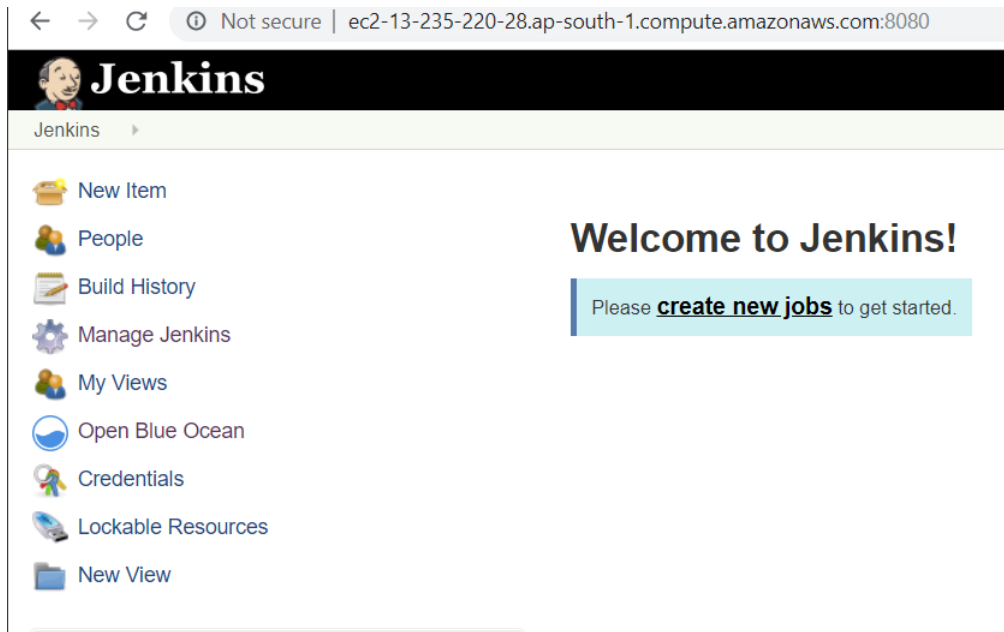


The screenshot shows the 'Getting Started' page in Jenkins 2.195. The main heading is 'Create First Admin User'. Below it is a form with five fields: 'Username' (filled with 'debjcet'), 'Password' (filled with seven dots), 'Confirm password' (filled with seven dots), 'Full name' (filled with 'Debjcet'), and 'E-mail address' (filled with 'cloudaffaire@gmail.com'). The form is outlined with an orange border. At the bottom right, there is a blue button labeled 'Save and Continue', also outlined with an orange border. At the bottom left, it says 'Jenkins 2.195'. In the center bottom, it says 'Continue as admin'.

Step 4: Our Jenkins configuration completed, click 'Start using Jenkins' to open the Jenkins UI.



The screenshot shows the 'Getting Started' page in Jenkins. The main heading is 'Jenkins is ready!'. Below it, the text says 'Your Jenkins setup is complete.' At the bottom, there is a blue button labeled 'Start using Jenkins', outlined with an orange border.



Hope you have enjoyed this article. To install Jenkins on another platform, you can refer below Jenkins documentation

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/>

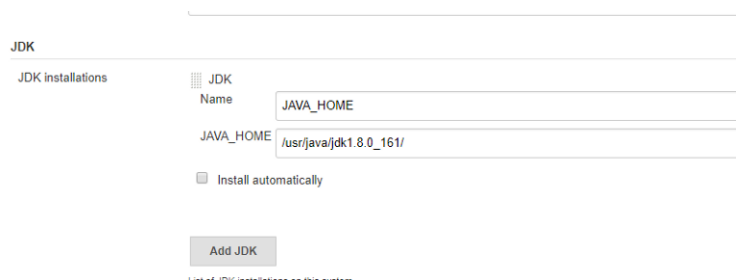
It will ask for the initial password, please run the below command

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
<div>Name</div> <div>JAVA_HOME</div> <div>JAVA_HOME</div> <div>/usr/java/jdk1.8.0_161/</div> <div><input type="checkbox"/> Install automatically</div>

Add JDK

1 out of 1 JDK installation on this machine

Add MAVEN -> Select Install automatically.

Maven

Maven installations

Maven

Name

☒ Install automatically

Install from Apache

Version

Create a freestyle project in Jenkins

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

Source Code Management

☐ None
☒ Git

Repositories

Repository URL:

Credentials: [Add](#)

[Advanced...](#)
[Add Repository](#)

Branches to build

Branch Specifier (blank for 'any'):

[Add Branch](#)

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:

Root POM:

Goals and options:

[Advanced...](#)

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.

For linux

Copy software from your machine to linux

Java should be present

Install JAVA 8

yum install wget (in case wget is not found)

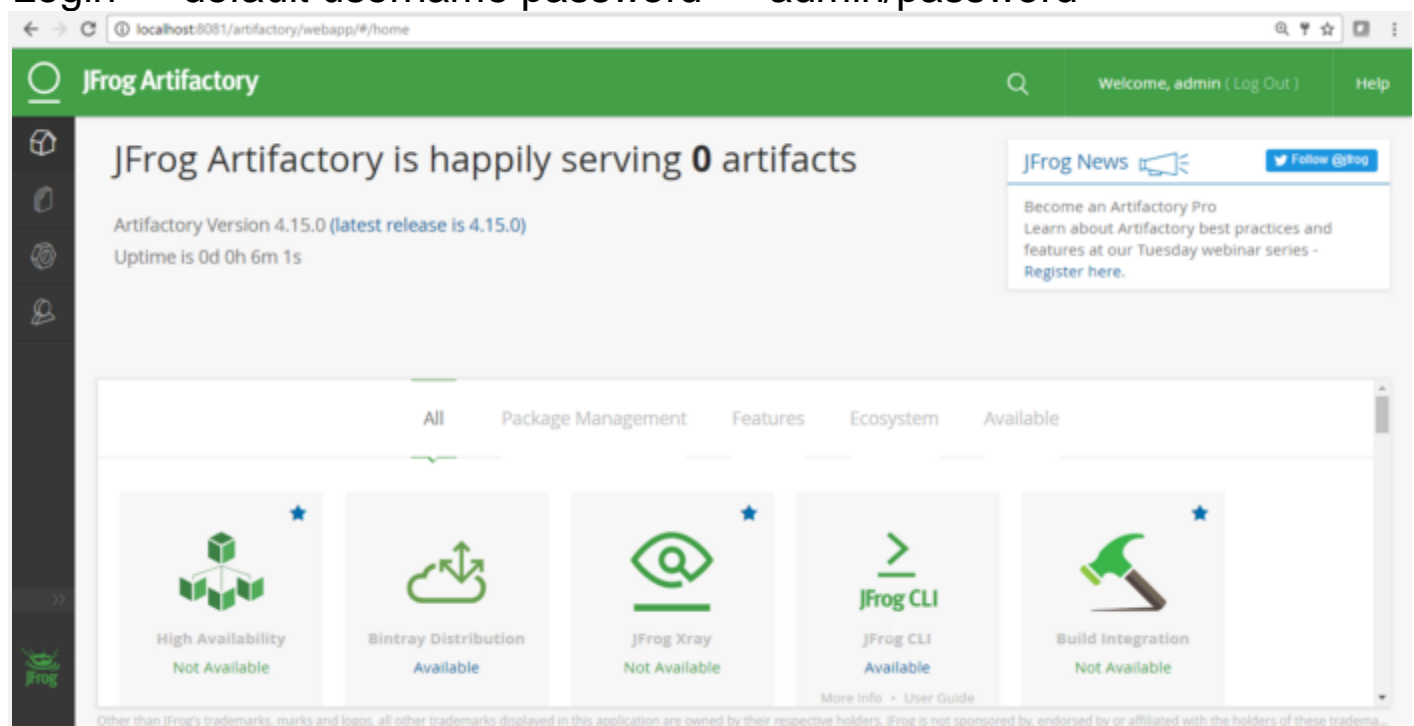
```
$ wget --header "Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u161-
b12/2f38c3b165be4555a1fa6e98c45e0808/jdk-8u161-linux-x64.rpm$ sudo yum localinstall
jdk-8u161-linux-x64.rpmexport JAVA_HOME=/usr/java/jdk1.8.0_161/
export
JRE_HOME=/usr/java/jdk1.8.0_161/jrePATH=$PATH:$HOME/bin:$JAVA_HOME/binexport
```

```
PATH
sudo alternatives --config java
```

Start artifactory.bat/.sh,present in bin folder

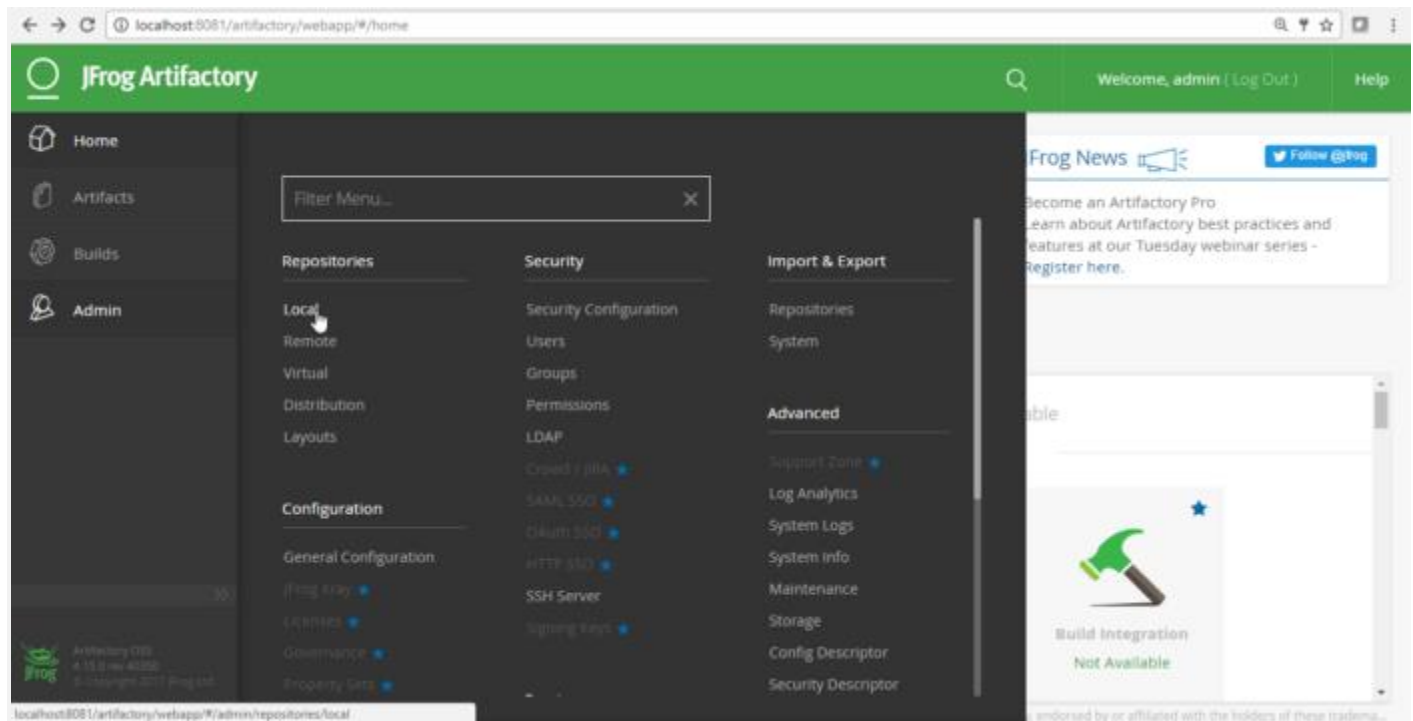
Go to your browser and visit IP_ADDRESS:8081 to visit Artifactory in the browser:

Login — default username password — admin/password

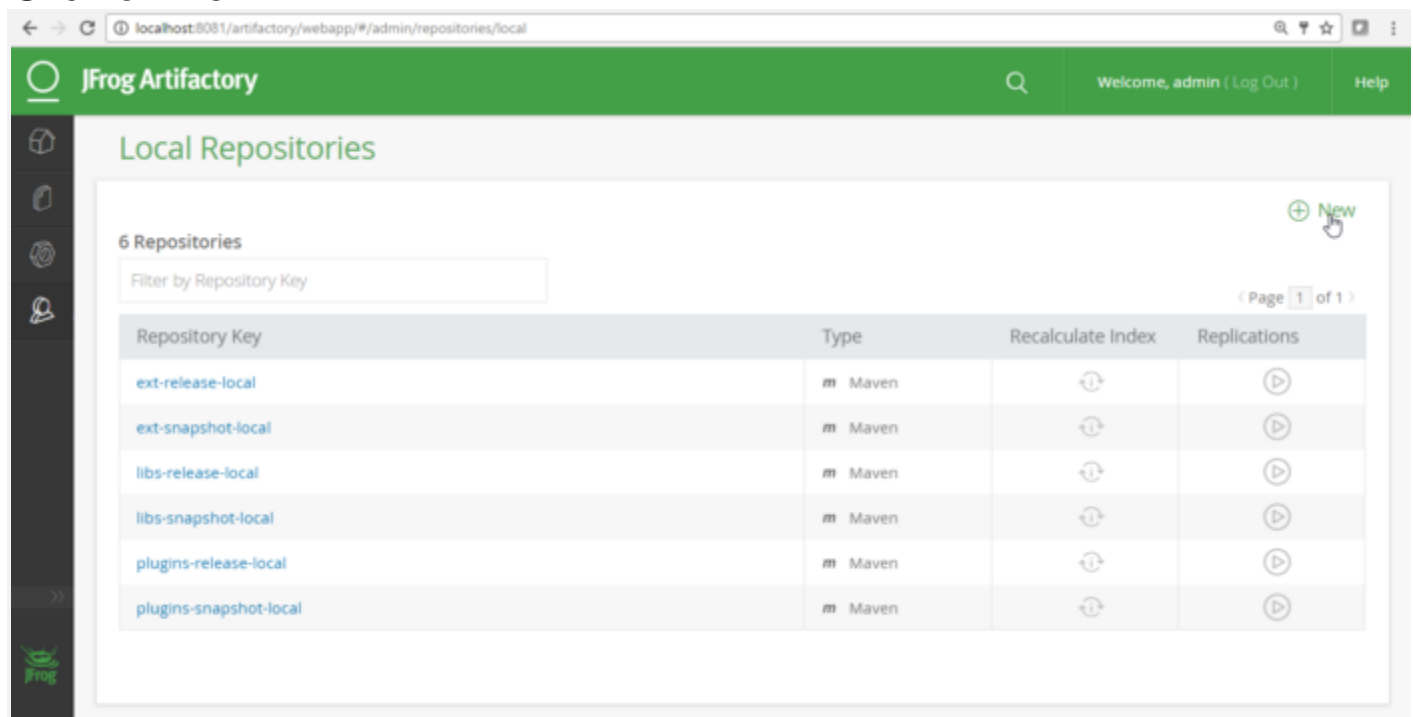


Create a Local repository to store package files created by the Jenkins project:

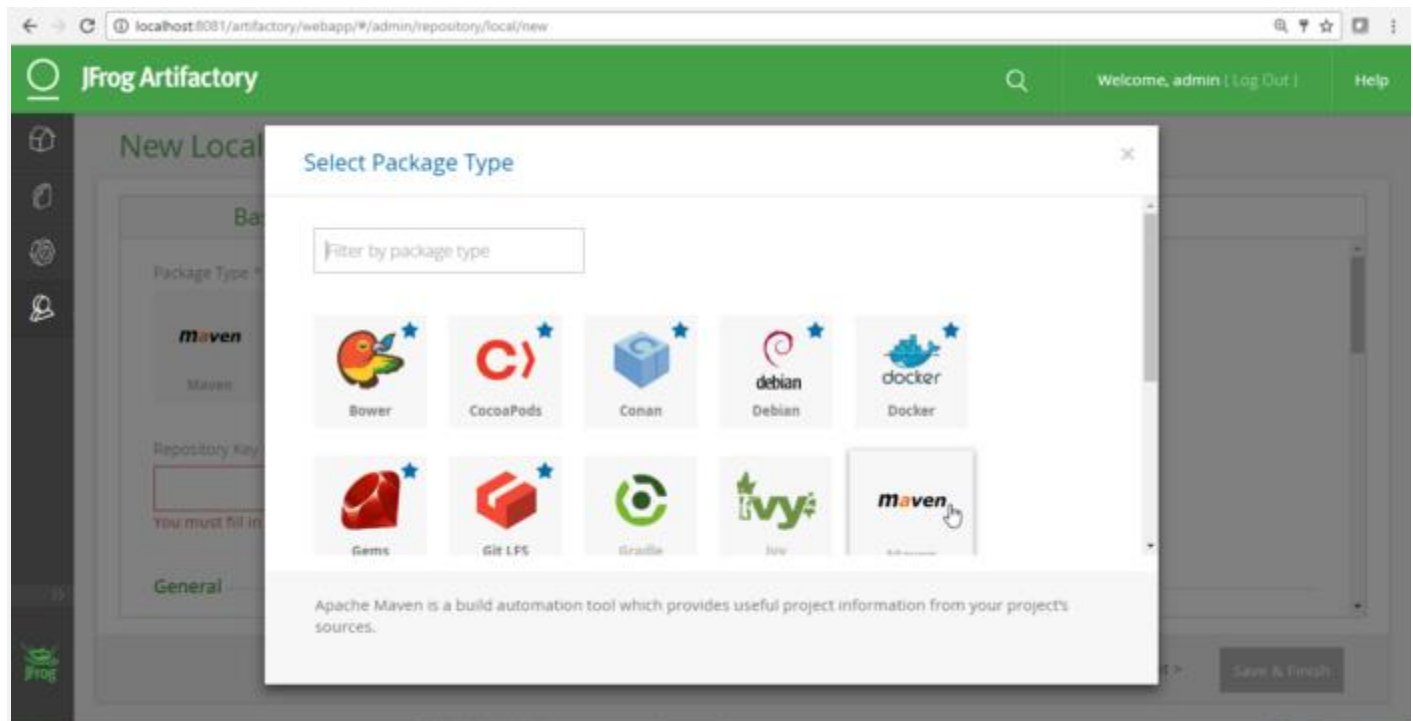
Click on Admin -> Repositories ->Local



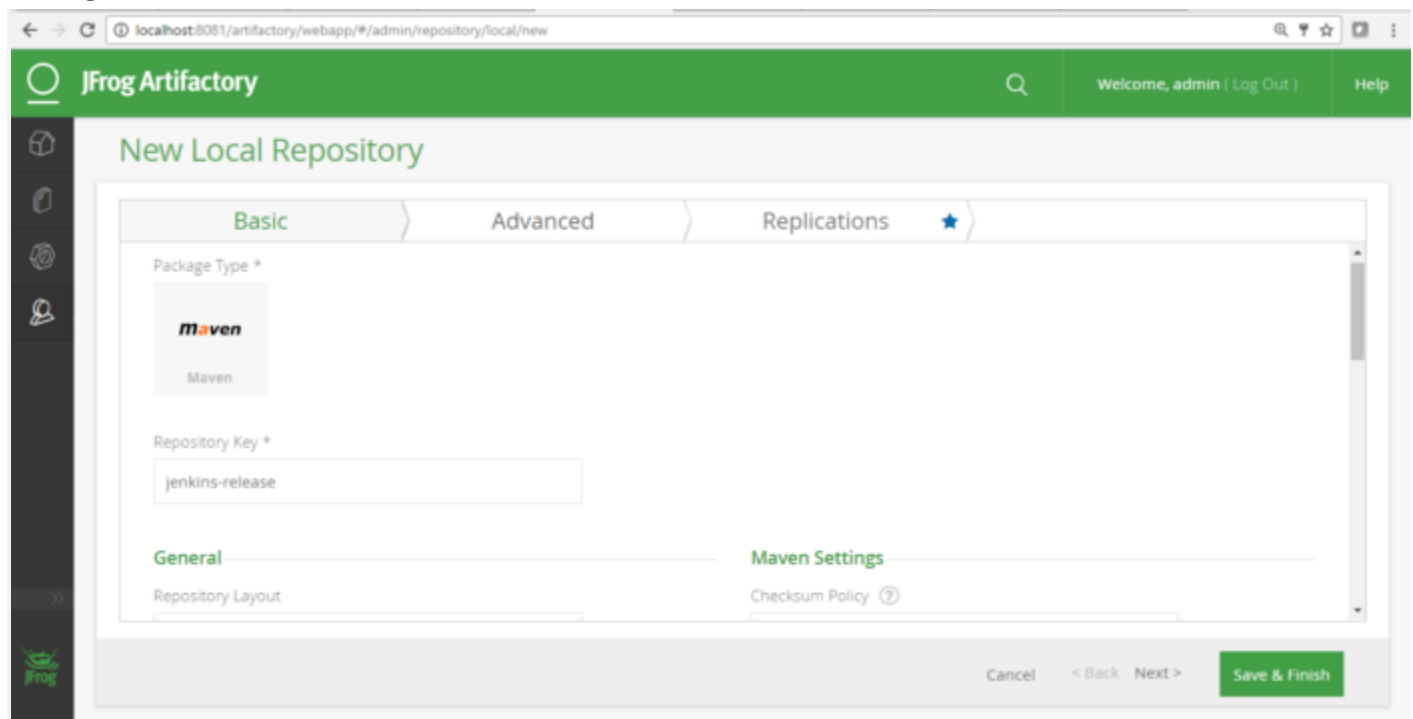
Click on New:



Select Maven as a Project Type:



Give a Repository Key “jenkins-release” and click on Save and Finish:



Similarly, create a Jenkins-snapshot repository:

The screenshot shows the 'New Local Repository' configuration page in JFrog Artifactory. The 'Basic' tab is selected, showing the 'Package Type' as 'Maven' and the 'Repository Key' as 'jenkins-snapshot'. The 'Save & Finish' button is highlighted with a mouse cursor.

Package Type *

Maven

Repository Key *

jenkins-snapshot

General

Repository Layout

Maven Settings

Checksum Policy ?

Cancel < Back Next > Save & Finish

Verify all repositories in the list:

The screenshot shows the 'Local Repositories' list in JFrog Artifactory. A success message 'Successfully added repository 'jenkins-snapshot'' is displayed. The table lists 8 repositories, including 'jenkins-snapshot'.

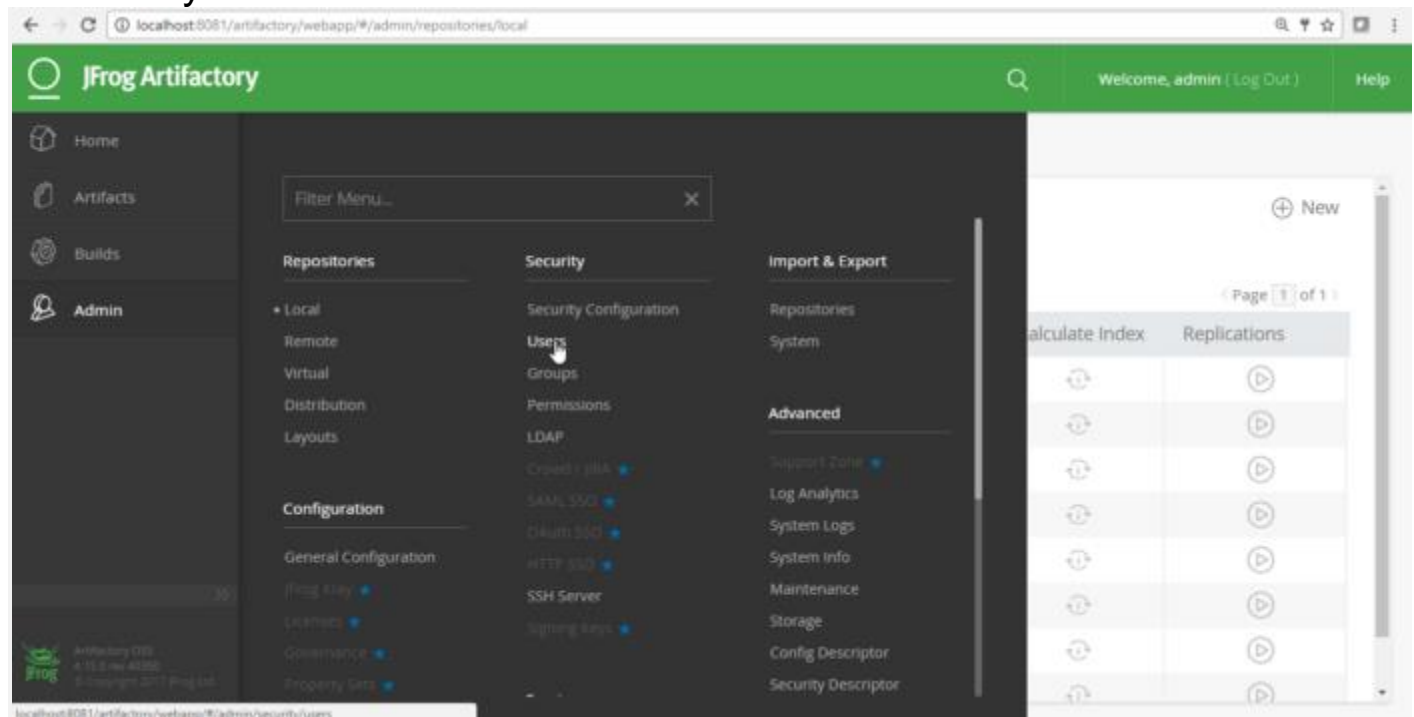
8 Repositories

Filter by Repository Key

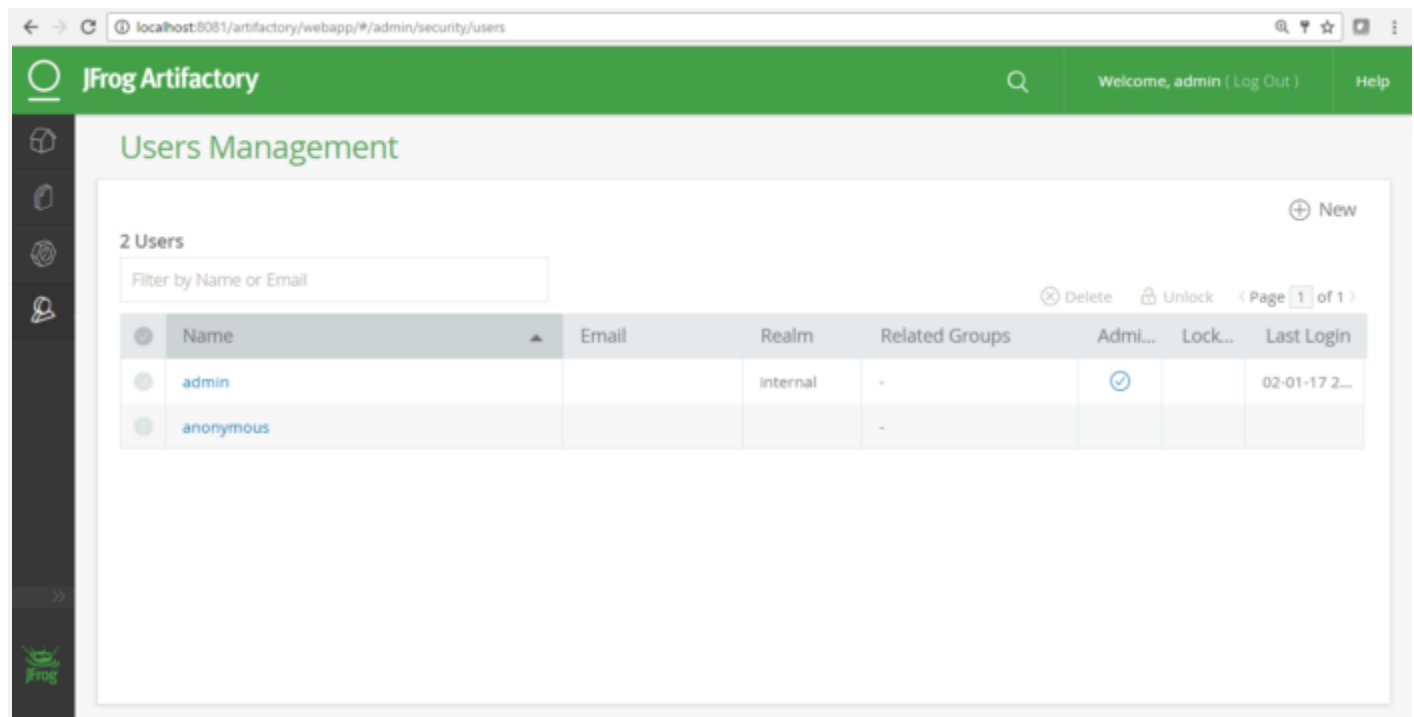
Page 1 of 1

Repository Key	Type	Recalculate Index	Replications
ext-release-local	Maven		
ext-snapshot-local	Maven		
jenkins-release	Maven		
jenkins-snapshot	Maven		
libs-release-local	Maven		
libs-snapshot-local	Maven		
plugins-release-local	Maven		
plugins-snaoshot-local	Maven		

Create a user that you can utilize from Jenkins to access Artifactory:



Click on New:



Provide user details and Save:

The screenshot shows the 'Add New User' form in the JFrog Artifactory web interface. The form is titled 'Add New User' and is located under the 'User Settings' section. It contains the following fields and options:

- User Name ***: A text input field containing 'jenkins'.
- Email Address ***: A text input field containing 'mitesh.soni83@gmail.com'. The email address is partially obscured by a red box.
- Admin**: A checked checkbox.
- Disable UI Access**: An unchecked checkbox.
- Can Update Profile**: A checked checkbox.
- Disable Internal Password**: An unchecked checkbox with a help icon.
- Set Password**: A section with a **Password *** field (containing masked characters) and a **Password Strength** indicator (a green bar).

At the bottom right of the form, there are 'Cancel' and 'Save' buttons.

Verify the list of users:

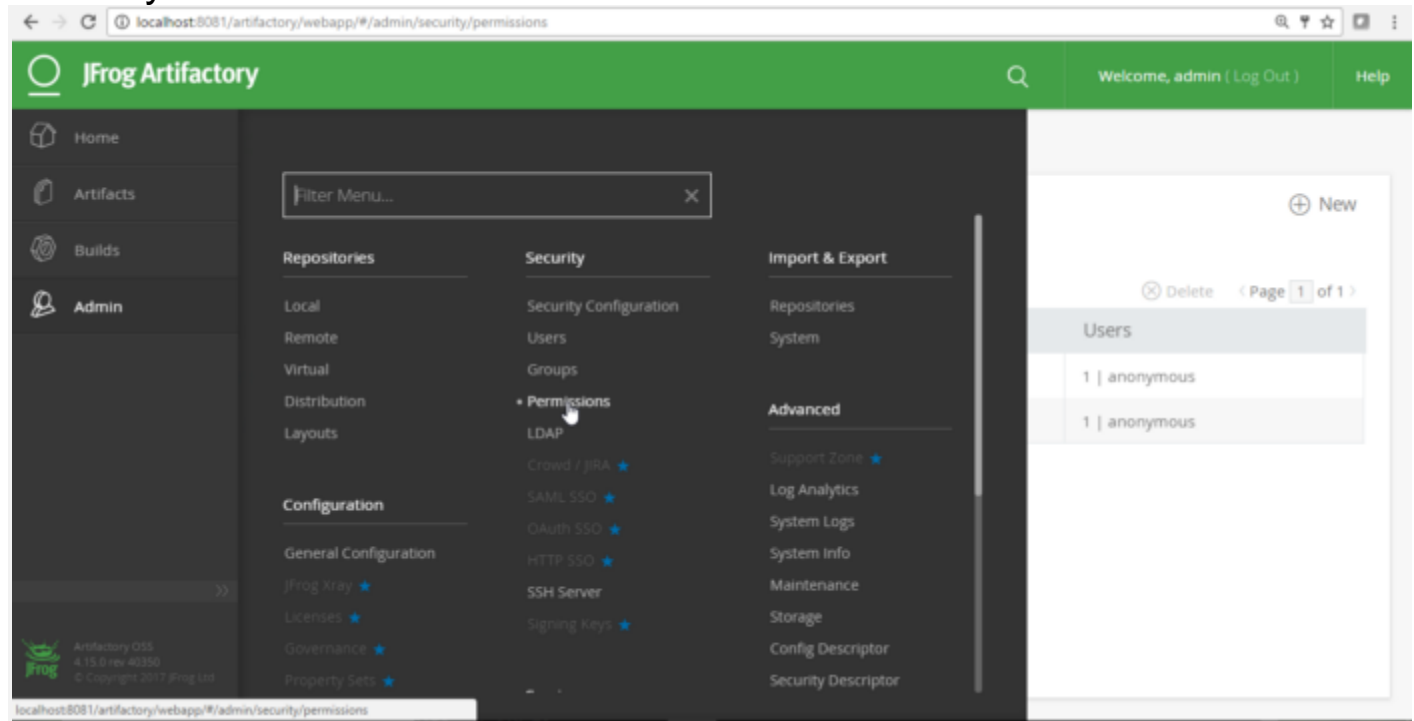
The screenshot shows the 'Users Management' page in the JFrog Artifactory web interface. A success message is displayed at the top right: 'Successfully created user 'jenkins''.

The page displays a list of 3 users. The table has the following columns: Name, Email, Realm, Related Groups, Admin..., Lock..., and Last Login.

Name	Email	Realm	Related Groups	Admin...	Lock...	Last Login
admin		internal	-	✓		02-01-17 2...
anonymous			-			
jenkins	mitesh.soni83@gm...		1 readers	✓		

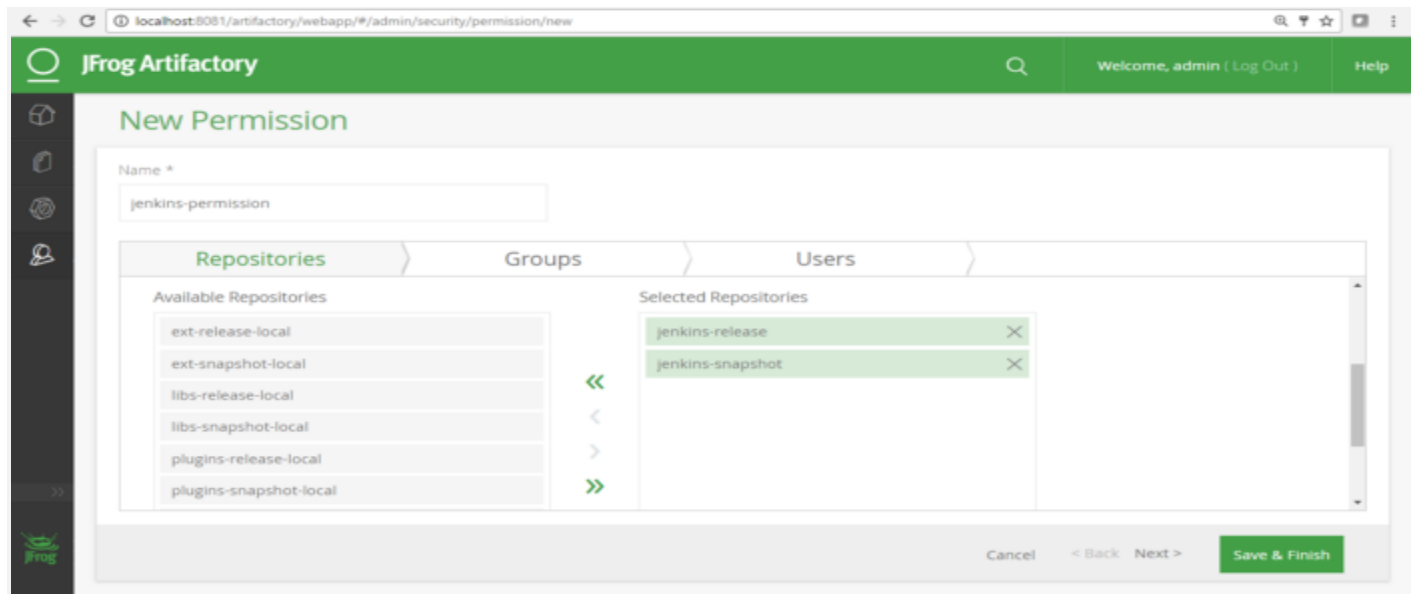
Provide the newly created user with permissions to the repositories:

Security -> Permissions

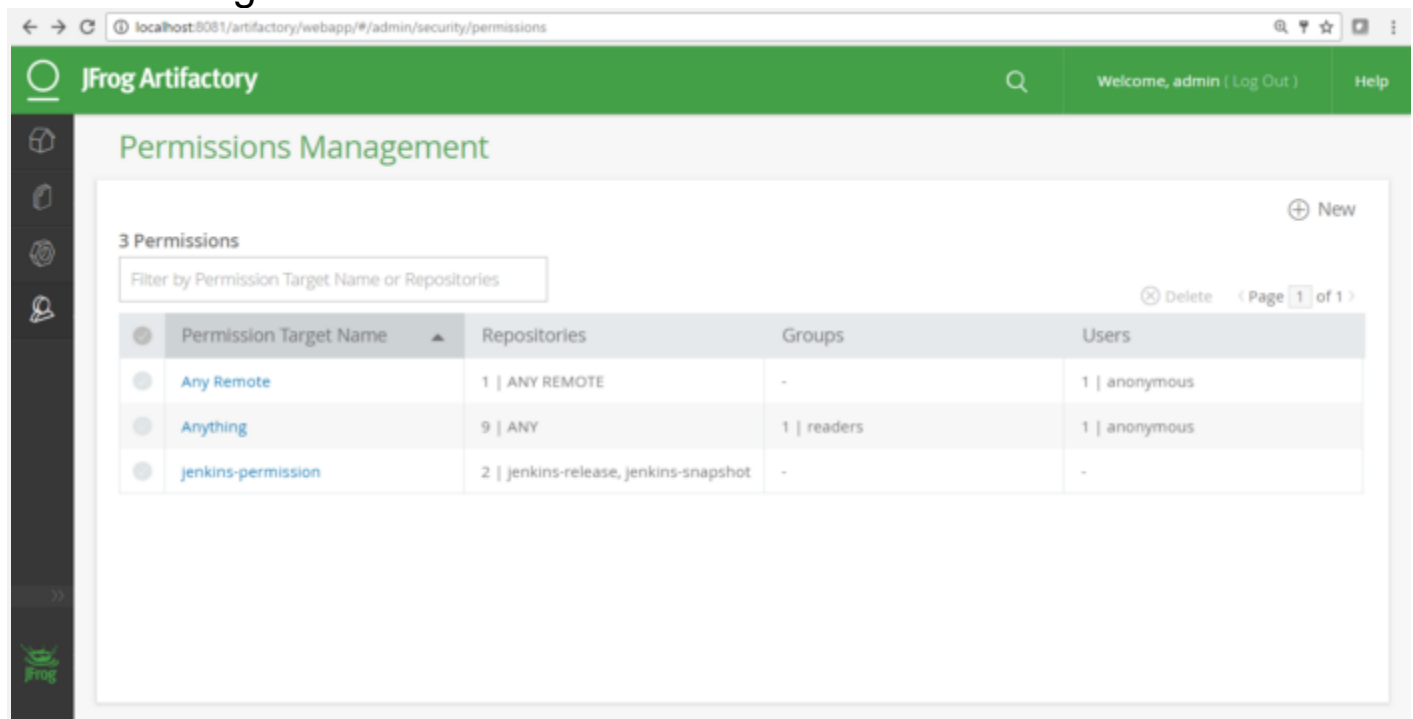


Create a new permission “jenkins-permission”

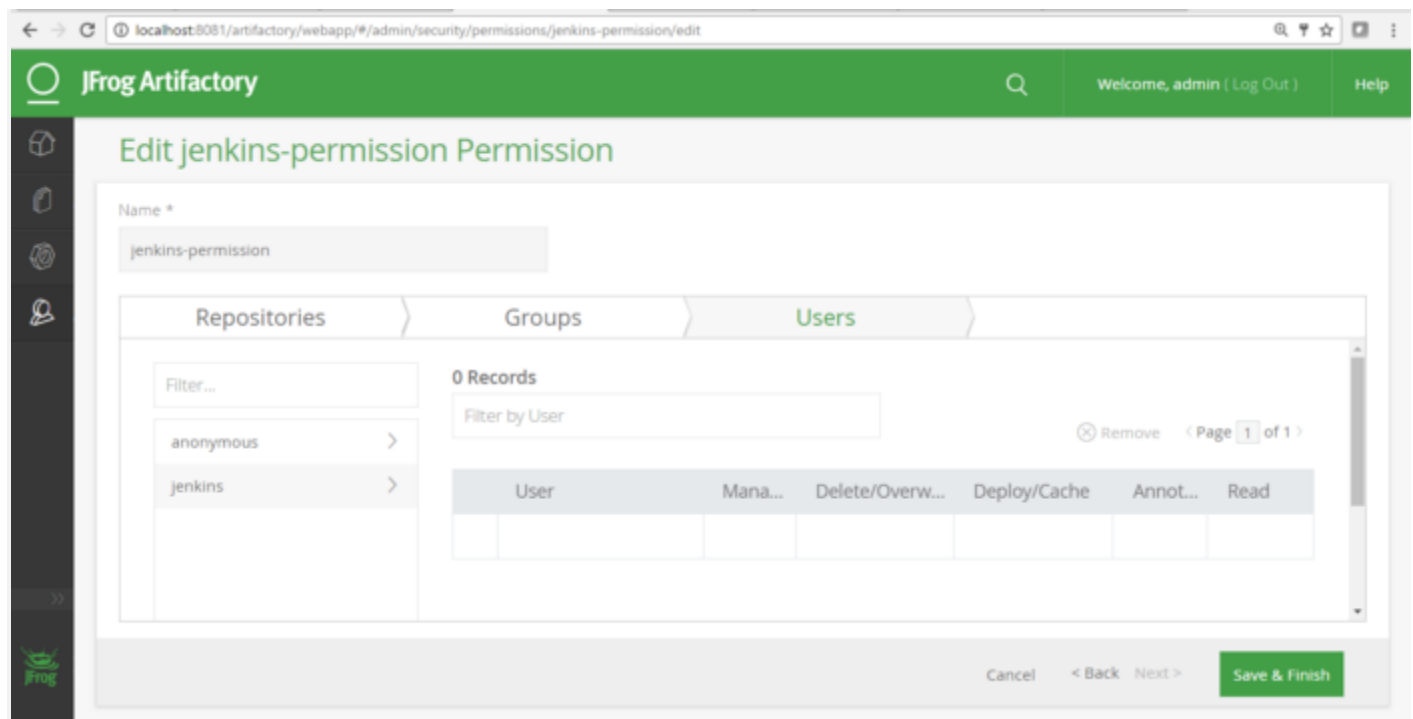
Select Repositories and click on Save & Finish:



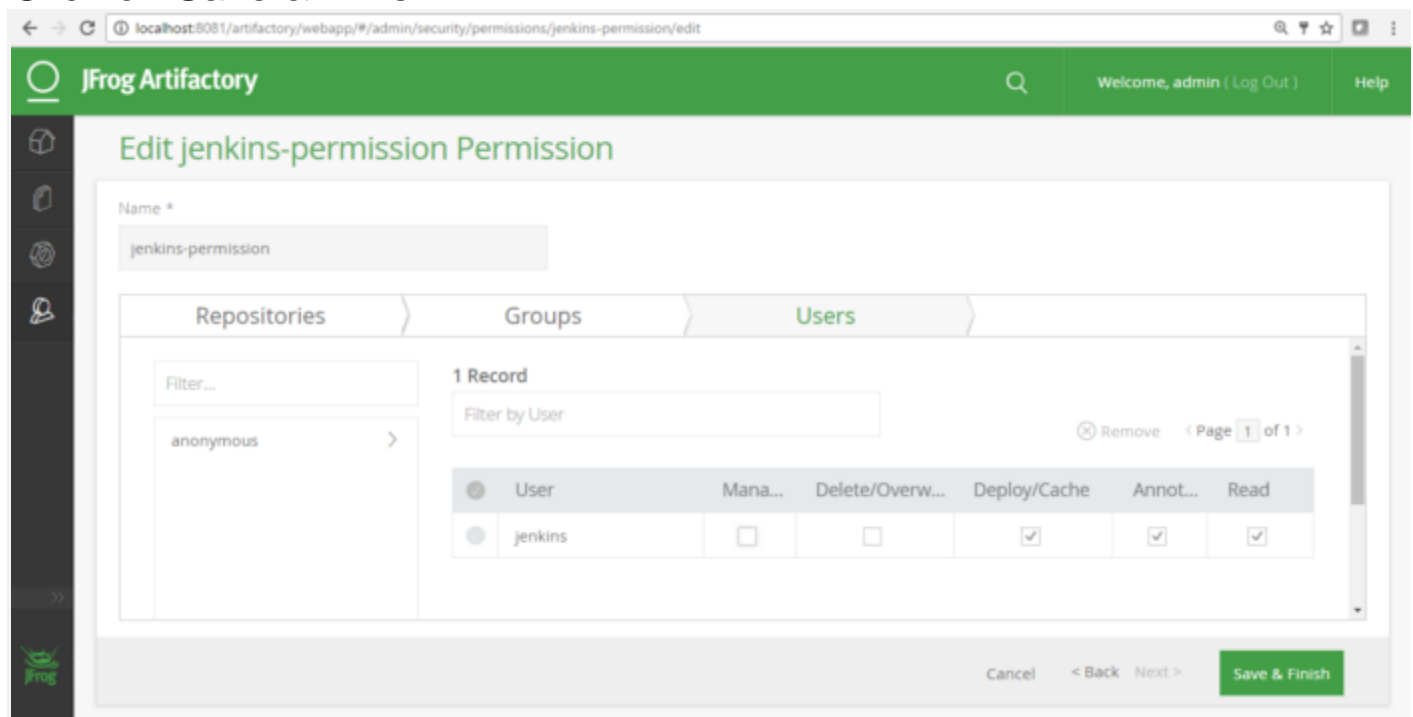
Check the Permissions Management section in Artifactory for recent changes:



Edit the permissions and assign the user: If user is already admin then it has all permissions



Click on Save & Finish:



Check the Permissions Management section in Artifactory for recent changes:

The screenshot displays the Jfrog Artifactory web interface for Permissions Management. The header shows the Jfrog logo and the user 'admin' is logged in. A notification banner at the top right indicates a successful update for the 'jenkins-permission' target. Below the header, there is a search bar and a table listing permissions. The table has columns for Permission Target Name, Repositories, Groups, and Users. The 'jenkins-permission' target is highlighted in blue.

Permission Target Name	Repositories	Groups	Users
Any Remote	1 ANY REMOTE	-	1 anonymous
Anything	9 ANY	1 readers	1 anonymous
jenkins-permission	2 jenkins-release, jenkins-snapshot	-	1 jenkins

Jfrog setup is complete

Now storing the artifacts to the jfrog artifactory

Integrate Artifactory with Jenkins

Go to Manage Jenkins -> Manage Plugins

The screenshot shows the Jenkins Manage Plugins page. The 'Available' tab is selected, and the 'Artifactory' plugin is listed. The plugin description is 'Integrates Artifactory to Jenkins' and the version is '2.15.0'. The 'Install' button is visible, and there are options to 'Install without restart' or 'Download now and install after restart'.

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

URL

Default Deployer Credentials

Username

Password

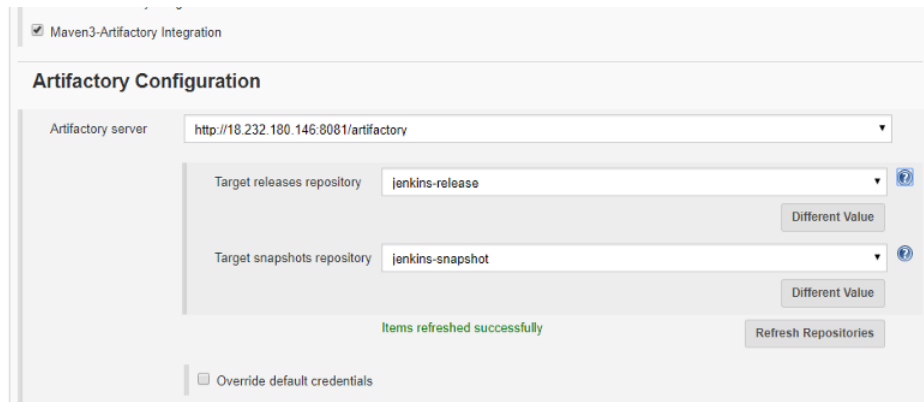
 Advanced...

Test Connection

☐ Use Different Resolver Credentials

Delete

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:



Add Build Step as shown below



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 msec]
[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 - -----
```

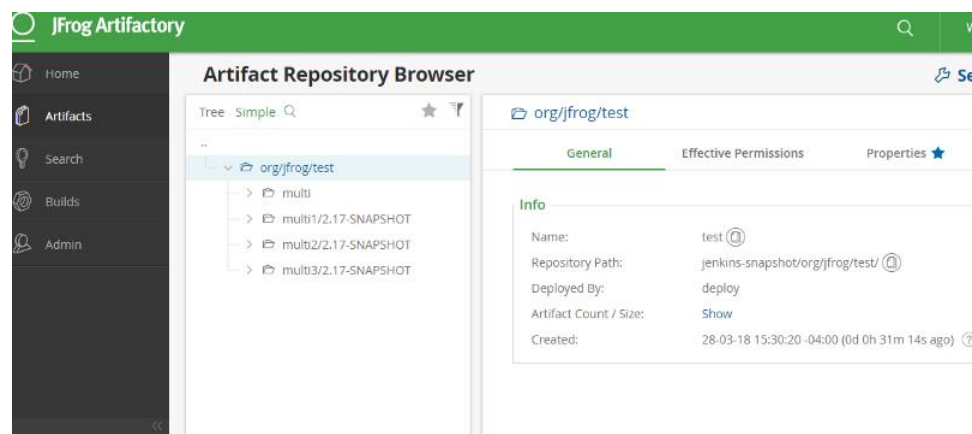
```
[main] INFO org.apache.maven.cli.event.ExecutionEventLogger - Total time: 5.419 s
```

```
[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:



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:

Updates Available Installed Advanced				
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	jQuery plugin This allows other plugins to use jQuery in UI.	1.11.2-0		Uninstall
<input checked="" type="checkbox"/>	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
<input checked="" type="checkbox"/>	Pipeline: Groovy Pipeline execution engine based on continuation passing style transformation of Groovy scripts.	2.34	Downgrade to 2.30	Uninstall
<input checked="" type="checkbox"/>	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
<input checked="" type="checkbox"/>	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:

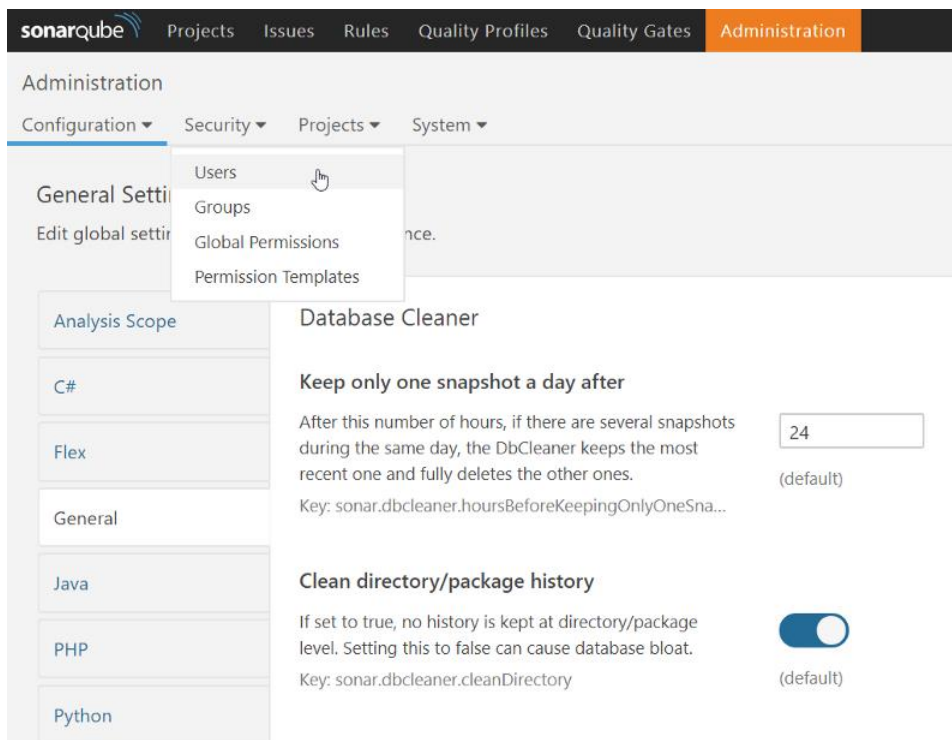
The screenshot shows the SonarQube dashboard. At the top, there's a navigation bar with 'sonarcube' logo and links for Projects, Issues, Rules, Quality Profiles, and Quality Gates. A 'Log in' button is on the right. Below the navigation bar, the main header says 'Continuous Code Quality' with a 'Log in' button and a 'Read documentation' link. To the right of the header, there's a summary of metrics: '2 Projects Analyzed', '66 Bugs', '0 Vulnerabilities', and '70 Code Smells'. Below this, a section titled 'Multi-Language' states that 20+ programming languages are supported, including Java, C/C++, C#, COBOL, ABAP, HTML, RPG, JavaScript, Objective C, XML, VB.NET, PL/SQL, Flex, Python, Groovy, PHP, Swift, Visual Basic, and PL/I. At the bottom, a 'Quality Model' section explains the metrics: 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; and 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:

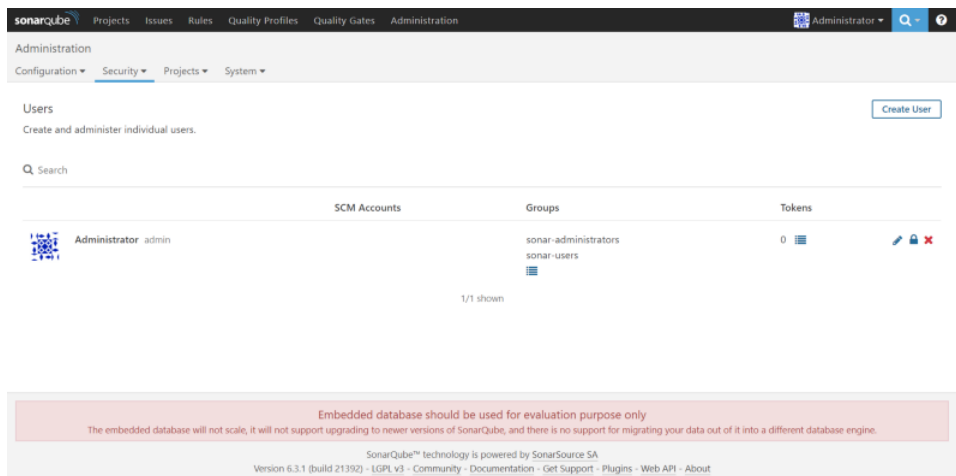
Log In to SonarQube

[Cancel](#)

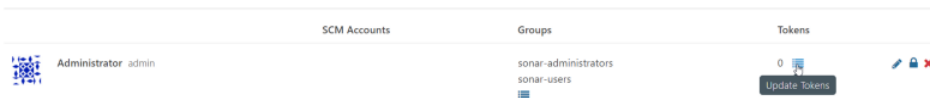
1. As of now, there is no project available in the SonarQube dashboard.
2. Click on the Administration tab and in the Security menu click on Users:



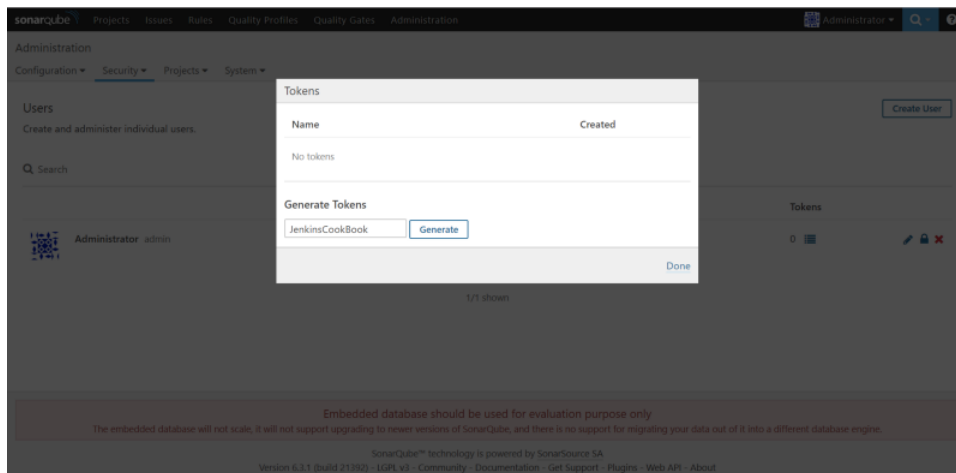
1. Initially, there are no tokens issued; there is a 0 token for Administrator:



1. Click on Tokens:



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



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

Done

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	<input type="text" value="Sonarqube6.3"/>
Server URL	<input type="text" value="http://localhost:9000/"/> <small>Default is http://localhost:9000</small>
Server version	<input type="text" value="5.3 or higher"/>
Server authentication token	<input type="text" value="....."/> <small>Configuration fields depend on the SonarQube server version.</small>
SonarQube account login	<input type="text"/> <small>SonarQube authentication token. Mandatory when anonymous access is disabled.</small>
SonarQube account password	<input type="text"/> <small>SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.</small>

Advanced...

Delete SonarQube

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

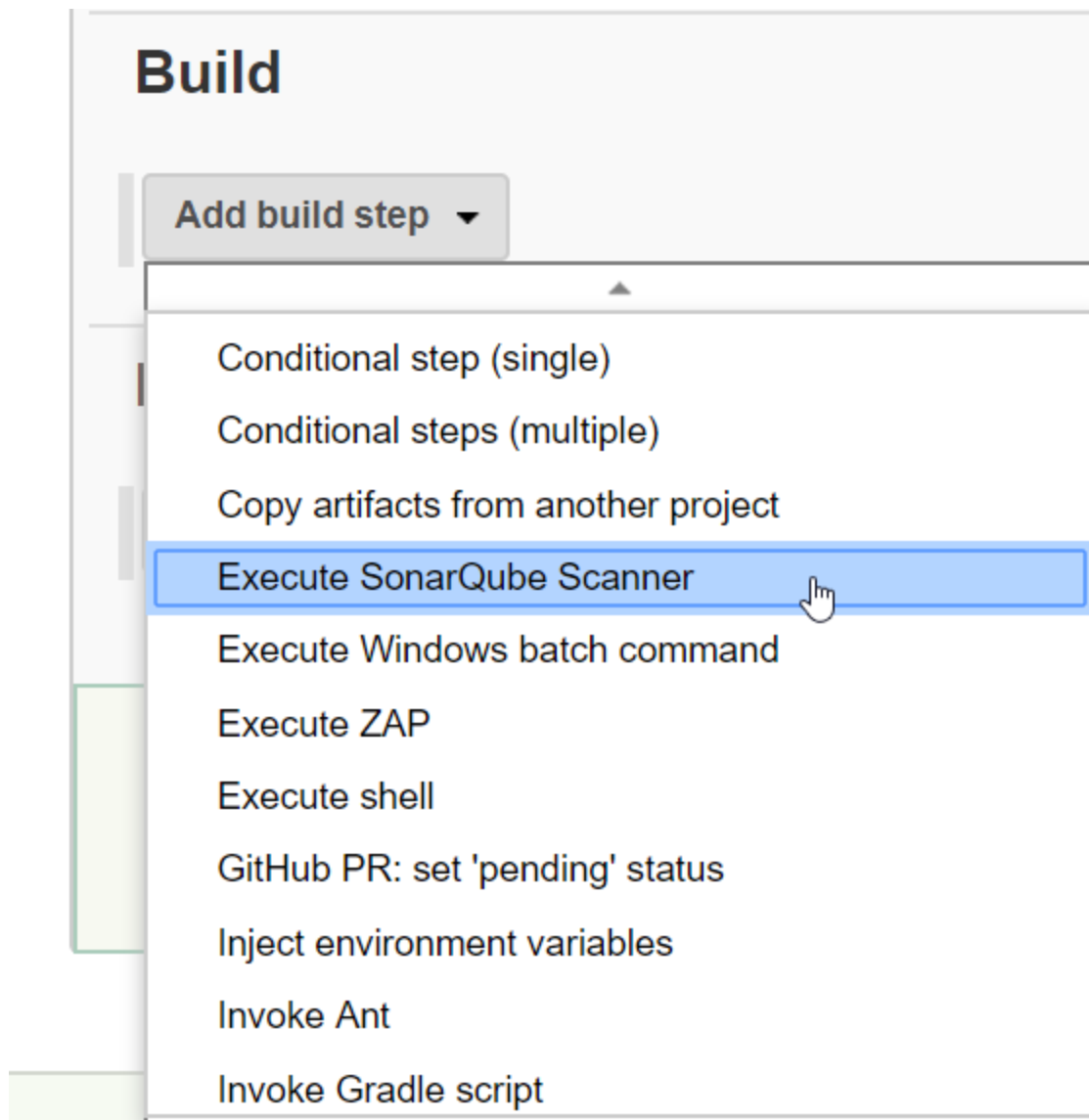
SonarQube Scanner

SonarQube Scanner installations

<div>SonarQube Scanner</div> <div>Name <input type="text" value="SonarQube Scanner 3.0.3"/></div> <div><input checked="" type="checkbox"/> Install automatically</div> <div><div>Install from Maven Central</div><div>Version <input type="text" value="SonarQube Scanner 3.0.3.778"/></div></div>	<div>Delete Installer</div> <div>Add Installer</div> <div>Add SonarQube Scanner</div>
--	---

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:



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:

The screenshot shows the Jenkins 'Build' configuration page for the 'Execute SonarQube Scanner' step. The 'Task to run' field is empty. The 'JDK' dropdown is set to '(Inherit From Job)'. The 'Path to project properties' field is empty. The 'Analysis properties' field contains the following text:
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)
The 'Additional arguments' and 'JVM Options' fields are empty. At the bottom, there are 'Save' and 'Apply' buttons.

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

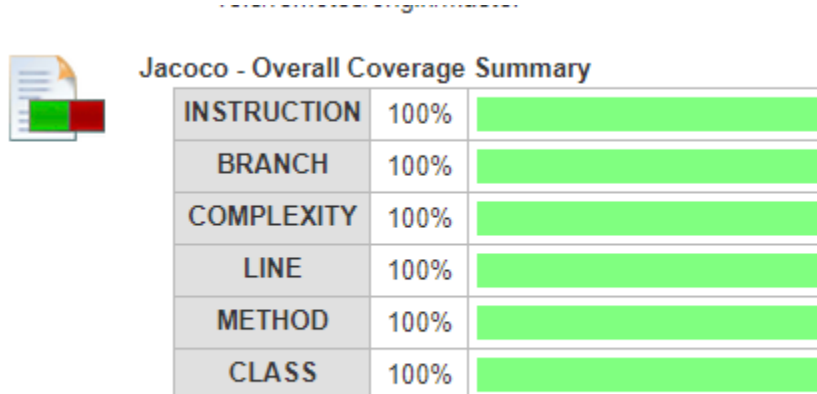
1. Install Jacoco plugin
2. Manage Jenkins -> Manage Plugin -> Search for Jacoco
3. Create a freestyle project in Jenkins
4. Use this link in source control amangement
“<https://github.com/pkainulainen/maven-examples.git>”
5. You have to add all configuration in pom.xml as present in the code

The screenshot shows the Jenkins 'Build' configuration page for the 'Invoke Artifactory Maven 3' step. The 'Maven Version' dropdown is set to 'MAVEN_HOME'. The 'Root POM' field contains 'code-coverage-jacoco'. The 'Goals and options' field contains 'test'. An 'Advanced...' button is visible at the bottom right.

6. Add post build actions

- 7 Select “Record Jacoco Coverage report”

8. Jenkins build logs as shown below



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/dhutifile/calculator.git'
      }
    }
    stage("Compile") {
      steps {
        sh "./gradlew compileJava"
      }
    }
    stage("Unit test") {
      steps {
```

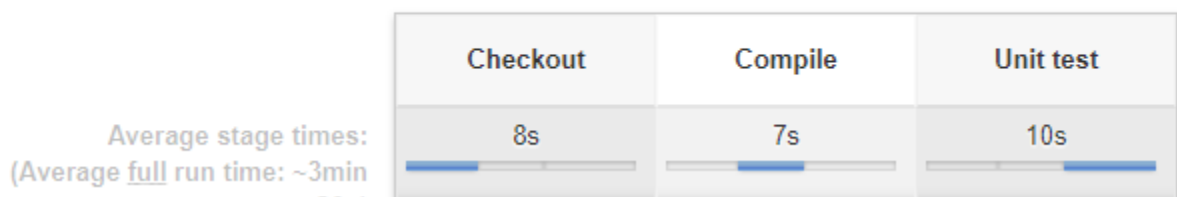


```
sh "./gradlew test"
}
}
}
}
```

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

```
./gradlew bootRun
```

Stage View



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/dhutifile/calculator.git> in Repository URL

The screenshot shows the Jenkins Pipeline Configuration page. At the top, it says "Pipeline script from SCM". Below this, the "SCM" section is expanded, showing "Git" as the selected provider. Under "Repositories", the "Repository URL" is set to "https://github.com/leszko/calculator.git", and "Credentials" is set to "- none -". There are buttons for "Advanced...", "Add Repository", and "Add". Under "Branches to build", the "Branch Specifier (blank for 'any')" is set to "*/master", and there is a button for "Add Branch". Below this, the "Repository browser" is set to "(Auto)". At the bottom, there is a section for "Additional Behaviours" with an "Add" button. Finally, the "Script Path" is set to "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:

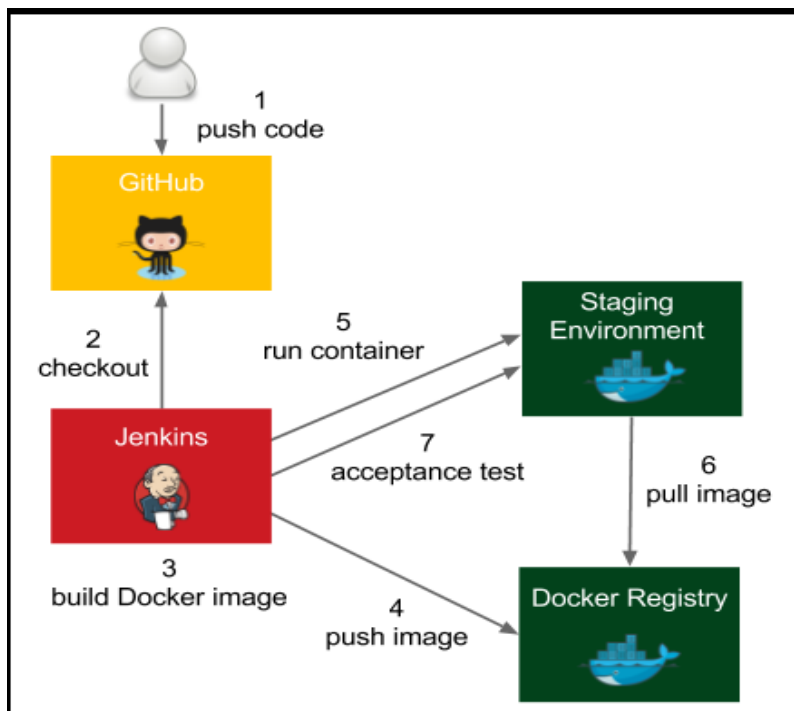
```
apply plugin: "jacoco"
```

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 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"
      }
    }
  }

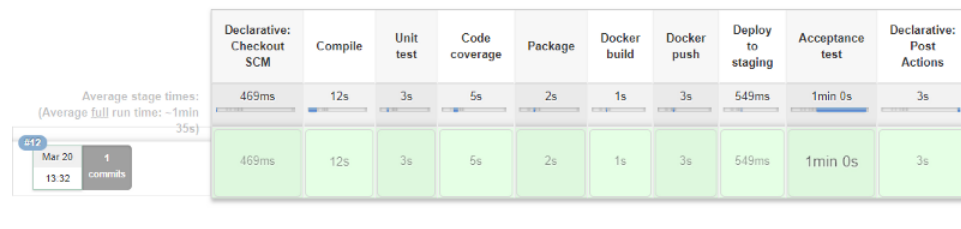
  stage("Package") {
    steps {
```

```
    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



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