Jenkins

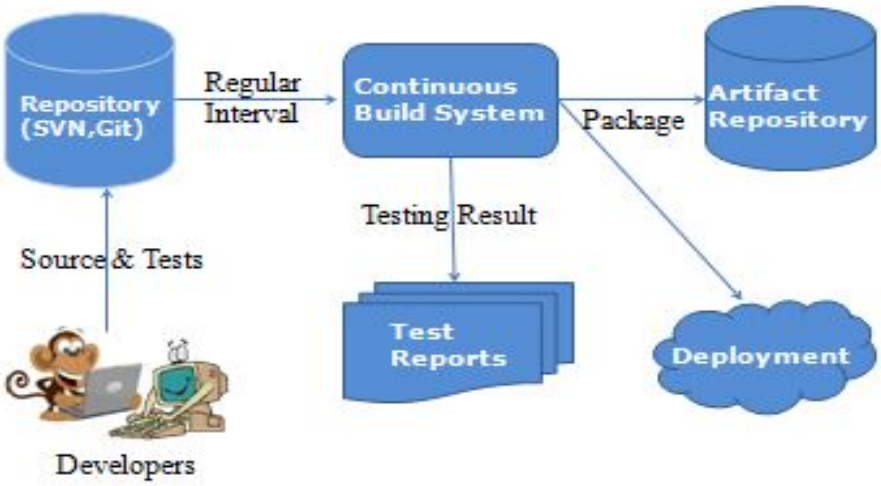
Jenkins Introduction

Jenkins, is an open source Continuous Integration, cross-platform tool written in Java. Kohsuke Kawaguchi is Creator of the Jenkins CI server in 2004. Initially, it was called Hudson, but in 2011 it was renamed to Jenkins because of disputes with Oracle.
The tool simplifies the process of integration of changes in to the project and delivery of fresh builds to users.

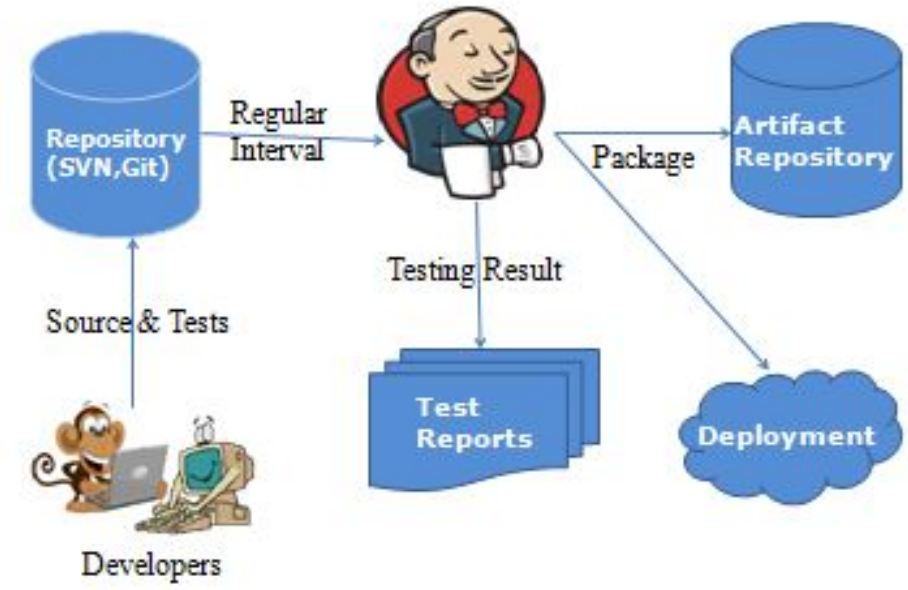
Continuous Integration: Continuous Integration (CI) is the process of automating the build and testing of code every time a team member commits changes to version control.
(OR)
Continuous Integration is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. Each integration can then be verified by an automated build and automated tests. While automated testing is not strictly part of CI it is typically implied.
One of the key benefits of integrating regularly is that you can detect errors quickly and locate them more easily. As each change introduced is typically small, pinpointing the specific change that introduced a defect can be done quickly.

CI – What does it really mean?
At a regular frequency (ideally at every commit), the system is:
Integrated All changes up until that point are combined into the project
Built The code is compiled into an executable or package
Tested Automated test suites are run
Archived Versioned and stored so it can be distributed as is, if desired
Deployed Loaded onto a system where the developers can interact with it

CI Flow



Below diagram CI flow with Jenkins as Build tool.



CI – Benefits

- Immediate bug detection
- No integration step in the lifecycle
- A deployable system at any given point
- Record of evolution of the project

CI – The tools

Code Repositories
GitHub, SVN, CVS, TFS and Mercurial,

Continuous Build Systems
Jenkins, Bamboo, Cruise Control, Team City, Circle CI and Travis CI

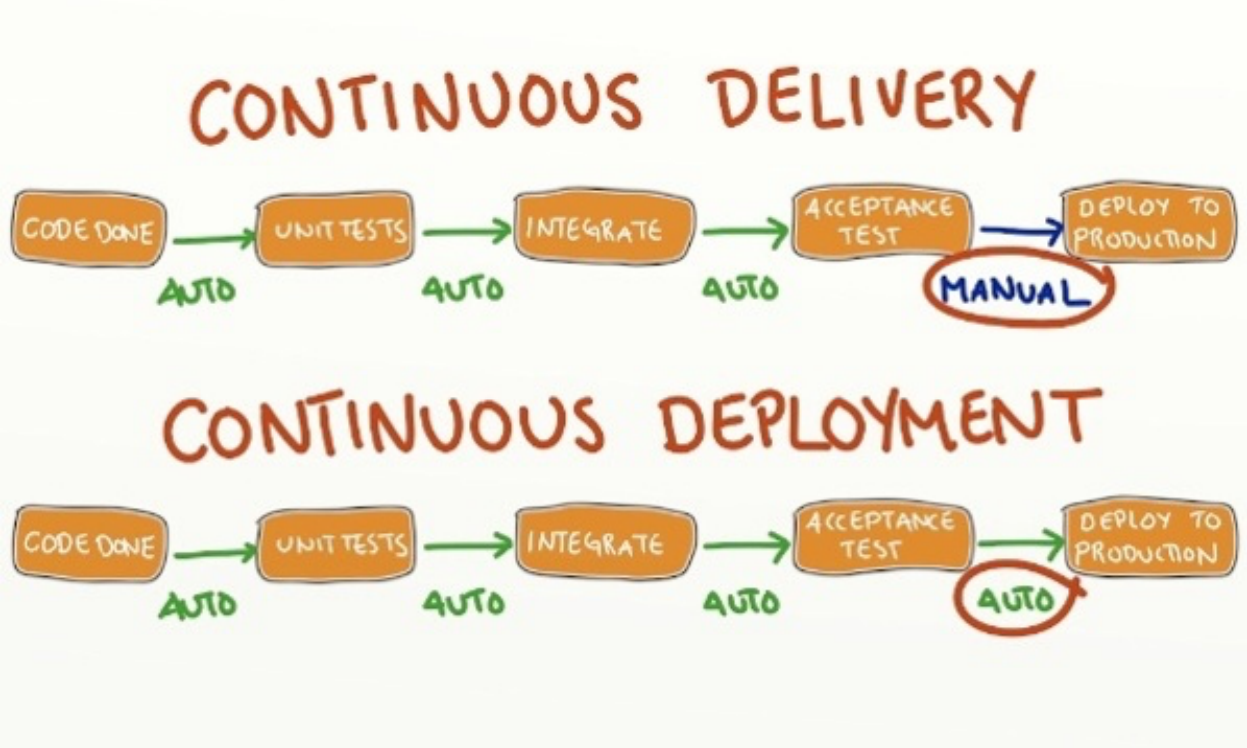
Test Frameworks
JUnit, Cucumber, CppUnit

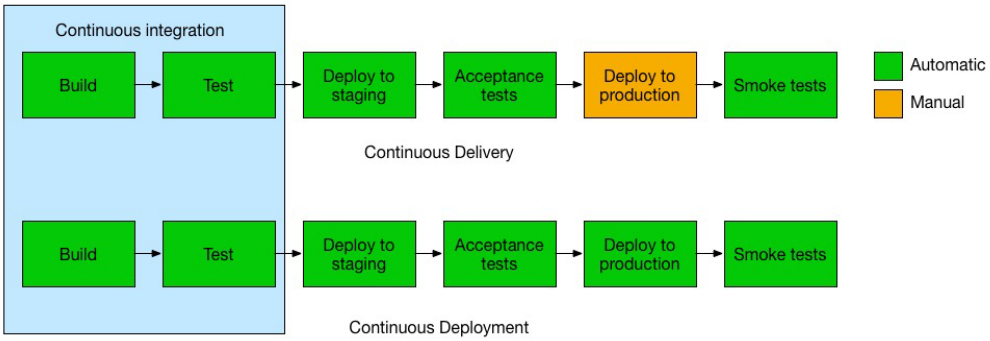
Artifact Repositories
Nexus, Artifactory, Archiva

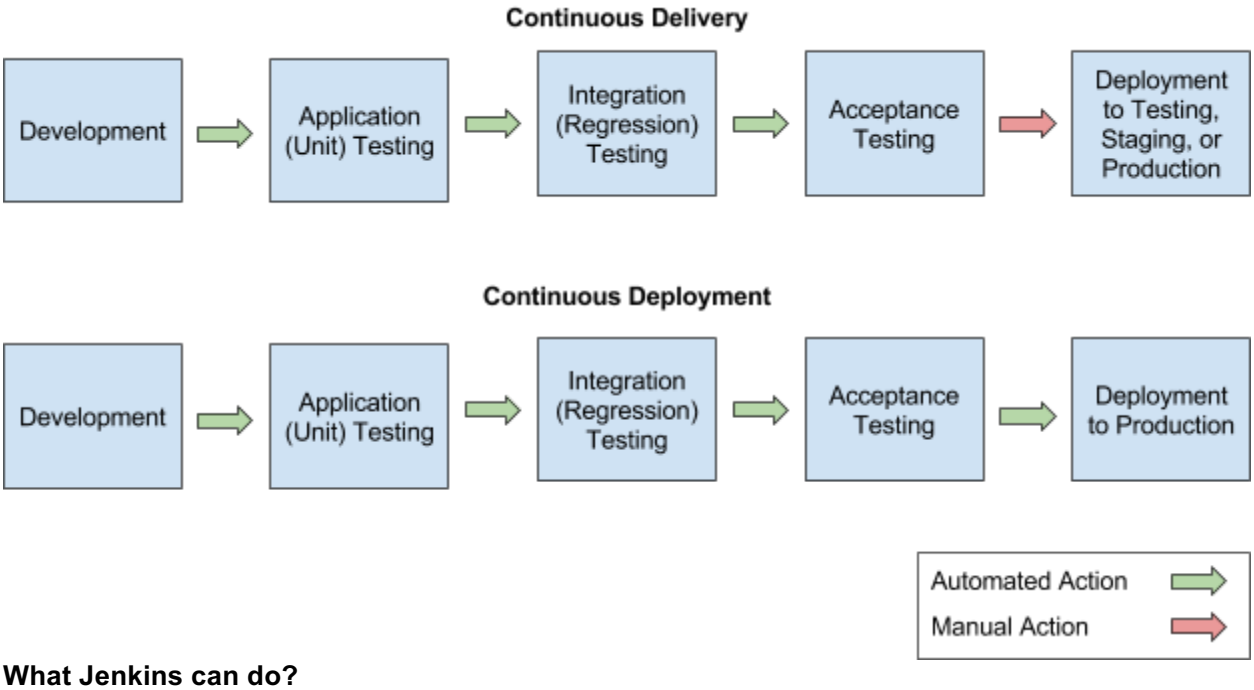
Build Tools
Ant, Maven and Gradle

Continuous Delivery: Any and every successful build that has passed all the relevant automated tests and quality gates can potentially be deployed in to production via fully automated one click process.

Continuous Deployment: The practicing of automatically deploying every successful build directly into production without any manual steps knows as Continuous deployment.
(OR)
It is closely related to Continuous Integration and refers to keeping your application deployable at any point or even automatically releasing to a test or production environment if the latest version passes all automated tests.







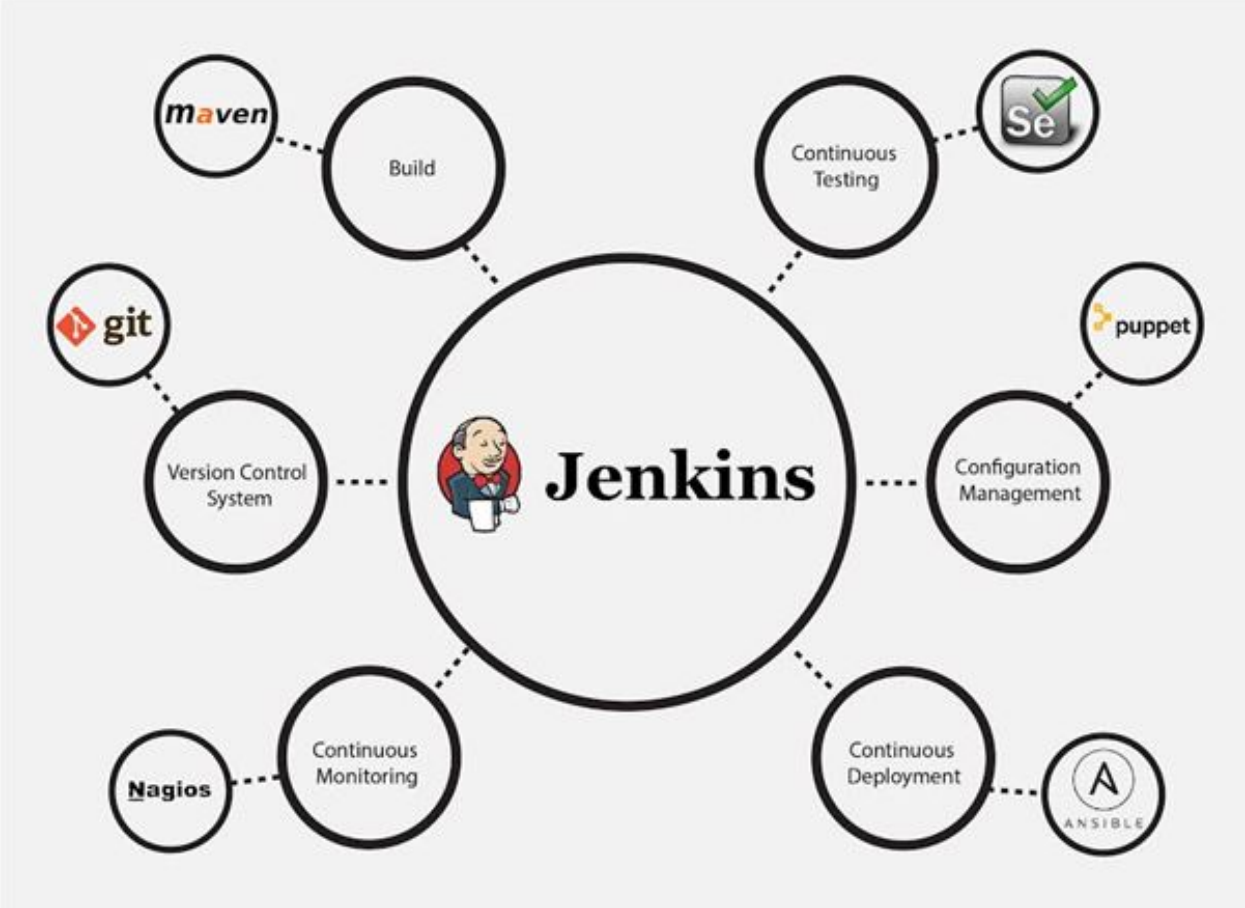
What Jenkins can do?

- Integrate with many different Version Control Systems (GitHub, CVS, SVN, TFS …)
- Generate test reports（JUnit)
- Push the builds to various artifact repositories
- Deploys directly to production or test environments
- Notify stakeholders of build status (Through Email)

Benefits of Jenkins

- ✓ Its an open source tool with great community support.
- ✓ Easy to install and It has a simple configuration through a web-based GUI, which speeds up the Job
- ✓ It has around 900+ plugins to ease your work. If a plugin does not exist, just code it up and share with the community.
- ✓ Its built with Java and hence, it is portable on all major platforms.
- ✓ Good documentation and enriched support articles/information available on internet which will help beginners to start easy
- ✓ Specifically, for a test only project, it is used to schedule jobs for regression testing without manual intervention and hence monitor infrastructural and functional health of a application. It can be used like a scheduler for integration testing and also can be used to validate new deployments/environments on a single click on a Build now button.

The diagram below depicts that Jenkins is integrating various DevOps stages:



List of popular continuous Integration tools

SNo	Product	Is Open Source?
1	Jenkins	Yes
2	Bamboo	No
3	Cruise Control	Yes
4	Travis CI	Yes and Paid also
5	Circle CI	Yes and Paid also
6	GitLab CI	Yes and Paid
7	TeamCity	Yes and Paid

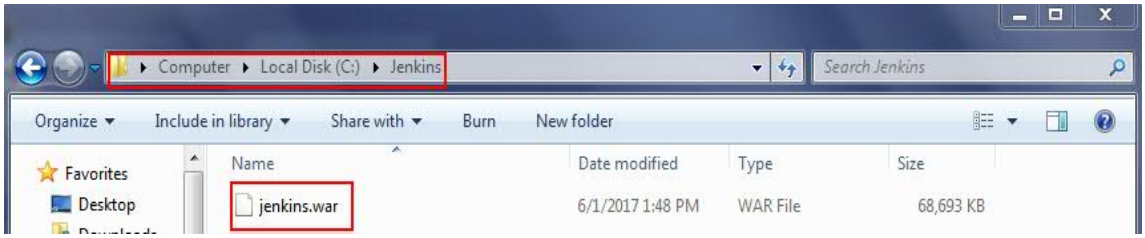
Jenkins Installation

- Jenkins is java based CI tool, so we need to install jdk/jre before installing.
- **Pre Requisite Software:** Java (Check weather java is installed or not with java –version command)
- We can Install Jenkins in 3 ways.

Method 1:

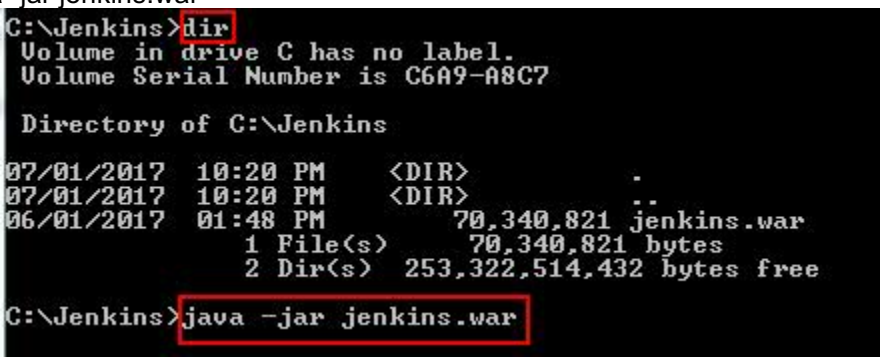
Step 1: Download jenkins.war file from https://jenkins.io/download/

Step 2: Now copy 'jenkins.war' in a folder. I have created one folder like Jenkins in C drive and placed there.



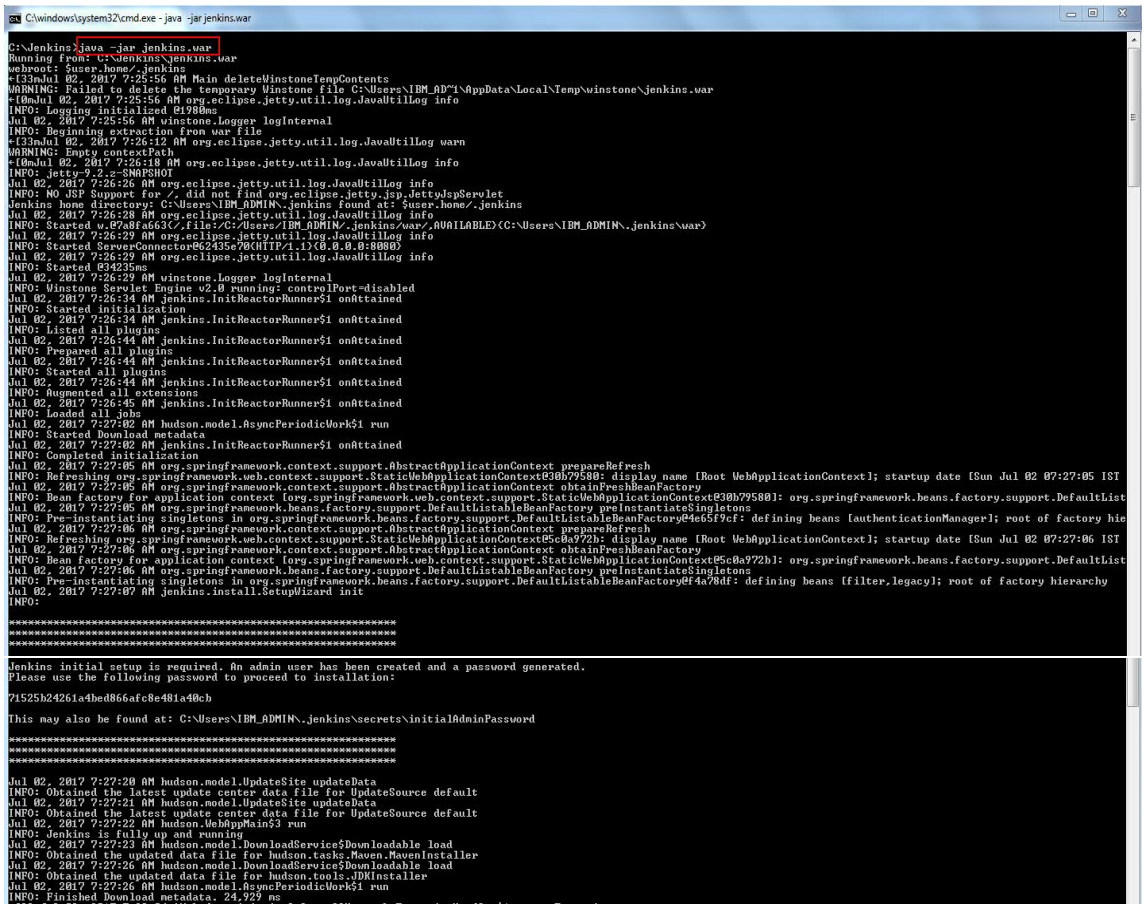
Step 3: Now go to the command prompt and execute the below statement via command line.

java -jar jenkins.war



Note: java -jar jenkins.war ---> With Default Port No 8080
java -jar jenkins.war --httpPort=8081 ---> With customised Port No
java -jar jenkins.war --help ---> Displays the all possible options.
nohup java -jar jenkins.war > $LOGFILE 2>&1 ---> Even if terminal close also job will run in background.

Step 4: After executing the above command, you should see something like below.

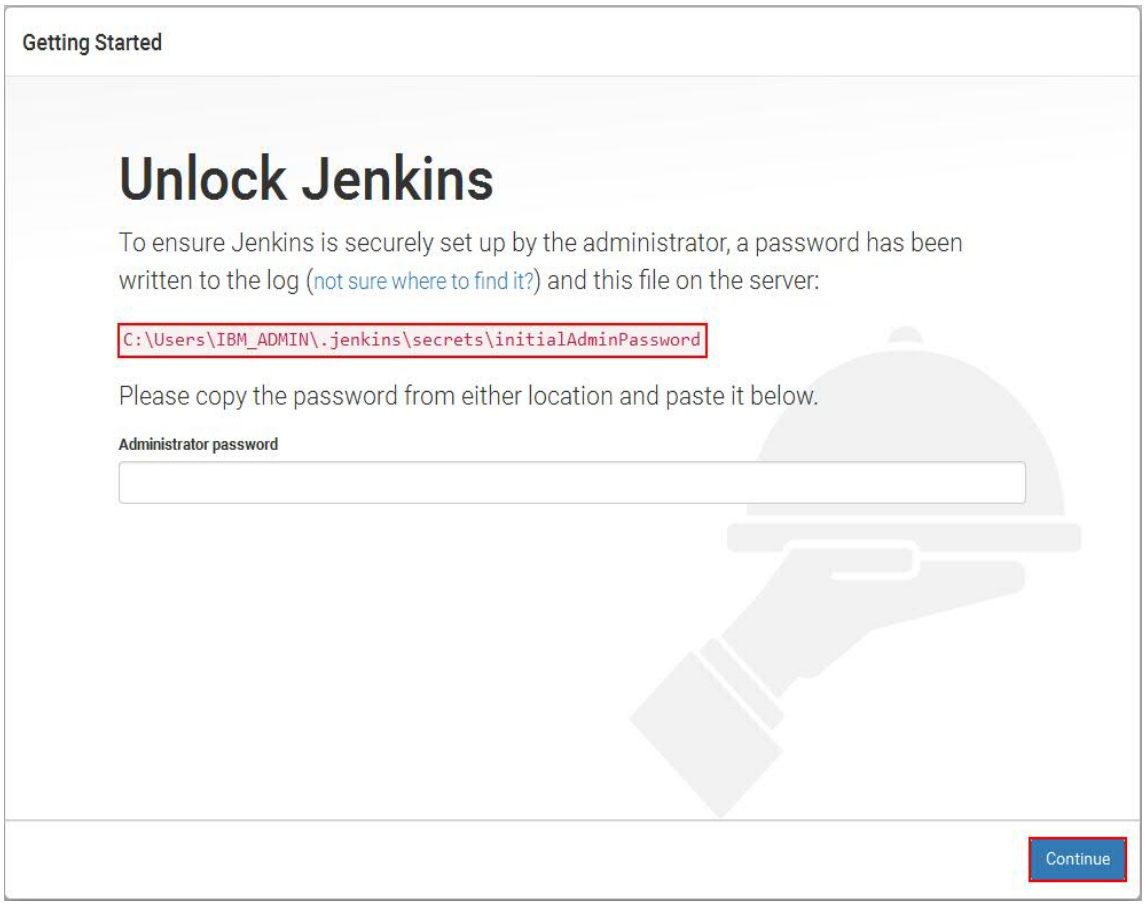


Step 5: Once Jenkins has started, you should now be able to access it by using URL - http://localhost:8080

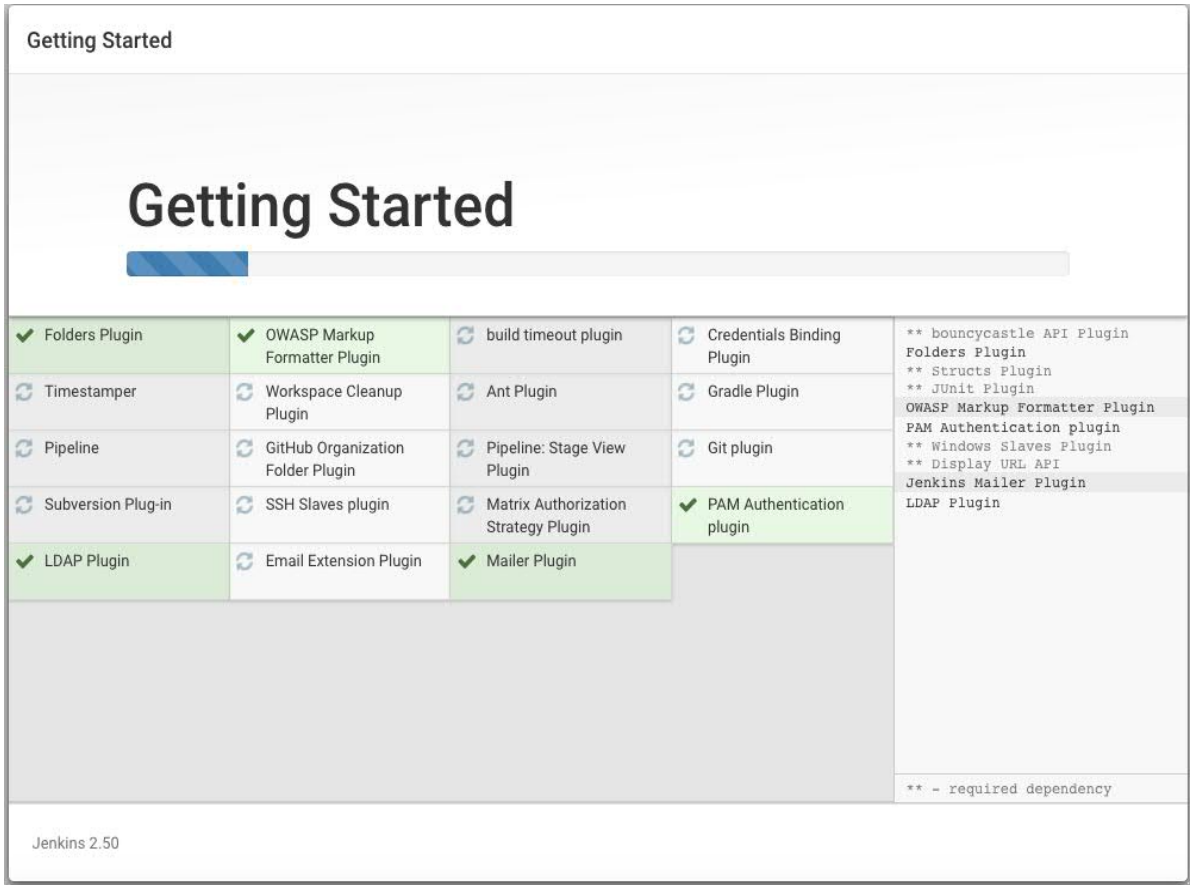
Note: Jenkins will run on 8080 port number by default.

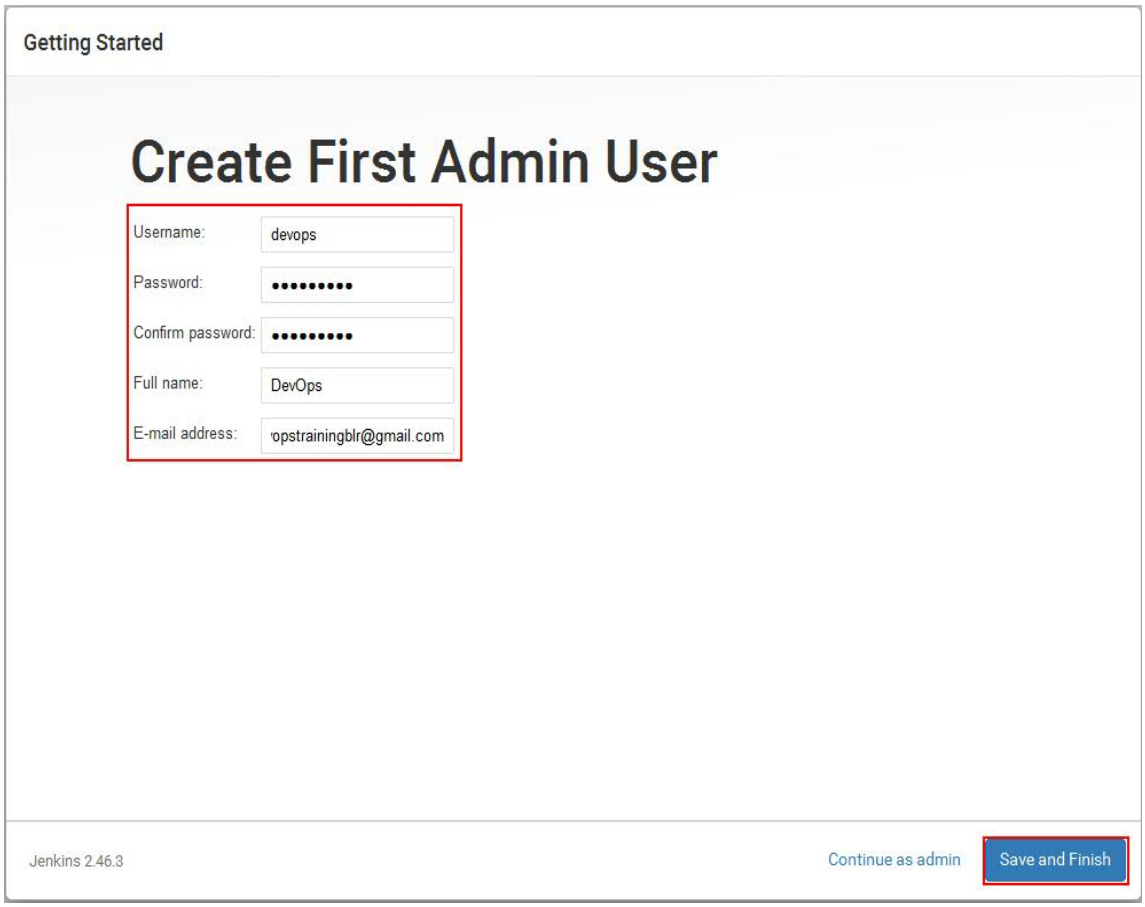
Jenkins 1.534 and earlier ships with Winstone, so no additional installation needed. Just run with java -jar jenkins.war.

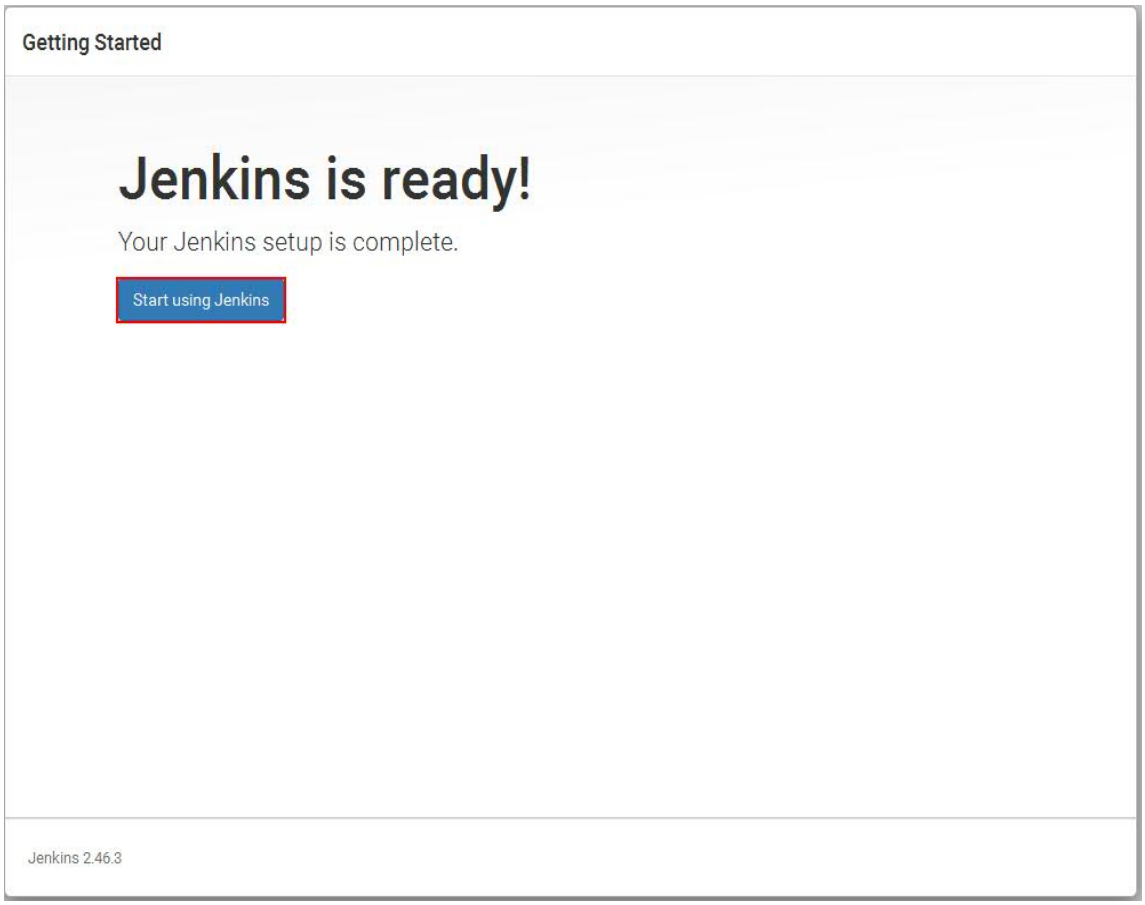
Jenkins comes bundled as a WAR file that you can run directly using an embedded servlet container. Jenkins uses the lightweight Winstone servlet engine to allow you to run the server out of the box, without having to configure a web server yourself.

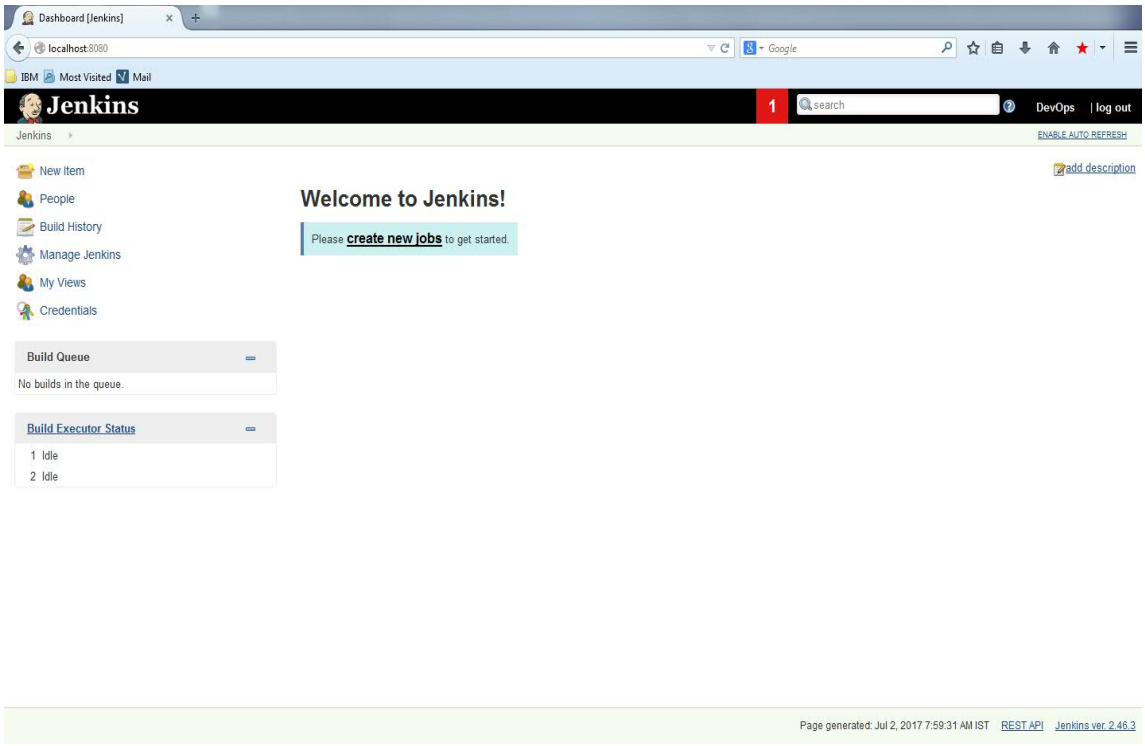

Getting Started
Unlock Jenkins
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:
C:\Users\IBM_ADMIN\.jenkins\secrets\initialAdminPassword
Please copy the password from either location and paste it below.
Administrator password
Continue


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.
Jenkins 2.46.3


Getting Started
Getting Started
Folders Plugin
OWASP Markup Formatter Plugin
build timeout plugin
Credentials Binding Plugin
Timestamper
Workspace Cleanup Plugin
Ant Plugin
Gradle Plugin
Pipeline
GitHub Organization Folder Plugin
Pipeline: Stage View Plugin
Git plugin
Subversion Plug-in
SSH Slaves plugin
Matrix Authorization Strategy Plugin
PAM Authentication plugin
LDAP Plugin
Email Extension Plugin
Mailer Plugin
** bouncycastle API Plugin
Folders Plugin
** Structs Plugin
** JUnit Plugin
OWASP Markup Formatter Plugin
PAM Authentication plugin
** Windows Slaves Plugin
** Display URL API
Jenkins Mailer Plugin
LDAP Plugin
** - required dependency
Jenkins 2.50


Getting Started
Create First Admin User
Username:
devops
Password:
Confirm password:
Full name:
DevOps
E-mail address:
opstrainingblr@gmail.com
Jenkins 2.46.3
Continue as admin
Save and Finish


Getting Started
Jenkins is ready!
Your Jenkins setup is complete.
Start using Jenkins
Jenkins 2.46.3


Dashboard [Jenkins]
localhost:8080
IBM Most Visited Mail
Jenkins
1
search
DevOps | log out
Jenkins
ENABLE AUTO REFRESH
New Item
People
Build History
Manage Jenkins
My Views
Credentials
add description
Welcome to Jenkins!
Please create new jobs to get started.
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Page generated: Jul 2, 2017 7:59:31 AM IST REST API Jenkins ver. 2.46.3

Method 2:

You can simply deploy the Jenkins WAR in your application server - with Tomcat, for example, you can simply place the jenkins.war file in the webapps Tomcat directory.
Start the tomcat server as follows.
>cd TOMCAT_HOME/bin directory

Windows
>startup.bat (OR) > catalina.bat start
MAC/Linux:
#startup.sh (OR) > catalina.sh start

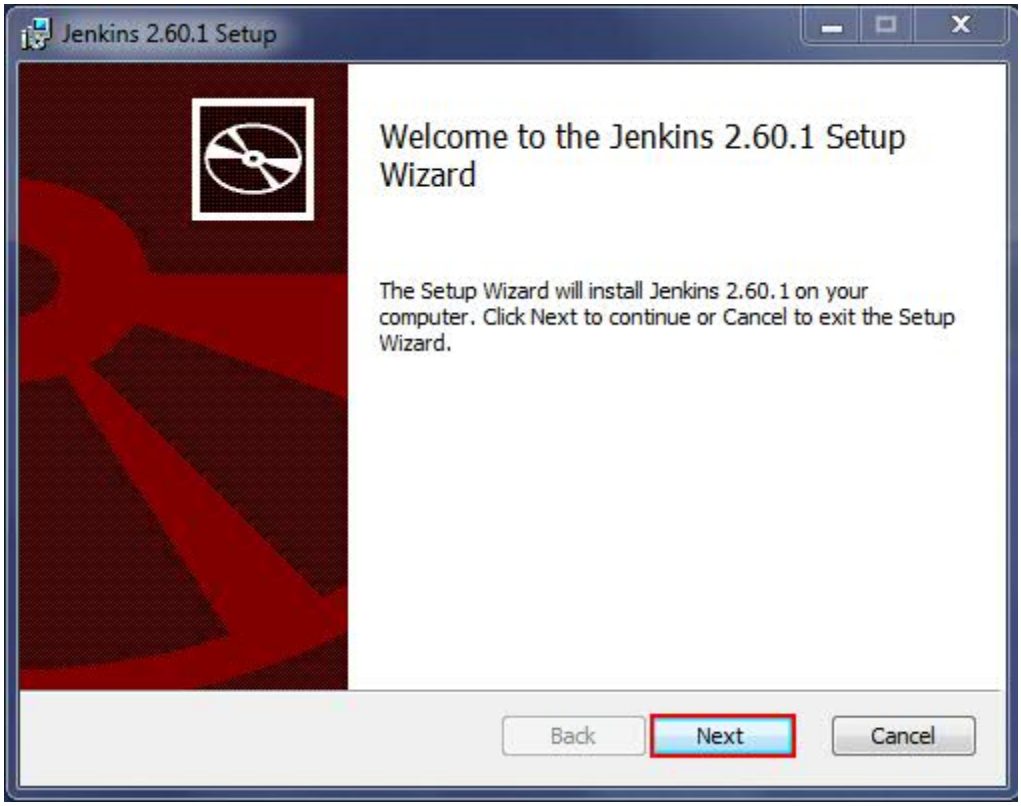
Tomcat will run 8080 port on by default.
Note: netstat -a will give the port numbers.

If you are running Jenkins on an application server, the URL to use to access Jenkins will be slightly different. For example, on a Tomcat installation by default, you can access Jenkins in your web browser at http://localhost:8080/jenkins .

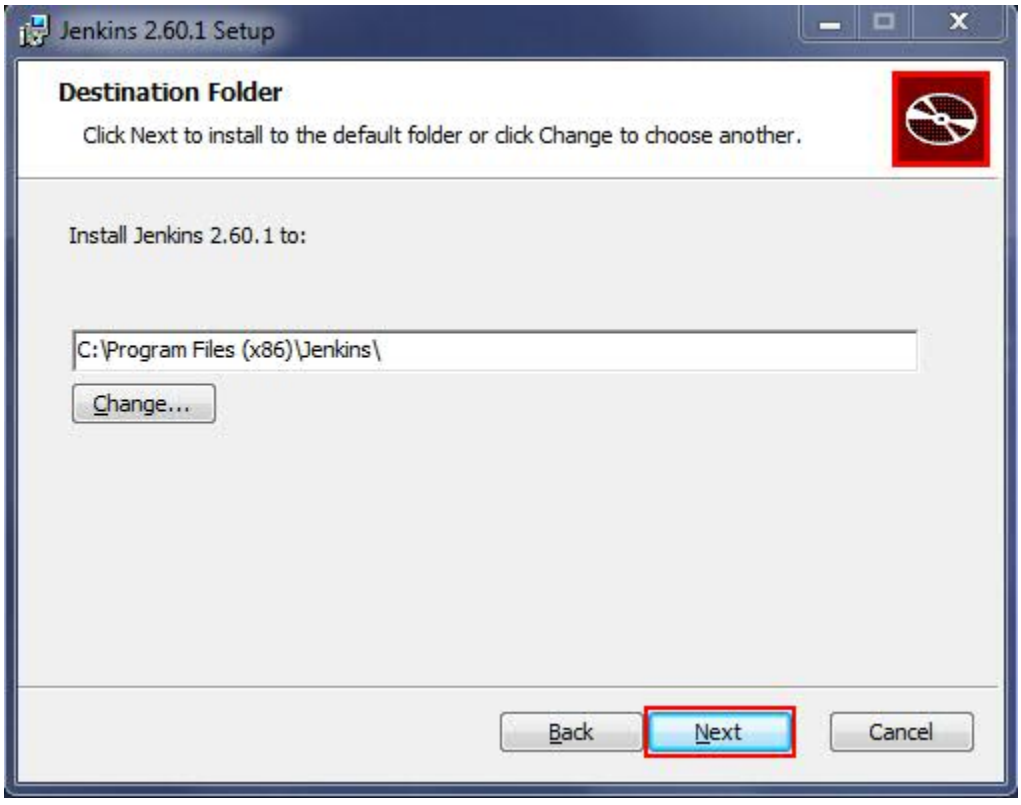
Method 3: Windows

Click on below icon.

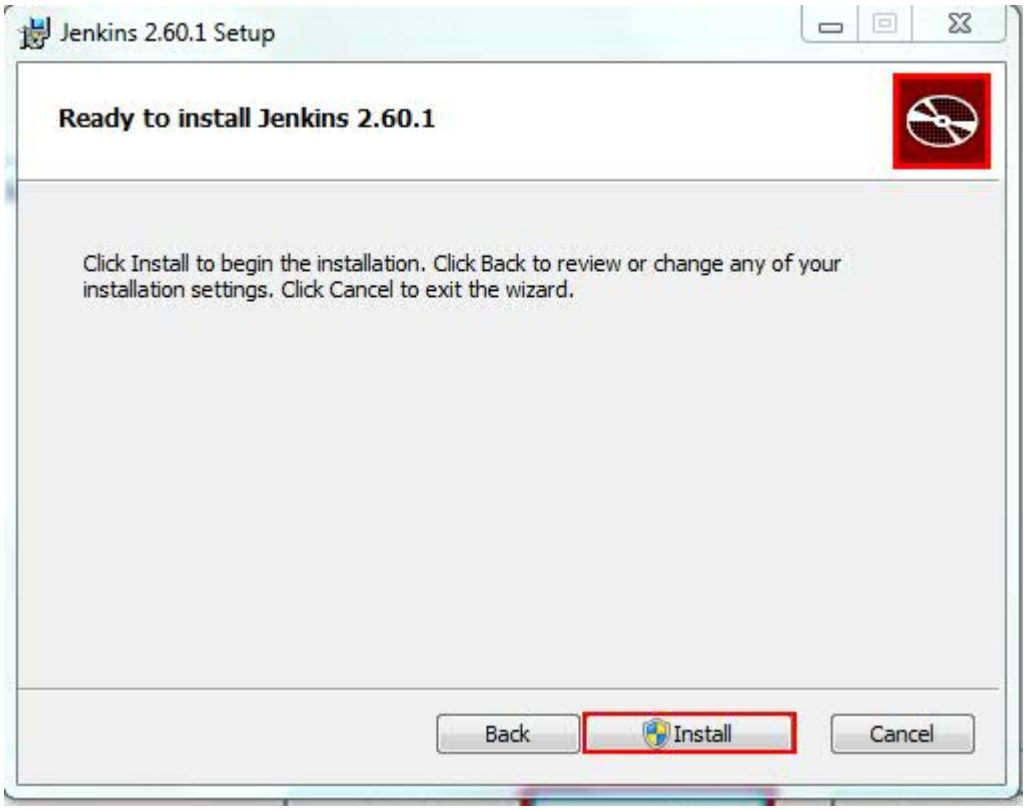
You will get the below screen



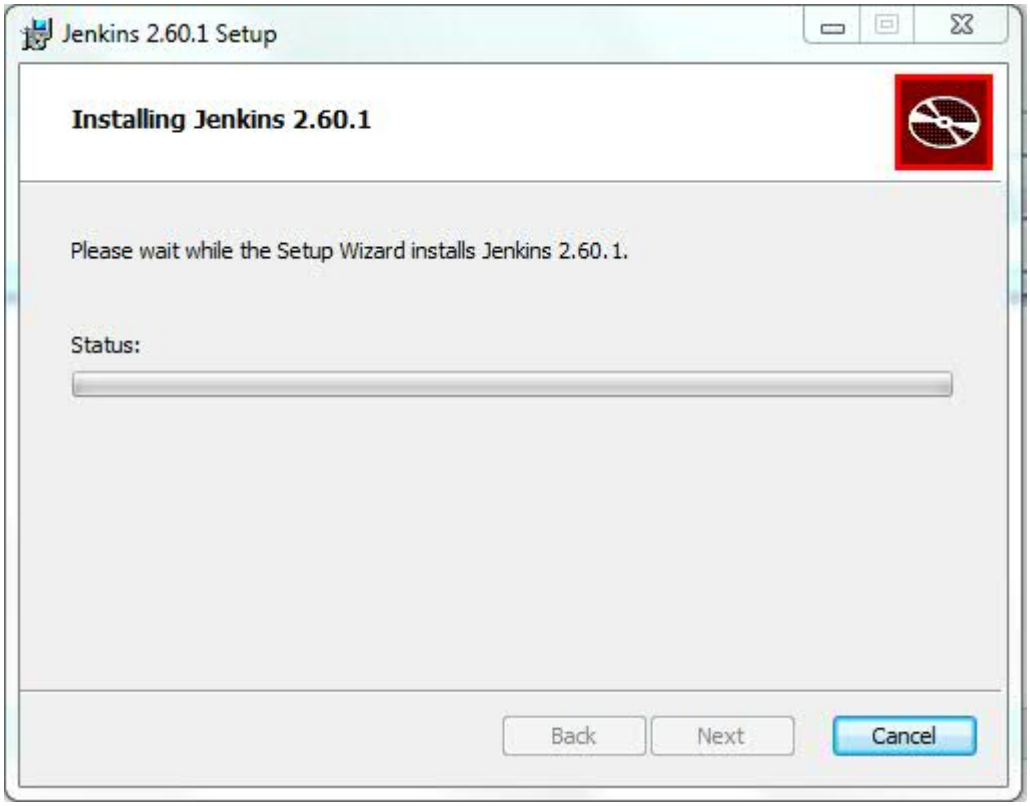
Click on Next

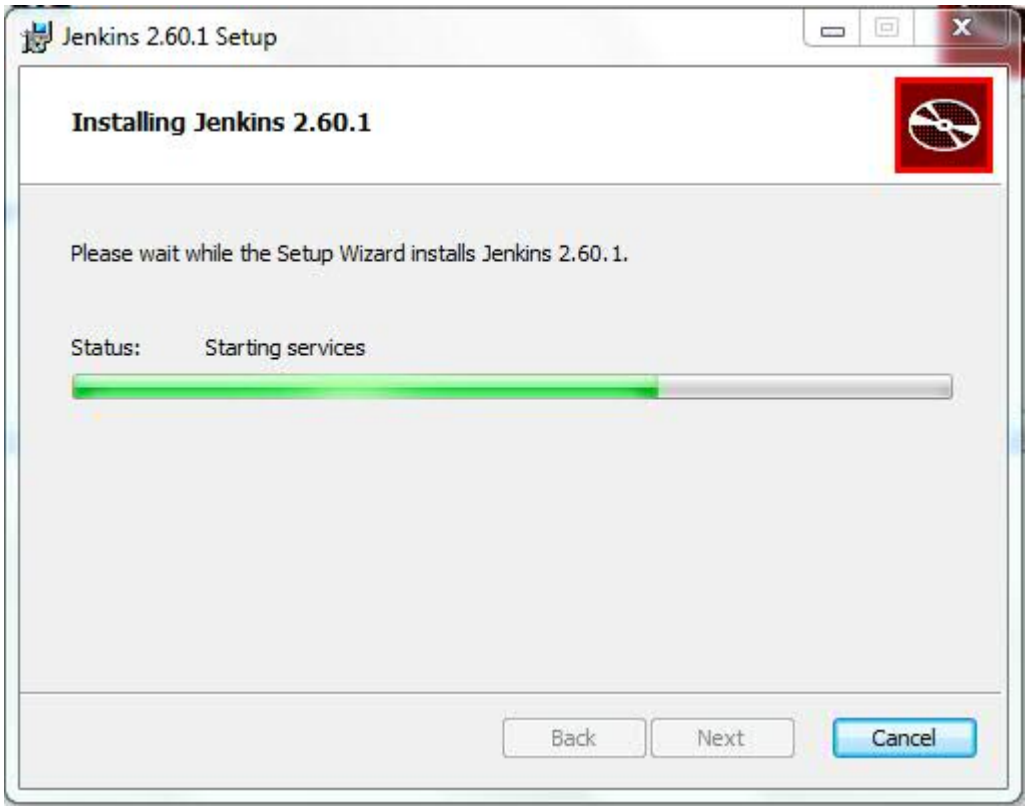


Click on Next

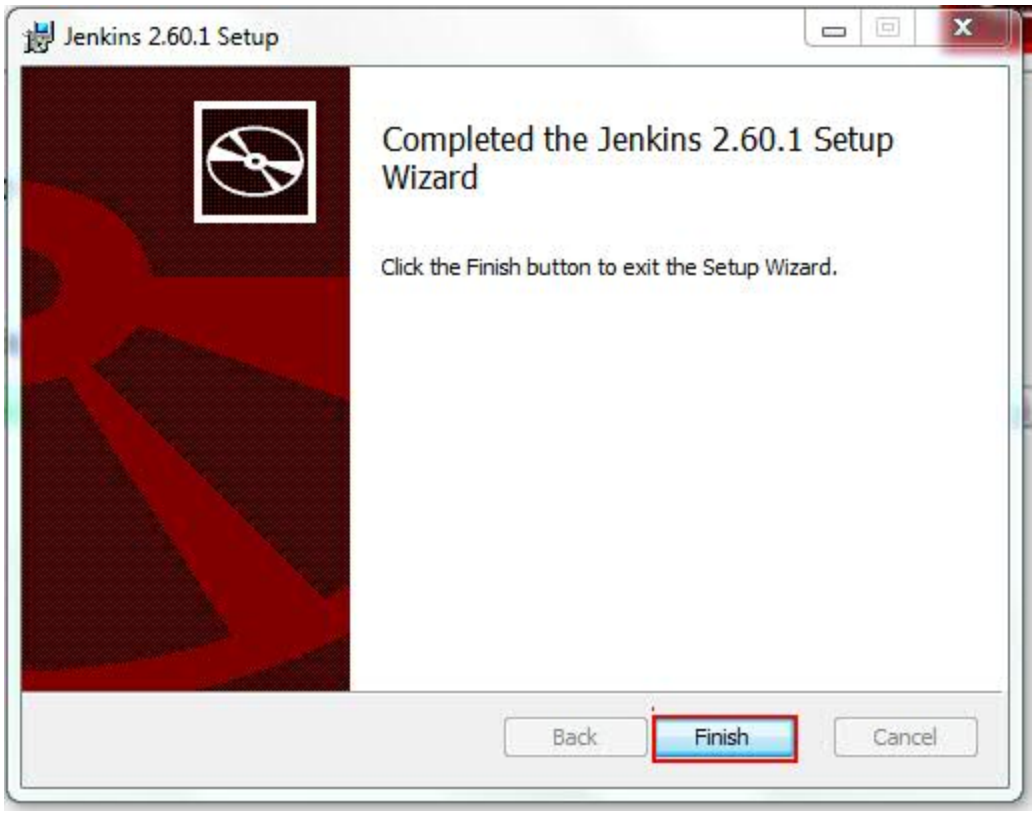


Click on Install



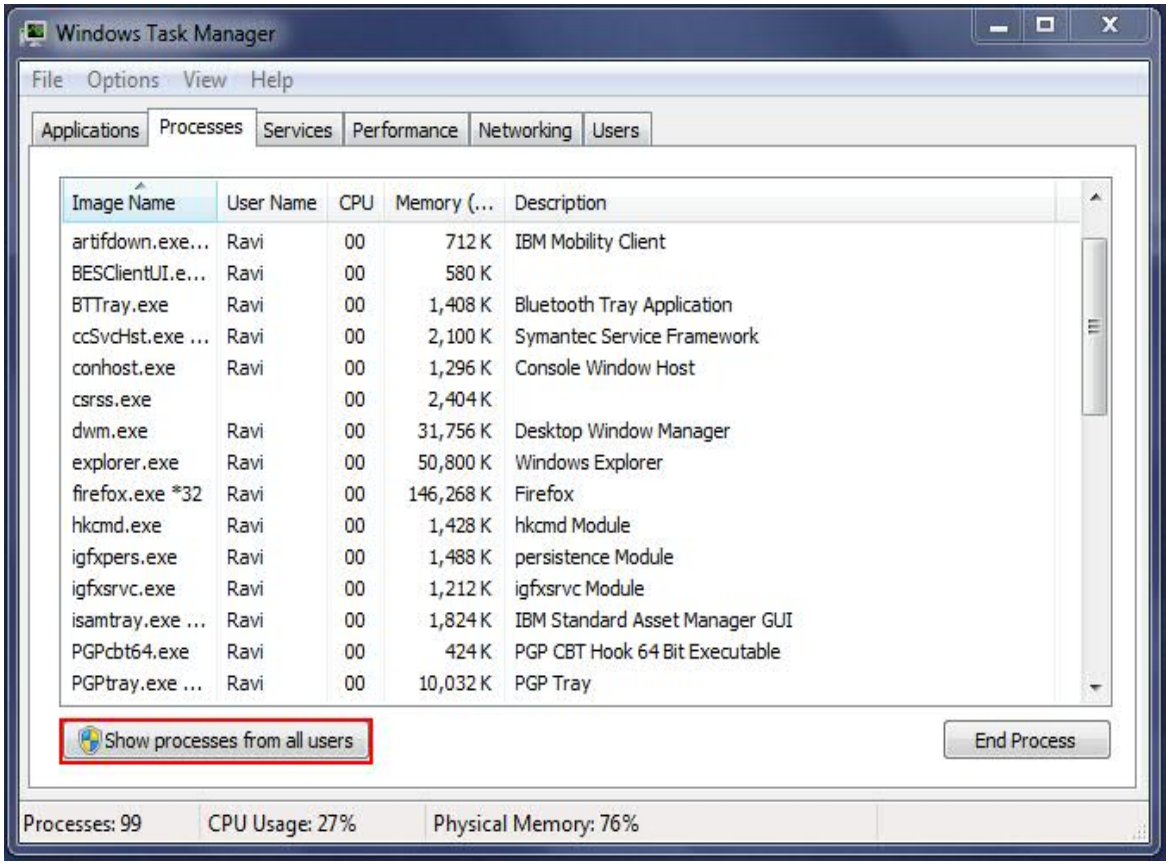


Click on Finish

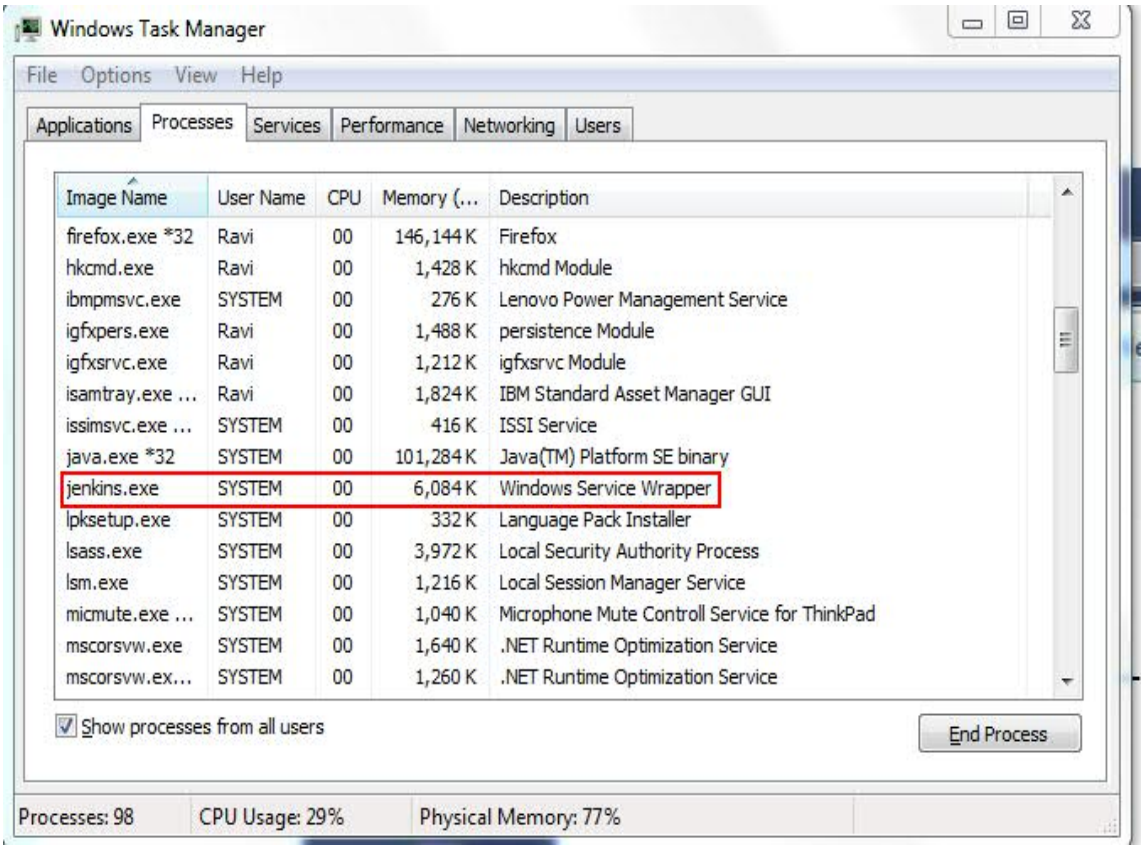


Once it successfully installed it will open by default in browser. Otherwise open the below url in browser.
http://localhost:8080

Once it got installed successfully, you will see Jenkins service in task manager list as follows.

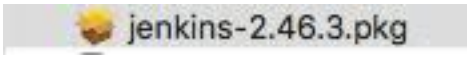


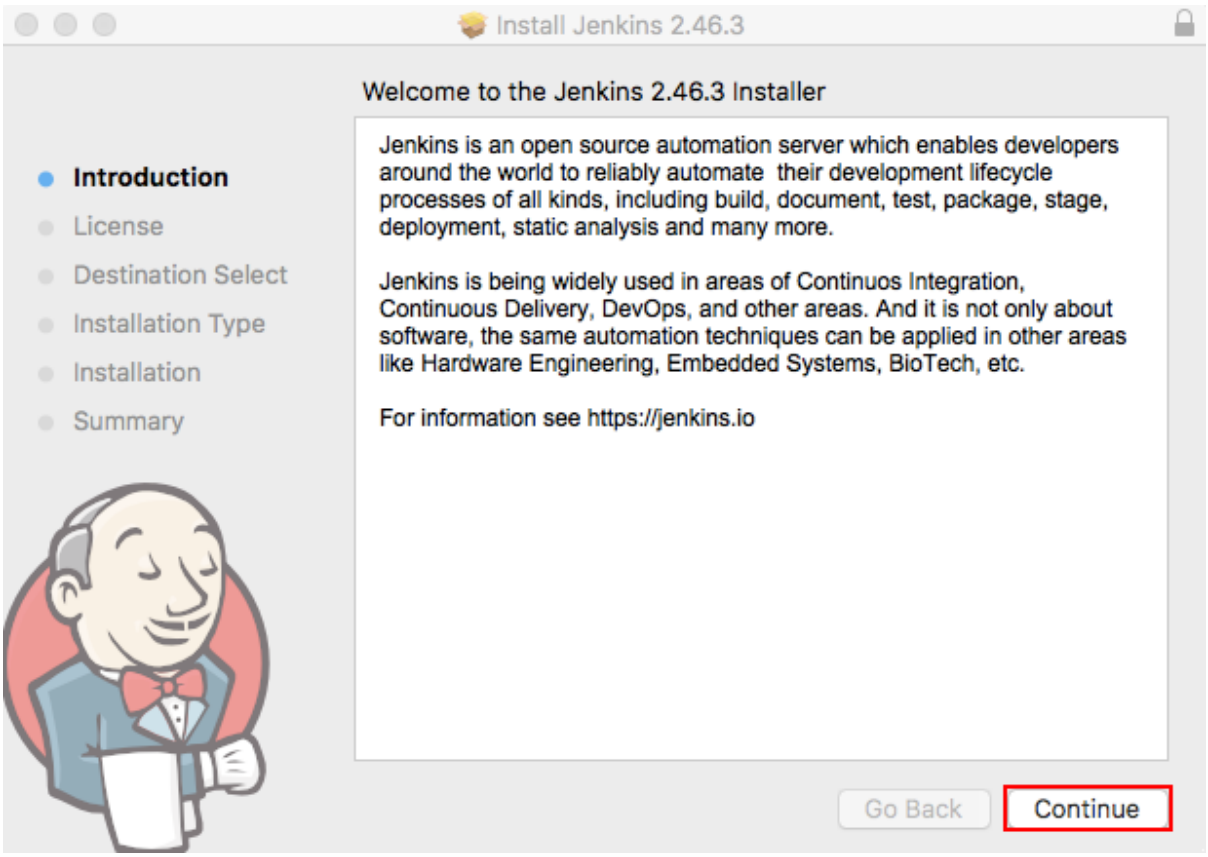
Note: Sometimes you will not see, so click on Show process from all users.



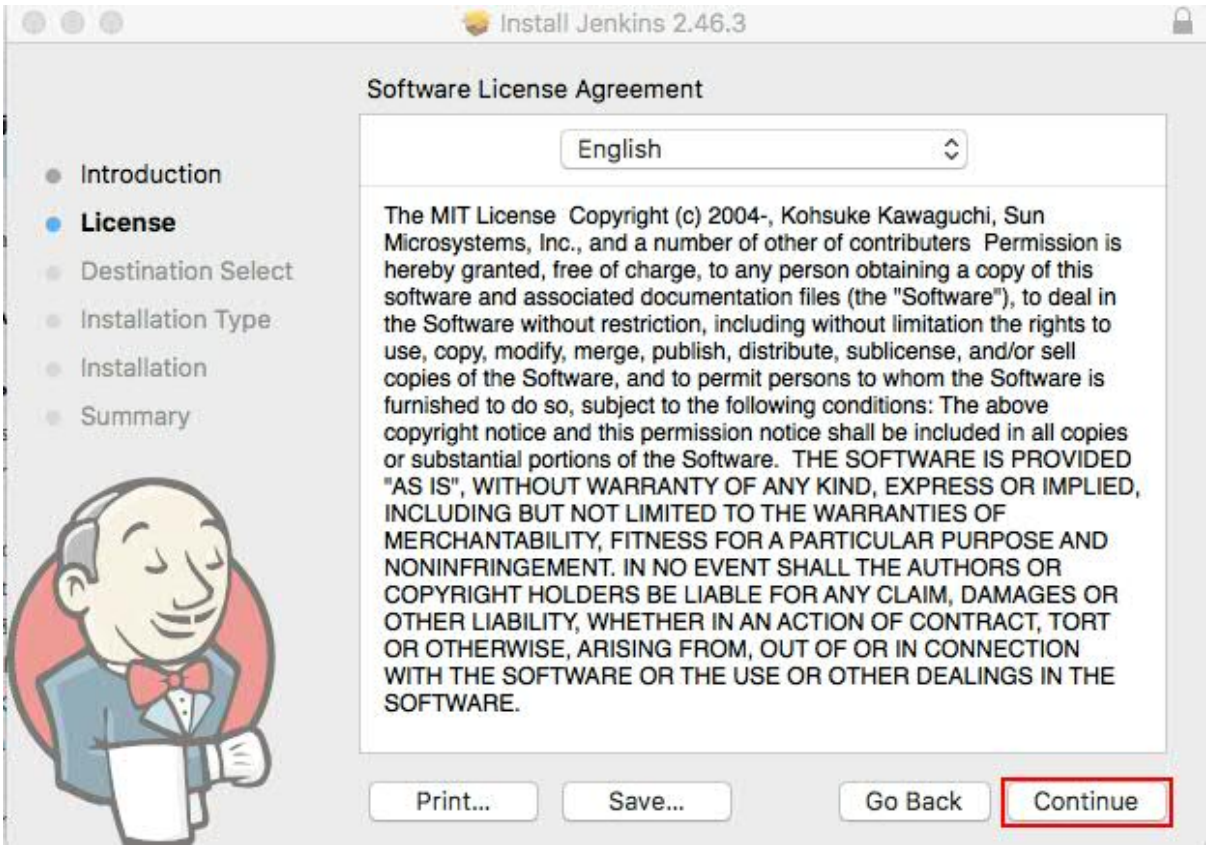
Method 3 MAC:

Click on below Jenkins package.

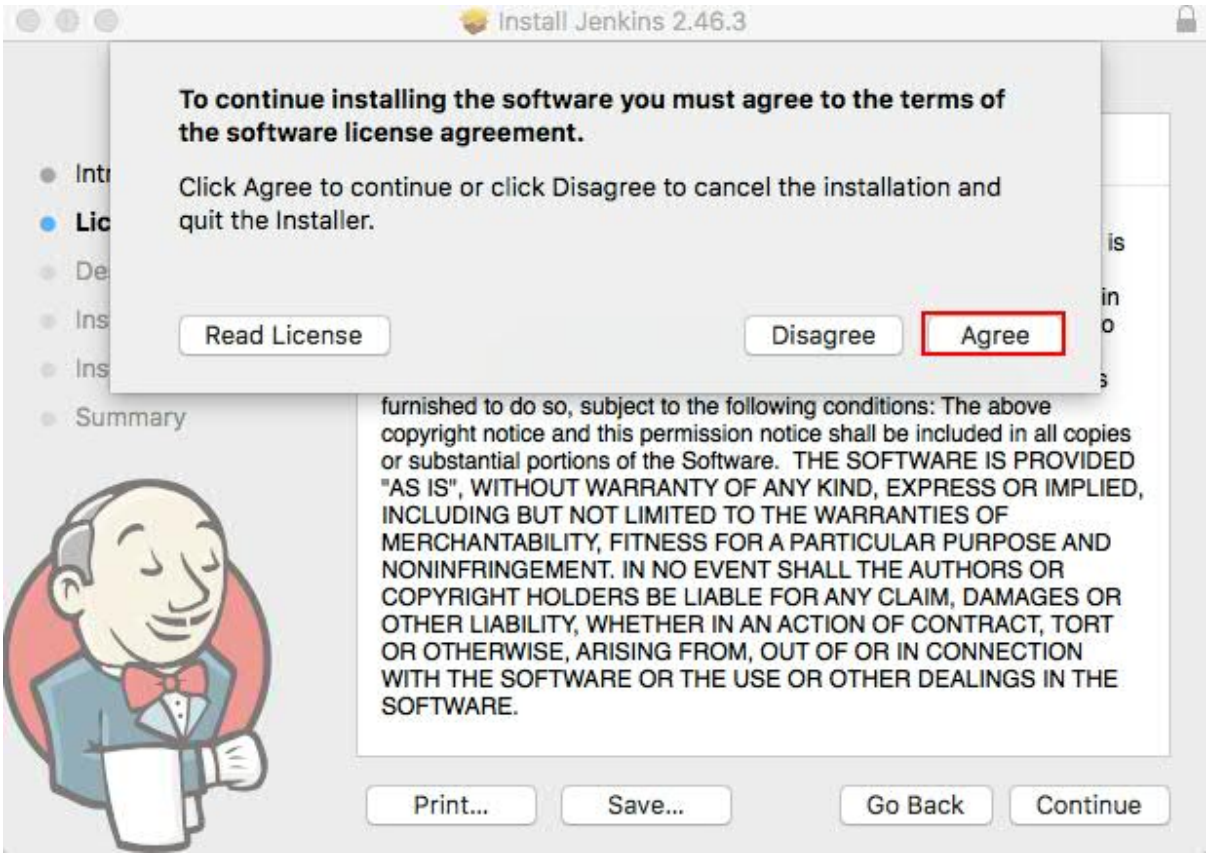




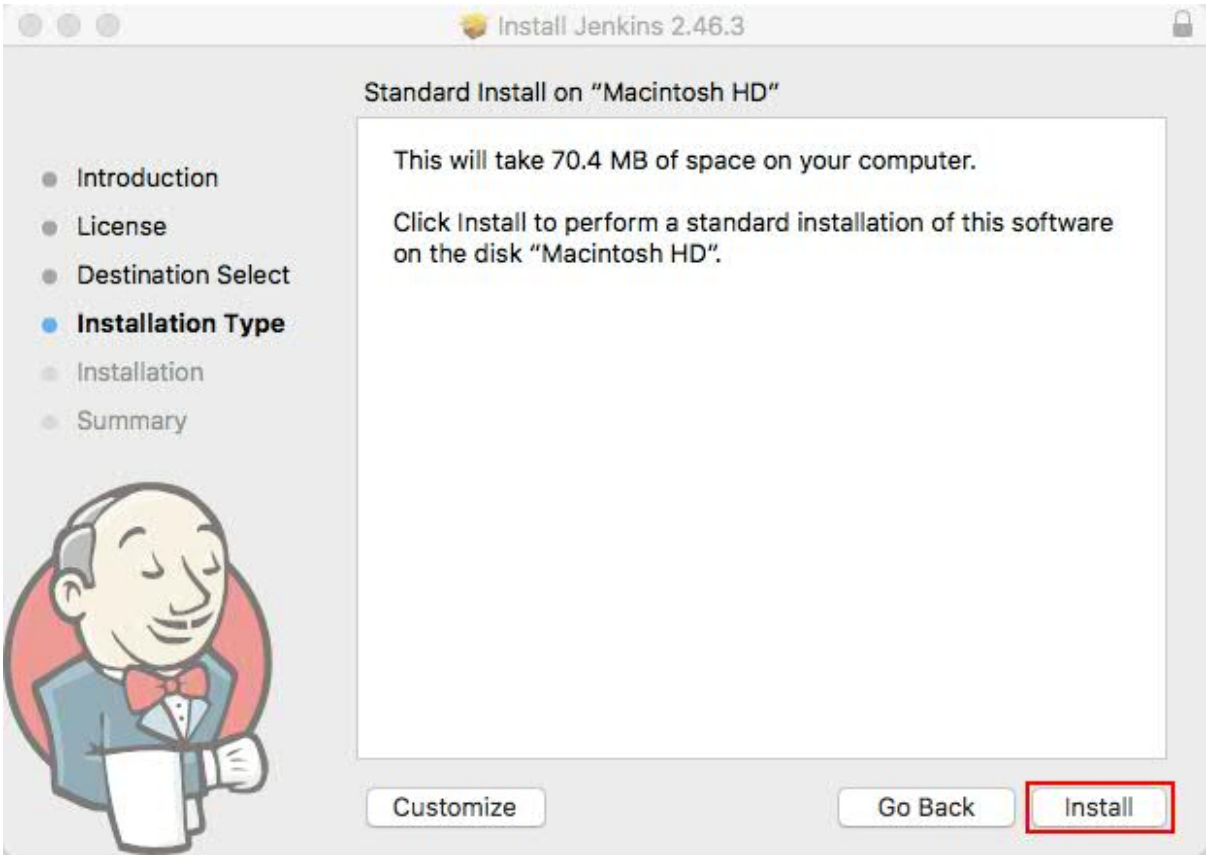
Click on Continue



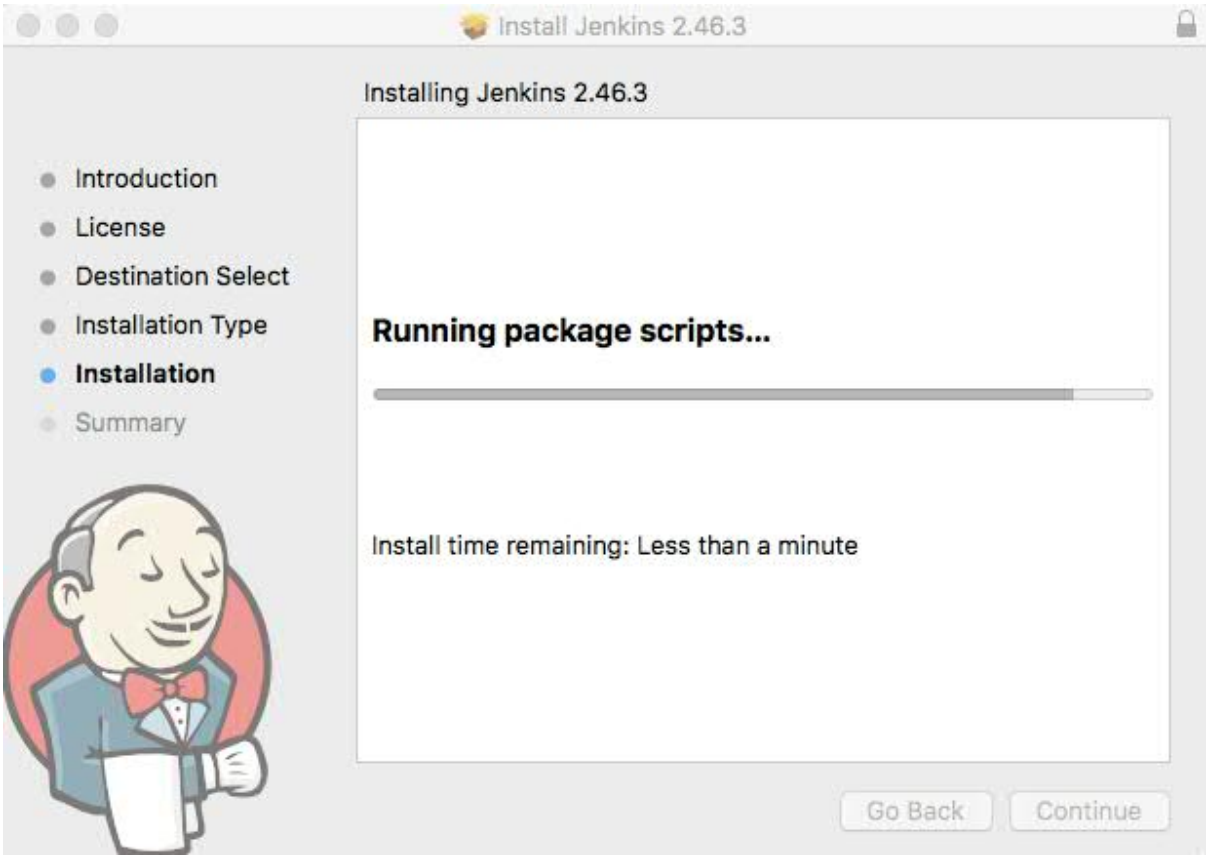
Click on Continue

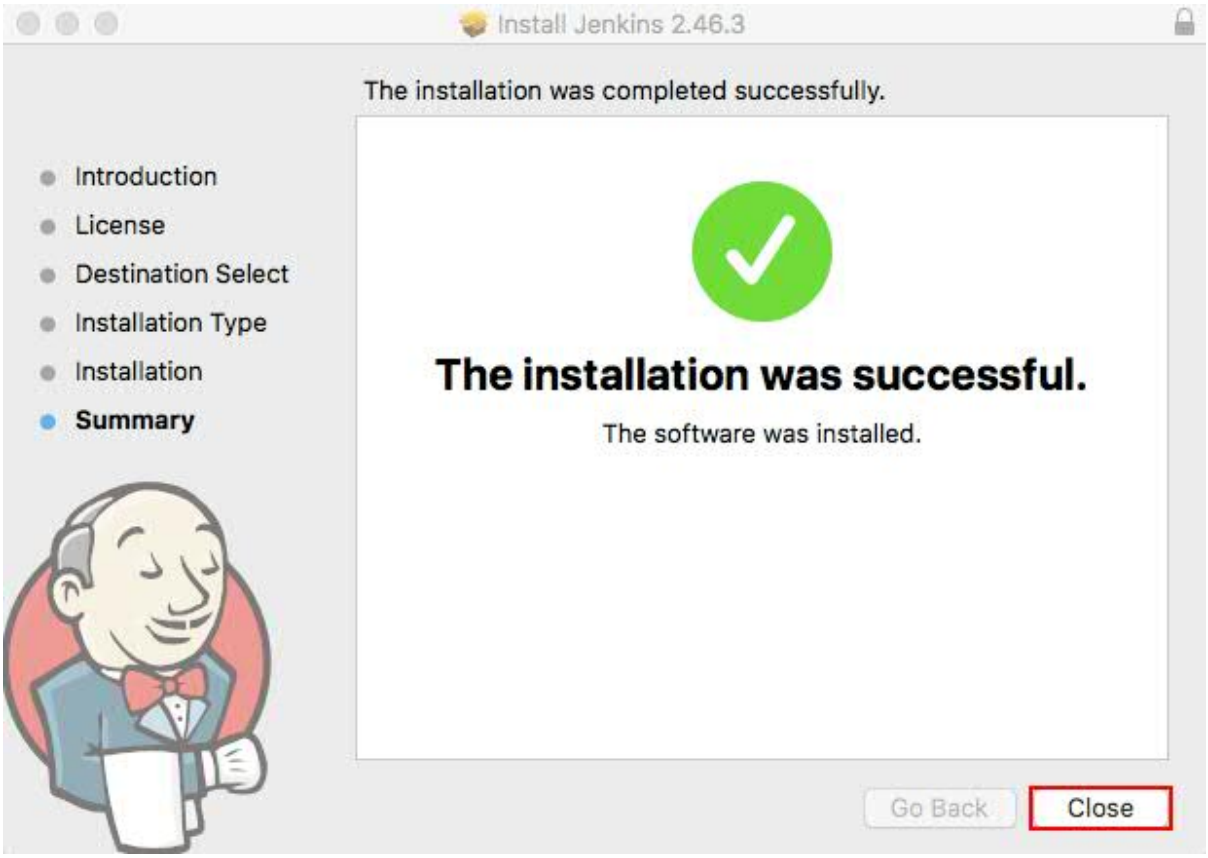


Click on Agree



Click on Install.


Install Jenkins 2.46.3
Installing Jenkins 2.46.3
Introduction
License
Destination Select
Installation Type
Installation
Summary
Running package scripts...
Install time remaining: Less than a minute
Go Back
Continue



Click on Close.

Method 3 Linux:

#wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo

rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
yum install Jenkins

Once the installation of Jenkins completed stop firewall or add the Jenkins port in Firewall in the following way.
firewall-cmd --zone=public --add-port=8080/tcp --permanent
firewall-cmd --zone=public --add-service=http --permanent
firewall-cmd --reload
firewall-cmd --list-all

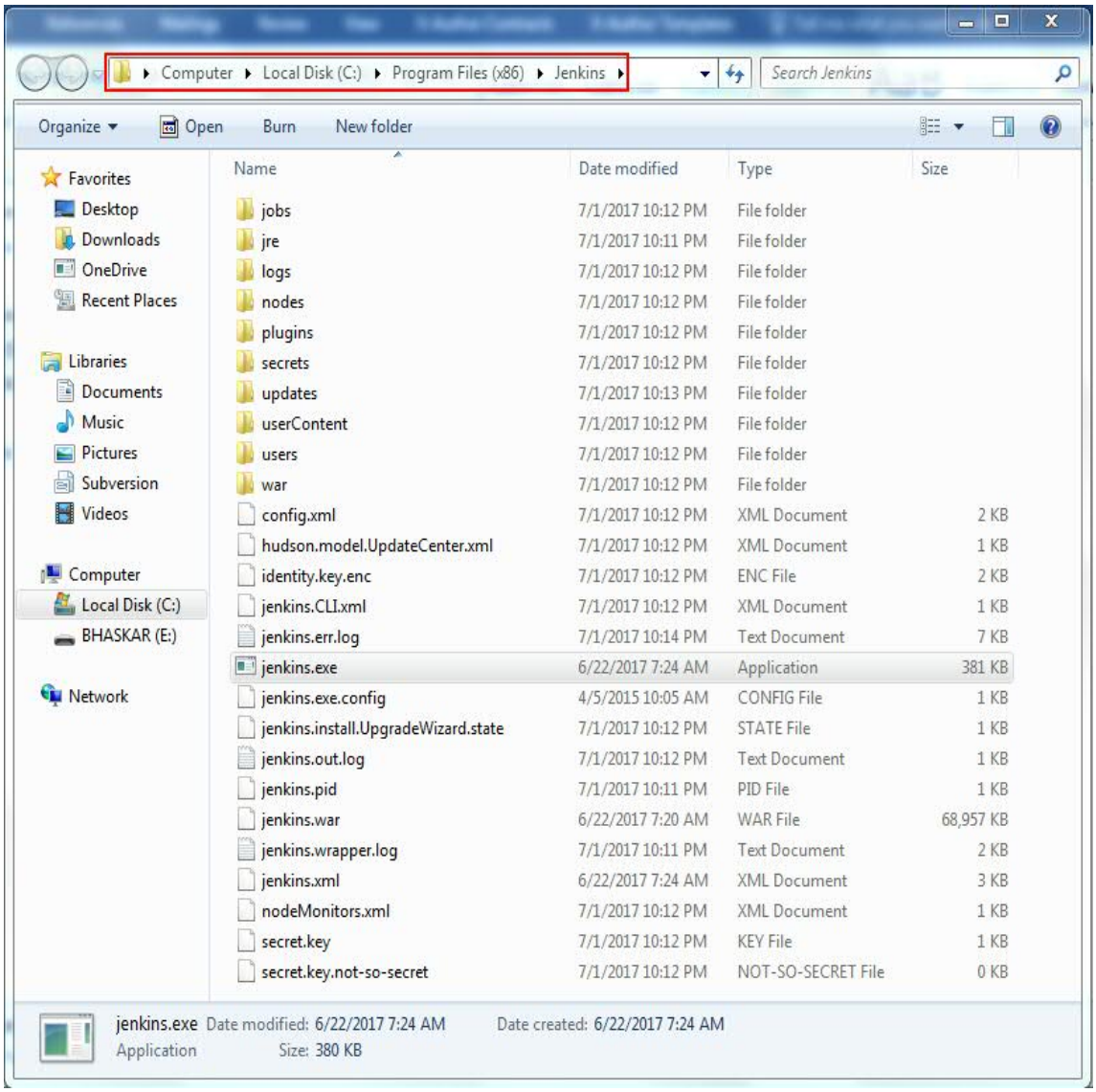
Now we can start the Jenkins Service by using systemctl as shown below
systemctl start jenkins
systemctl status Jenkins

Now we can access the Jenkins webpage by using the Url: http://ipaddress of server:8080 or http://localhost:8080

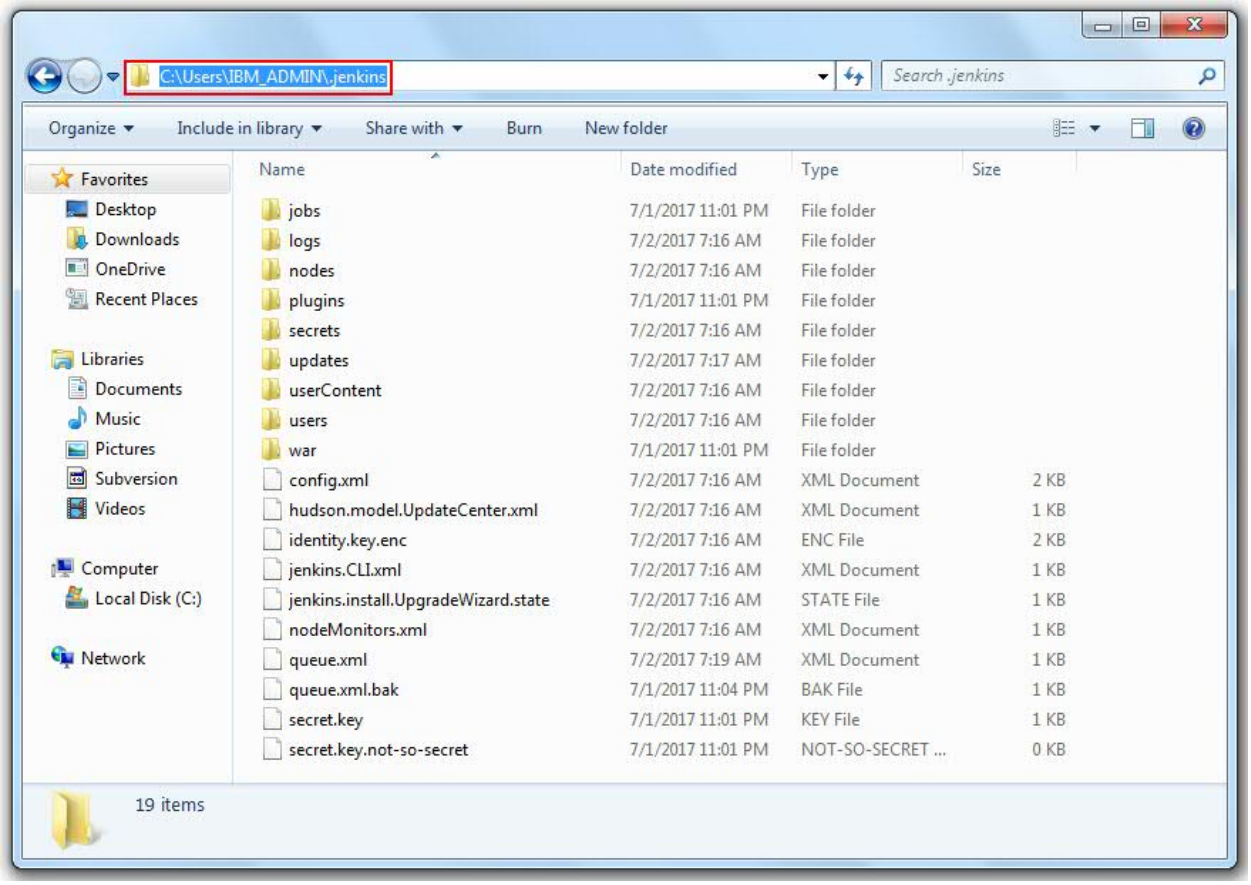
Reference Url: https://wiki.jenkins.io/display/JENKINS/Installing+Jenkins+on+Red+Hat+distributions

---

Below screen, installed the Jenkins using .exe file.



Below screen, installed the Jenkins using .war file (java -jar jenkins.war).



Open http://localhost:8080/systemInfo and check JEKNINS_HOME from browser.

.jenkins : This is the default Jenkins home directory (may be .hudson in older installations) and it will be placed in user's home directory (C:\Users\IBM_ADMIN\ ---> Windows & /Users/BhaskarReddy --> MAC).

Jenkins home directory contains the below sub directories and configuration files (.xml).

```
+- jobs
  +- [JOBNAME]      :Sub directory for each job
    +- config.xml     : Job configuration file
    +- latest          : Symbolic link to the last successful build)
    +- builds
      +- [BUILD_ID]    : for each build one build id
        +- build.xml     : build result summary
        +- log           : log file
        +- changelog.xml  (change log)
+- logs          ()
+- nodes         ()
+- plugins       : This directory contains any plugins that you have installed.
+- secrets       ()
+- updates          : This is an internal directory used by Jenkins to store information
                     about available plugin updates.
```

+- userContent : You can use this directory to place your own custom content onto your Jenkins server. You can access files in this directory at http://localhost/jenkins/userContent (if you are running Jenkins on an application server) or http://localhost:8080/userContent (if you are running in stand-alone mode).
+- users : If you are using the native Jenkins user database, user accounts will be stored in this directory.

+- war : This directory contains the expanded web application. When you start Jenkins as a stand-alone application, it will extract the web application into this directory.

+- config.xml (jenkins root configuration)
+- *.xml (other site-wide configuration files)
+- fingerprints (stores fingerprint records)

---

http://localhost:8080/configure

Home directory: By default, Jenkins stores all of its data in this directory on the file system. Under the Advanced section, you can choose to store build workspaces and build records elsewhere.

There are a few ways to change the Jenkins home directory:

- Edit the JENKINS_HOME variable in your Jenkins configuration file (e.g. /etc/sysconfig/jenkins on Red Hat Linux).
- Use your web container's admin tool to set the JENKINS_HOME environment variable.
- Set the environment variable JENKINS_HOME before launching your web container, or before launching Jenkins directly from the WAR file.
- Set the JENKINS_HOME Java system property when launching your web container, or when launching Jenkins directly from the WAR file.
- Modify web.xml in jenkins.war (or its expanded image in your web container). This is not recommended.

This value cannot be changed while Jenkins is running.
It is shown here to help you ensure that your configuration is taking effect.

Ex: /Users/BhaskarReddy/.jenkins is for my Jenkins which is installed in my local MAC.

Workspace Root Directory: Specifies where Jenkins will store workspaces for builds that are executed on the master.

Build Record Root Directory: Specifies where Jenkins will store build records on the file system. This includes the console output and other metadata generated by a build.

System Message: This message will be displayed at the top of the Jenkins main page.

# of executors: It shows the ow many builds run at a time. E.g.: If give 2, here two builds are running.

Labels:

Usage: Controls how Jenkins schedules builds on this node.

Quiet period:

SCM checkout retry count:

Restrict project naming:

Naming Strategy

Strategy

Default ---> This is the default configuration and allows the user to choose any name they like.

Pattern ----> Define a pattern (regular expression) to check whether the job name is valid or not. Forcing the check on existing jobs, will allow you to enforce a naming convention on existing jobs - e.g. even if the user does not change the name, it will be validated with the given pattern at every submit and no updates can be made until the name confirms.

This option does not affect the execution of jobs with non-compliant names. It just controls the validation process when saving job configurations.

Global properties

Environment variables

Tool Locations

SonarQube servers

etc.…

To Install any Jenkins Plugin, follow below steps

Manage Jenkins ---> Manage Plugins ---> Select the Plugin name (HTML Published plugin) ---> Install Without Restart

Package Names:

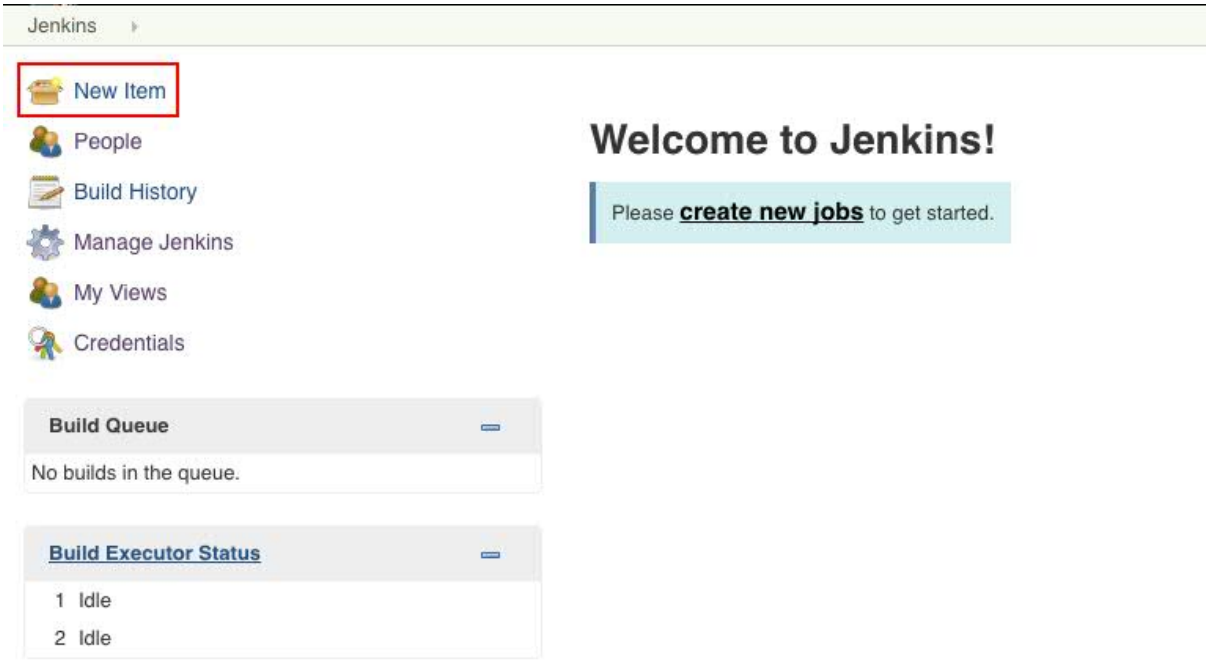
SafeRestartPlugin:
SonarQube Scanner:
JavaDoc Plugin
jQuery Plugin
Next Build Number Plugin: In UI (Set Next Build Number)
Email Extension Plugin: Is used to send the mail from Jenkins.
Delivery Pipeline Plugin: To move war/ear/jar into UCD.
Maven Integration plugin: For Maven project.
Pipeline Maven Integration Plugin: To see Maven Project option while creating job.
Run Condition Plugin
Conditional BuildStep
Parameterized Trigger plugin:
Cloud Foundry Plugin: To deploy app into Bluemix using cloud foundry.
Embeddable Build Status:

JobConfigHistory Plugin: This plugin saves a copy of the configuration file of a job (config.xml) for every change made and of the system configuration. You can also see what changes have been made by which user if you configured a security policy.
Repository browser:

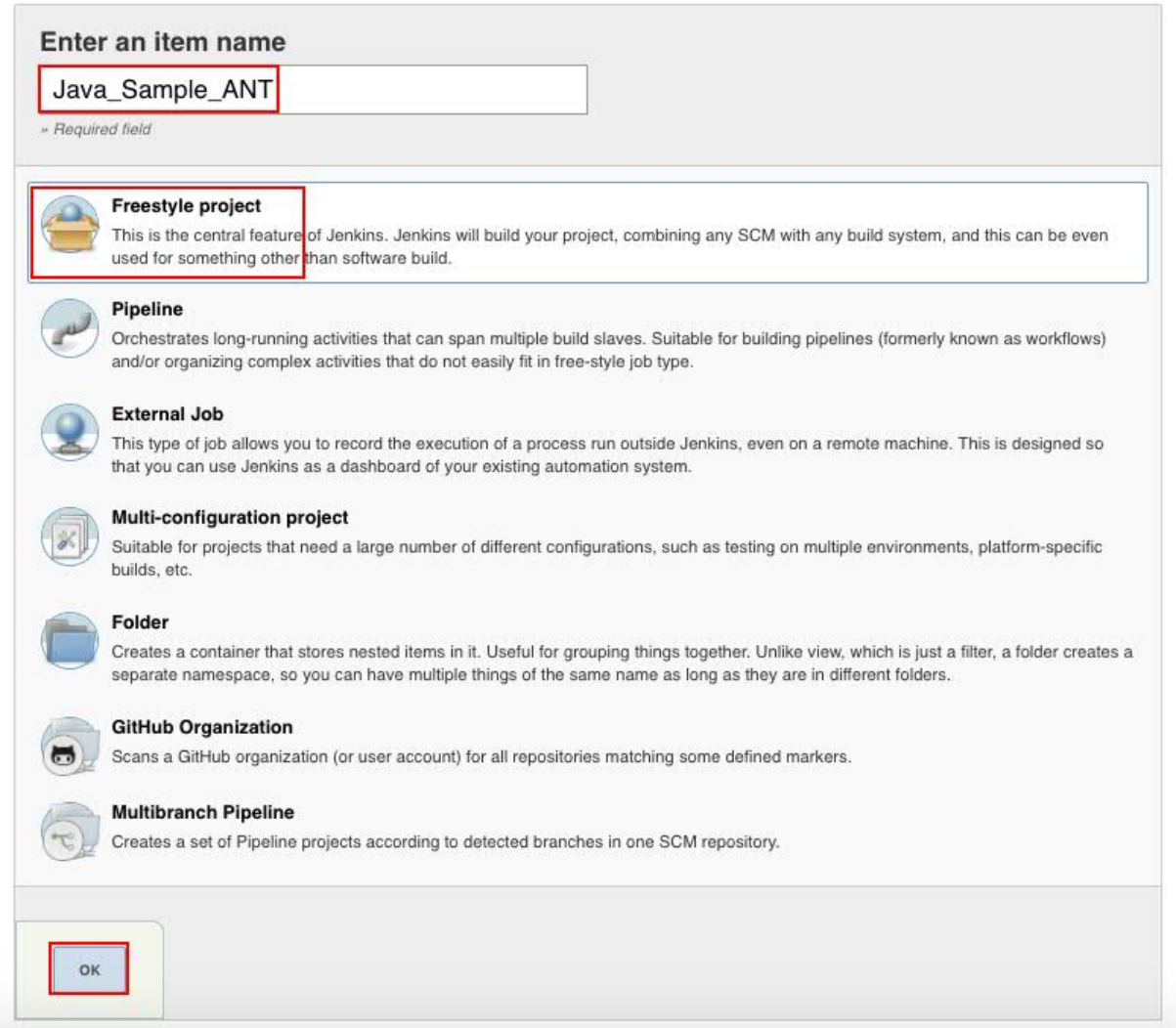
Role-based Authorization Strategy:
Slack Notification Plugin:
Cobertura Plugin: In UI we will see as Coverage Trend.
Artifactory Plugin: For JFrog …. Or any other repositories.
Deploy to container Plugin: For deploying war/ear into Application Server or Webserver.
Build History Metrics plugin:
Hudson global-build-stats plugin:
Delivery Pipeline View:
Enable project-based security

Create the project/job in Jenkins

Step 1: Login into the Jenkins, go to the Jenkins dashboard left side top corner, click on **New Item**.



Step 2: Enter the project name in **Enter an item name** input box and select the **Freestyle project** and click on **OK** Button.

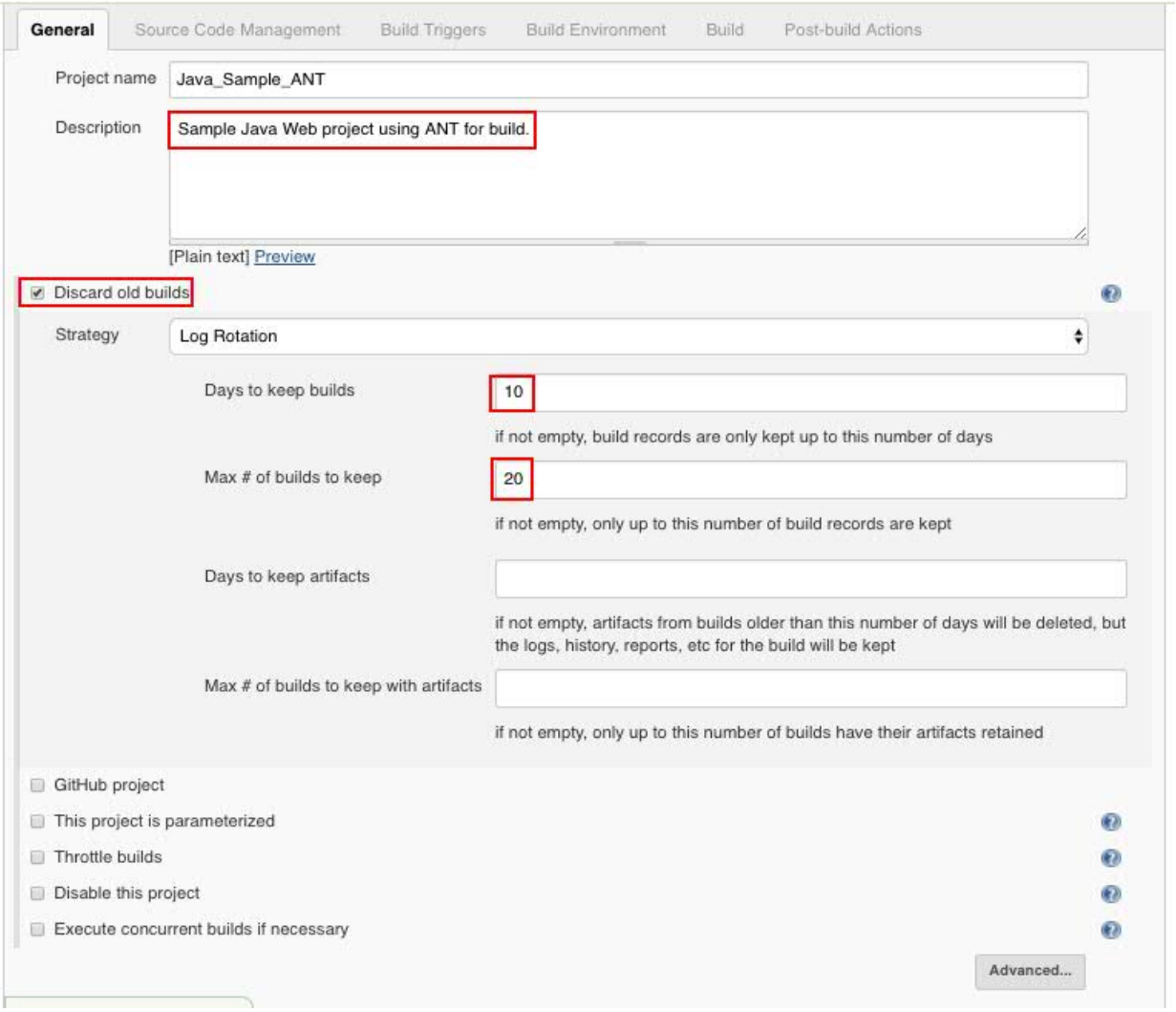


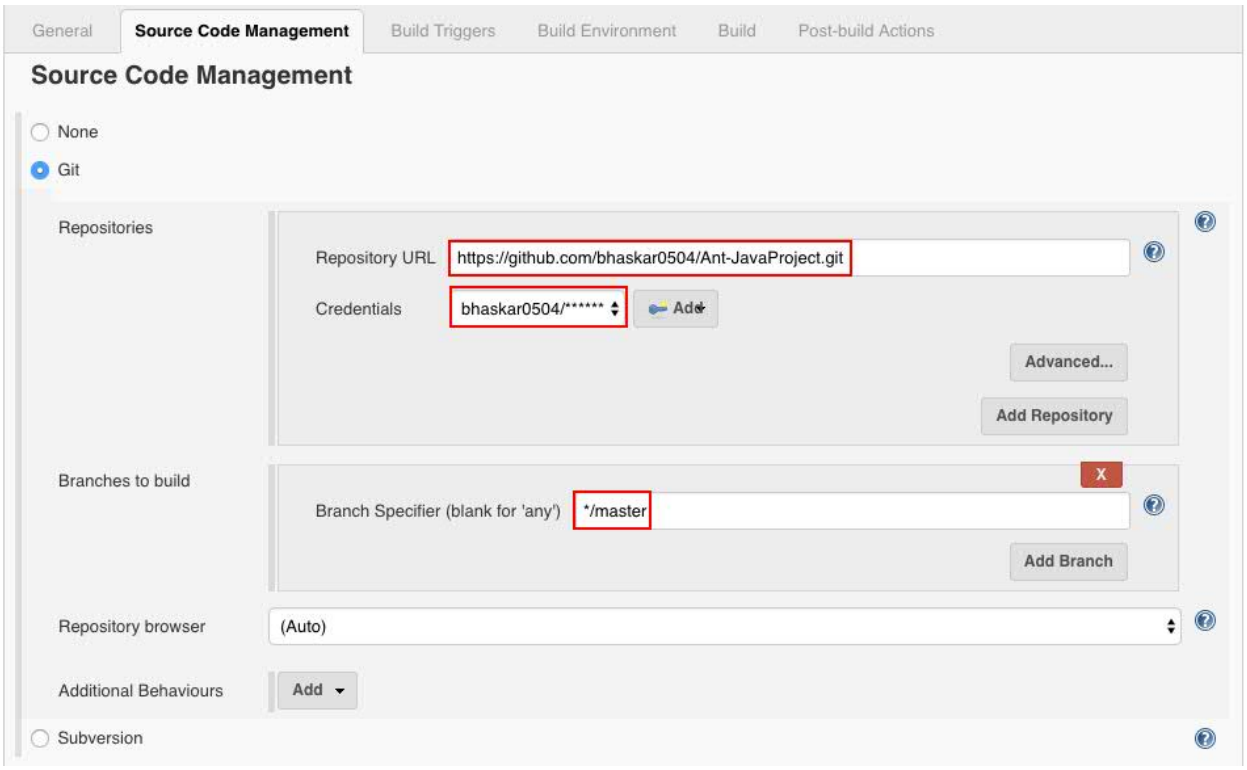
Freestyle project: This is the central feature of Jenkins. Jenkins will build your project combining any SCM and any build system.

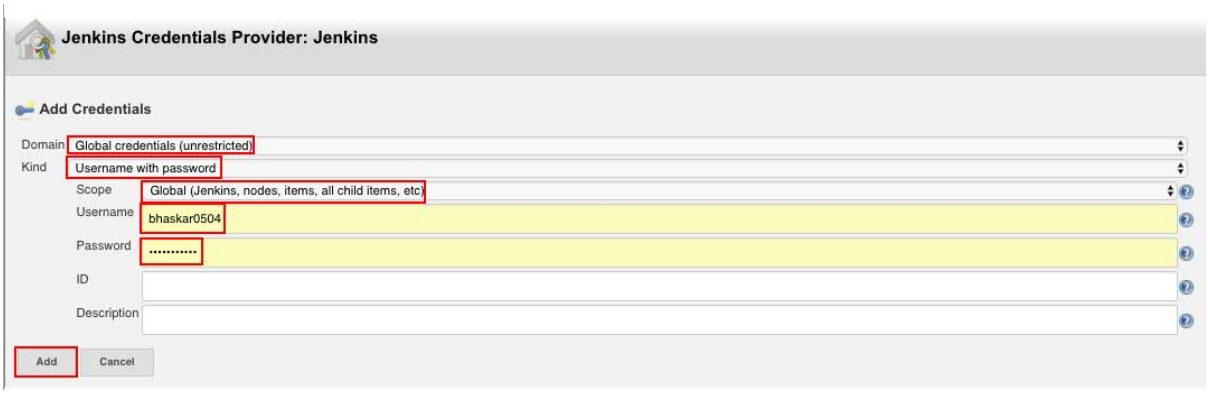
A Free-Style project is a project that can incorporate almost any type of build. The Free-Style project is the more "generic" form of a project. You can execute shell/dos scripts, invoke ant, and a lot more. Majority of the plugins are written to use the free-style project.

Maven project: A maven project is a project that will analyze the pom.xml file in greater detail and produce a project that's geared towards the targets that are invoked. The maven project is smart enough to incorporate build targets like the javadoc or test targets and automatically setup the reports for those targets.

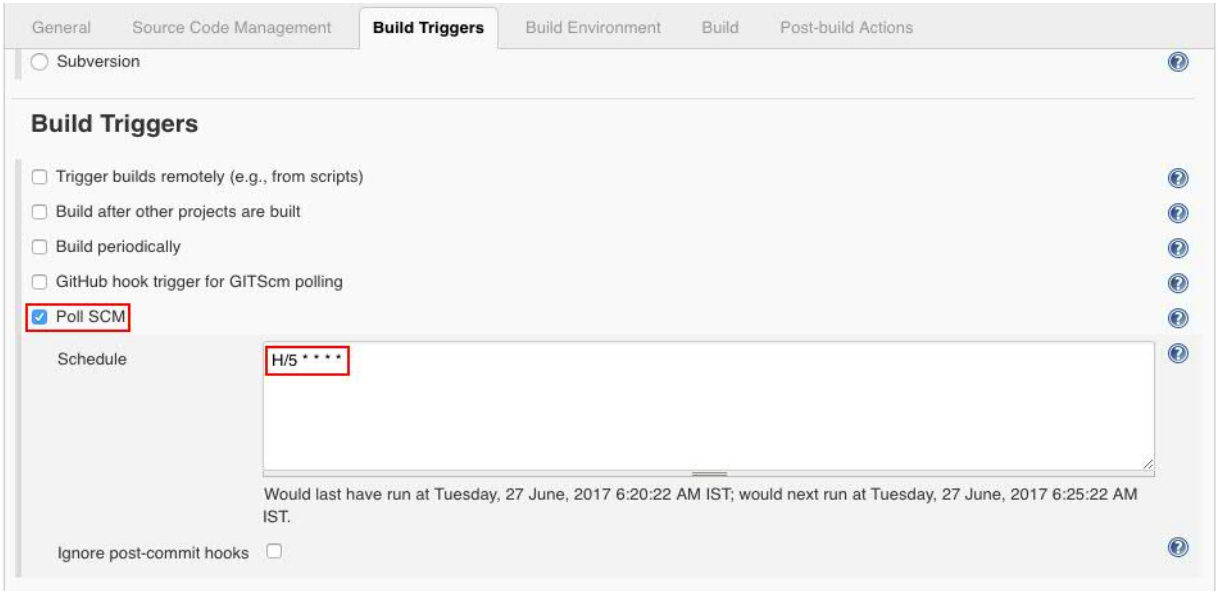
Multi-configuration project: The “multiconfiguration project” (also referred to as a “matrix project”) lets you run the same build job in many different configurations. This powerful feature can be useful for testing an application in many different environments, with different databases, or even on different build machines. We will be looking at how to configure multiconfiguration build jobs later on in the book.
Monitor an external job: The “Monitor an external job” build job lets you keep an eye on non-interactive processes, such as cron jobs.

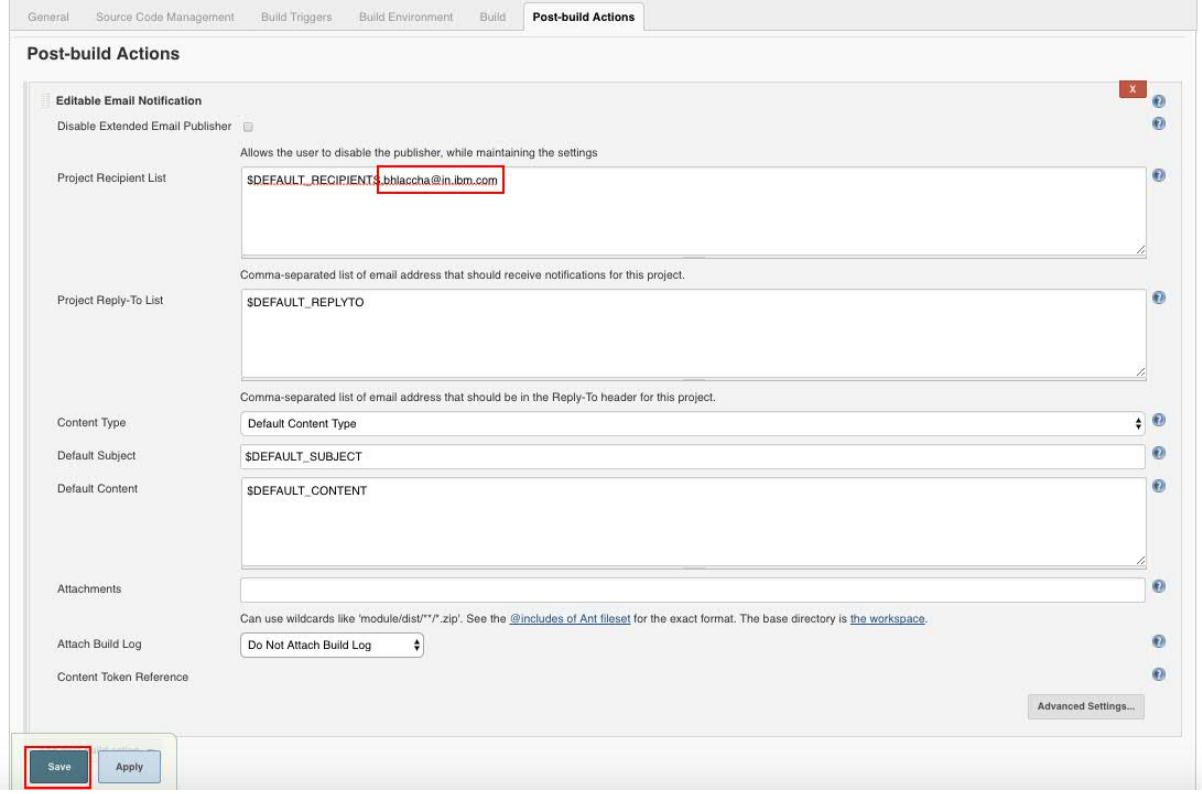

General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Project name
Java_Sample_ANT
Description
Sample Java Web project using ANT for build.
[Plain text] Preview
Discard old builds
Strategy
Log Rotation
Days to keep builds
10
if not empty, build records are only kept up to this number of days
Max # of builds to keep
20
if not empty, only up to this number of build records are kept
Days to keep artifacts
if not empty, artifacts from builds older than this number of days will be deleted, but the logs, history, reports, etc for the build will be kept
Max # of builds to keep with artifacts
if not empty, only up to this number of builds have their artifacts retained
GitHub project
This project is parameterized
Throttle builds
Disable this project
Execute concurrent builds if necessary
Advanced...


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Source Code Management
None
Git
Repositories
Repository URL
https://github.com/bhaskar0504/Ant-JavaProject.git
Credentials
bhaskar0504/******
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add
Subversion

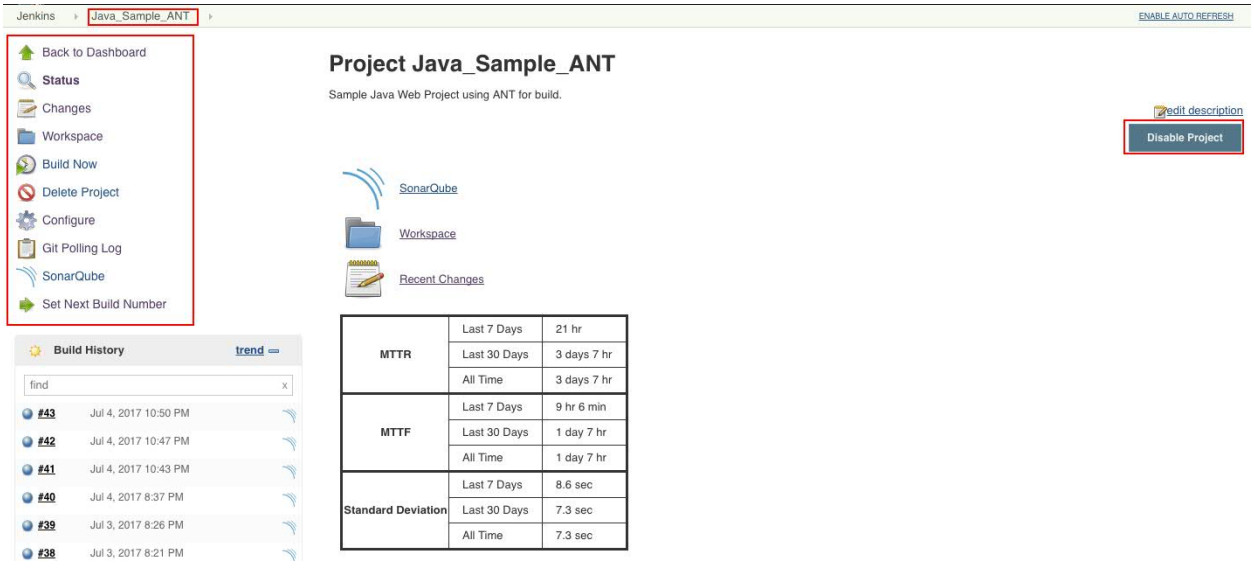

Jenkins Credentials Provider: Jenkins
Add Credentials
Domain
Global credentials (unrestricted)
Kind
Username with password
Scope
Global (Jenkins, nodes, items, all child items, etc)
Username
bhaskar0504
Password
ID
Description
Add
Cancel

Specify when and how your build should be triggered. The following example polls the Git repository every 5 min. It triggers a build, if something has changed in the repo.





Once you click on Save button it will come to that particular dashboard as follows.



Disable Build:
A disabled Build will not be executed until you enable it again. This option often comes in handly to temporarily suspend a build during maintenance work or major refactoring.

Once the project is configured in Jenkins then all future builds are automated. It has basic reporting features like status and weather reports (job health).

Status of the build	Description
	Failed
	Unstable
	Success
	Pending
	Disabled
	Aborted

Figure a: Build status

Job health	Description
	No recent builds failed
	20-40% of recent builds failed
	40-60% of recent builds failed
	60-80% of recent builds failed
	All recent builds failed
	Unknown status

Figure b: Weather reports

Create the Maven project/job in Jenkins

Method 1:

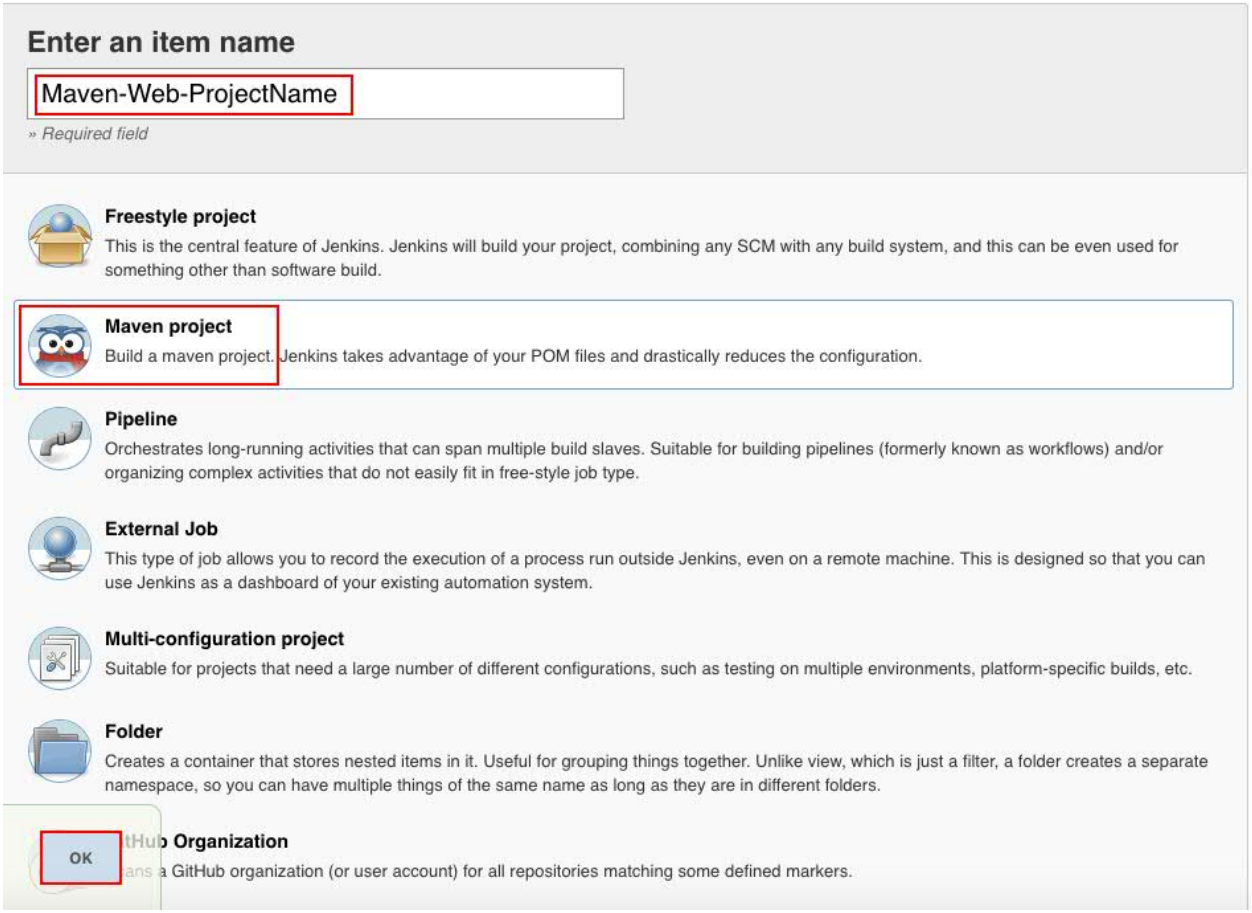
Install the **Pipeline Maven Integration Plugin** and follow the below steps.

Create the Job using Freestyle project and in the Build section click on Add build step and select the Invoke Top level Maven targets.

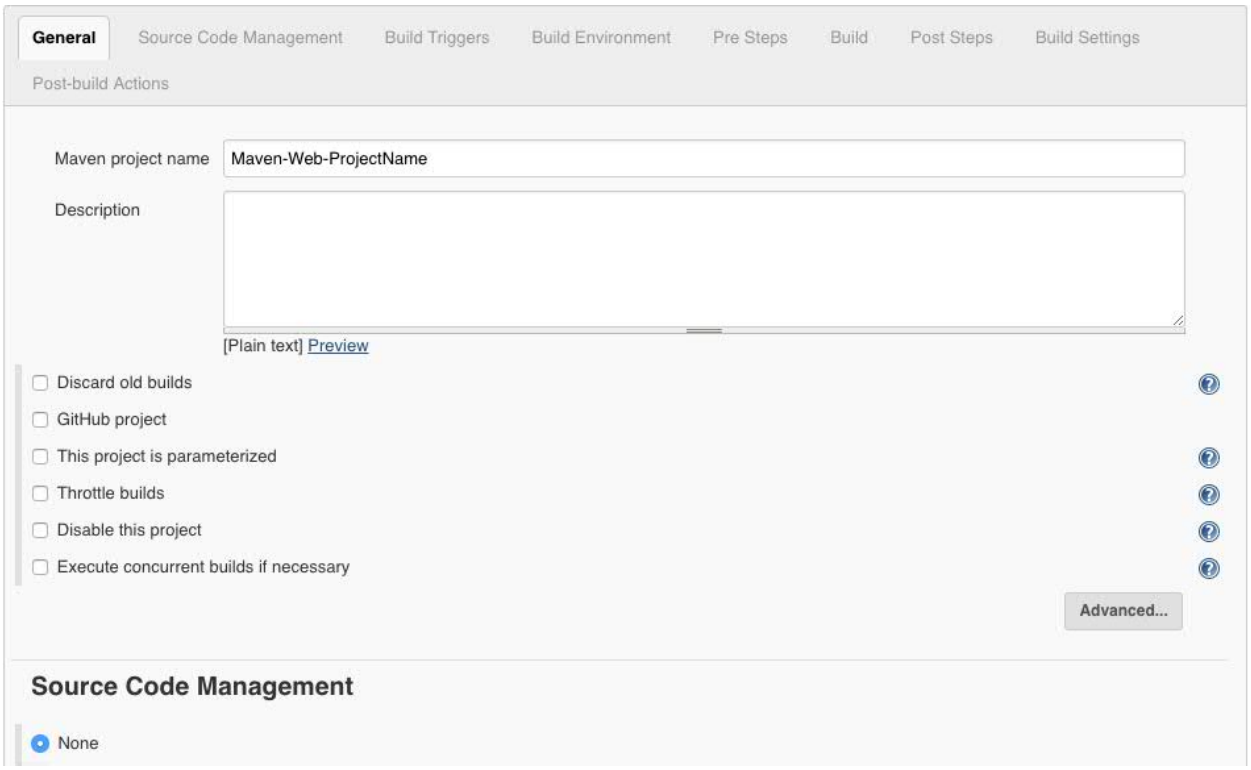
Method 2:

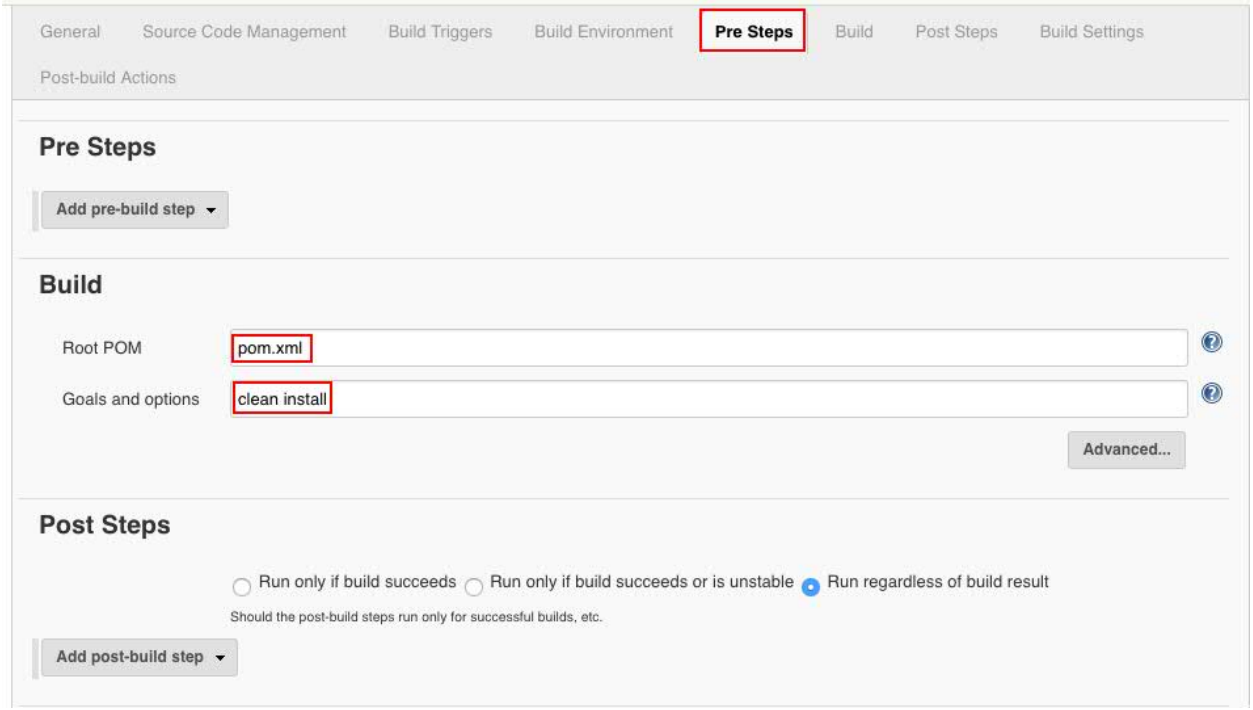
Install the **Maven Integration plugin** and follow the below steps.

Create the New Item as follows.
Provide the item name and select the Maven project and click on OK.



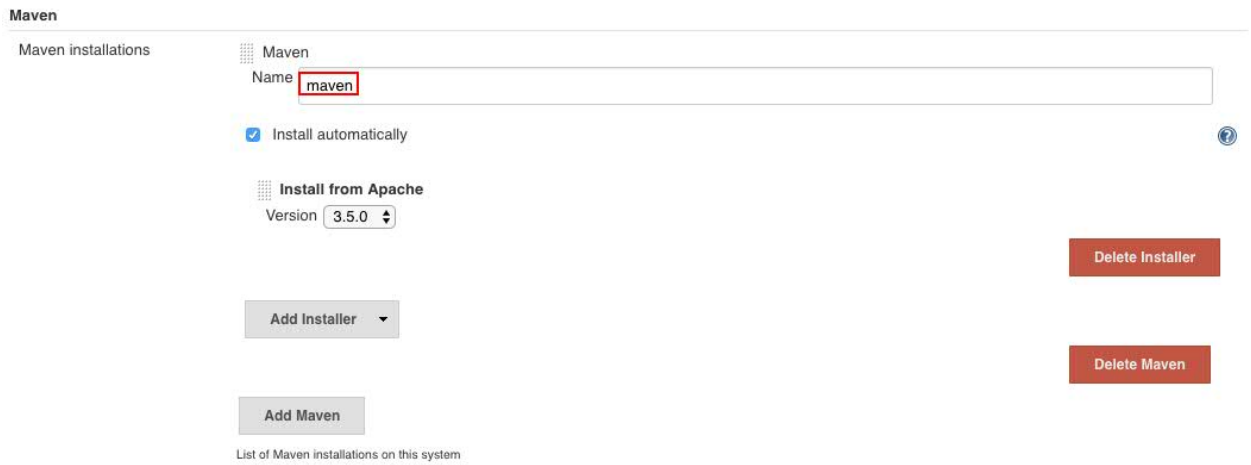
Once you click on OK, you will come to jobs configuration page as follows.





Once you provide all the details click on Save.

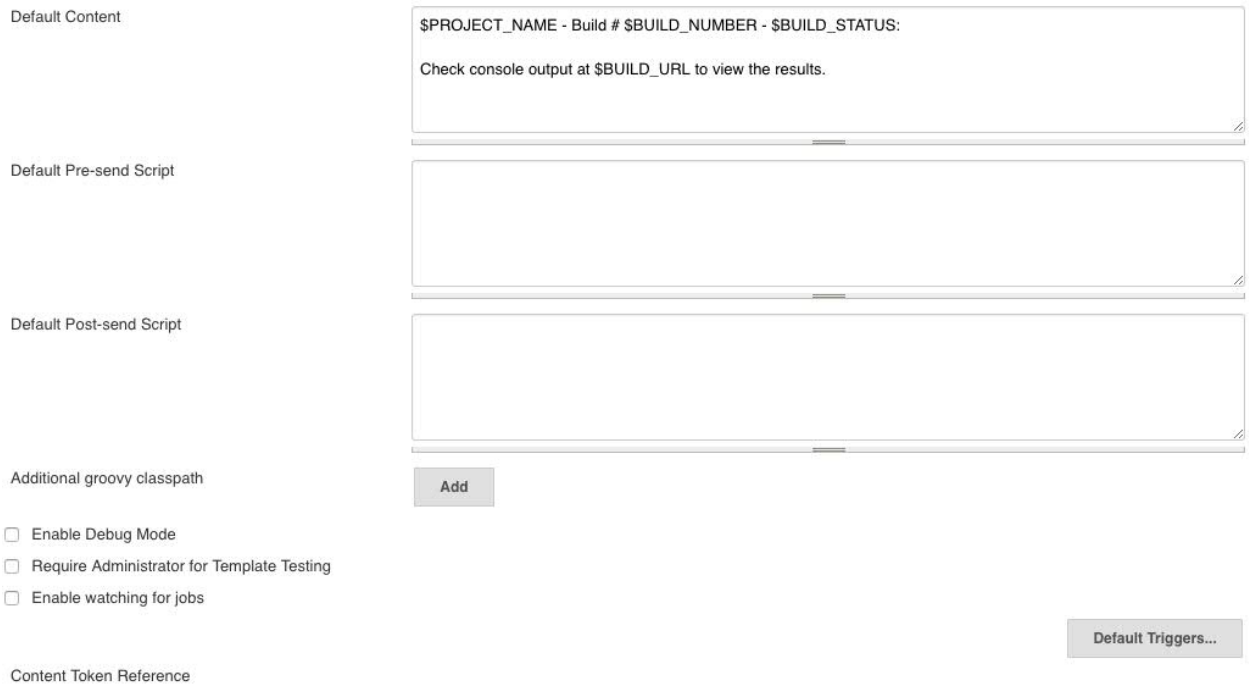
http://localhost:8080/configureTools/



Enable email notification

Step 1) Install Email Extension Plugin as follows.
Manage Jenkins ---> Manage Plugins ---> Install **"Email Extension Plugin "**

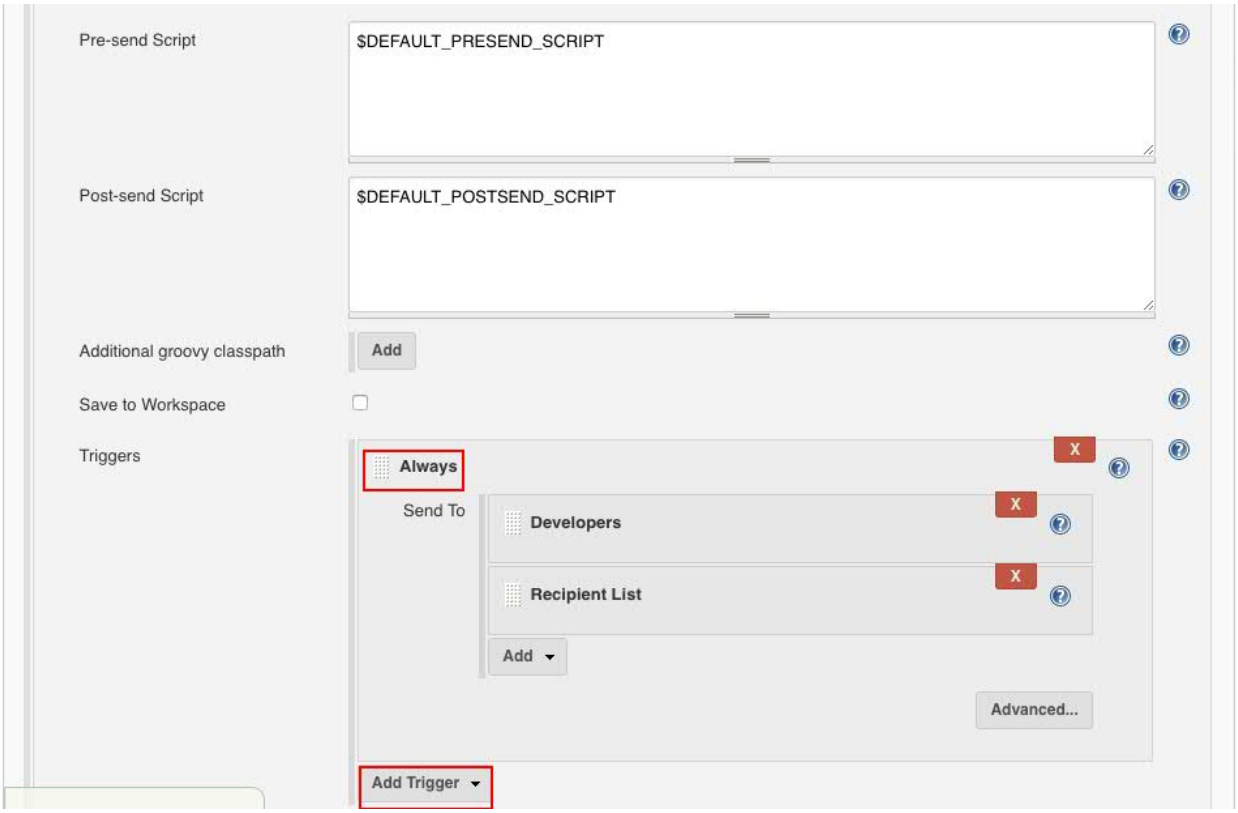
Step 2) Add the smtp server host as follows.
Click on Manage Jenkins ---> Configure System --->



Step 3: In Job configure Editable Email as follows.

Select any Job, which we need to configure Email notification ---> Click on Configure ---> Select the **Post-build Actions** section.

Click on Advanced Settings …
It will expand and will show more settings and click on **Add Trigger** and select the **Always**.



We can enable to attach the build logs while sending mail, as follows.

Output mail is like below.

We can enable to Compress and Attach Build Log to email as follows.

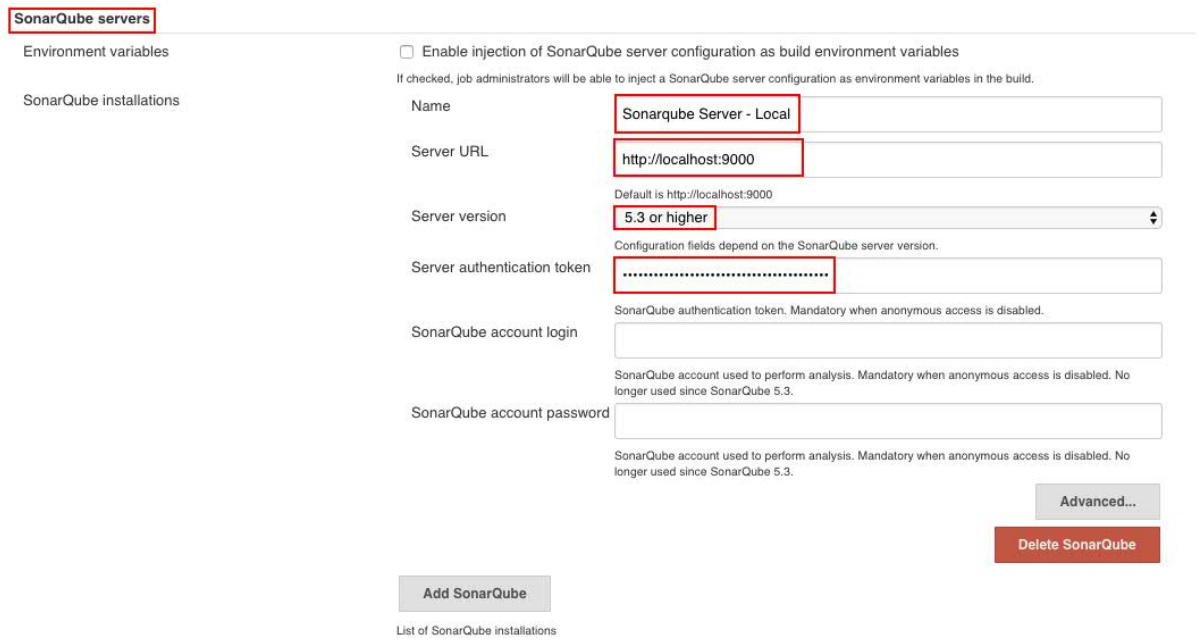
Output mail is like below.

Configure SonarQube with Jenkins

Step 1) Install **"SonarQube Scanner for Jenkins"** for Jenkins plugin.

Step 2) Configure SonarQube as follows.

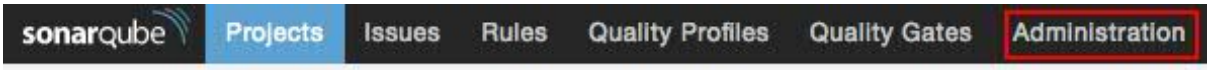
Manage Jenkins ---> Configure System ---> **SonarQube servers**



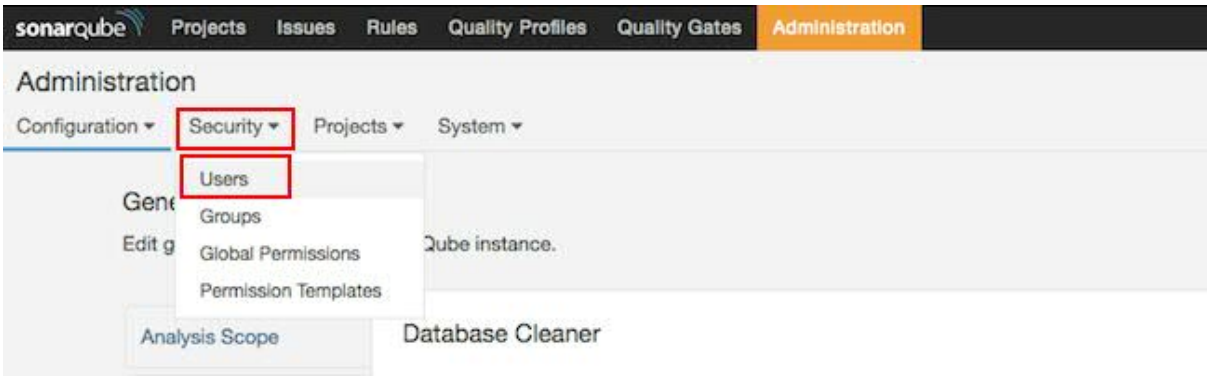
Generate the SonarQube server authentication token

Login into SonarQube with Admin user.
http://localhost:9000/
User: admin
Pwd: admin

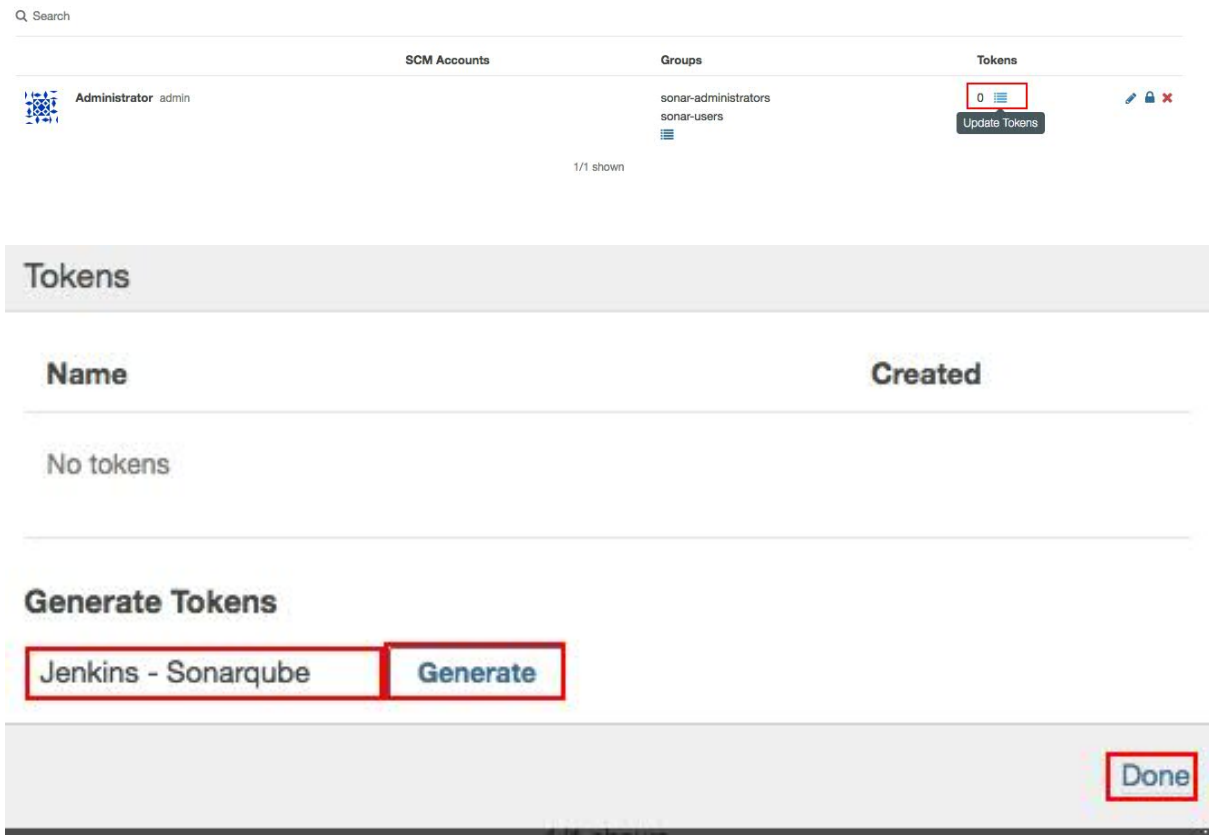
Click on Administration tab.

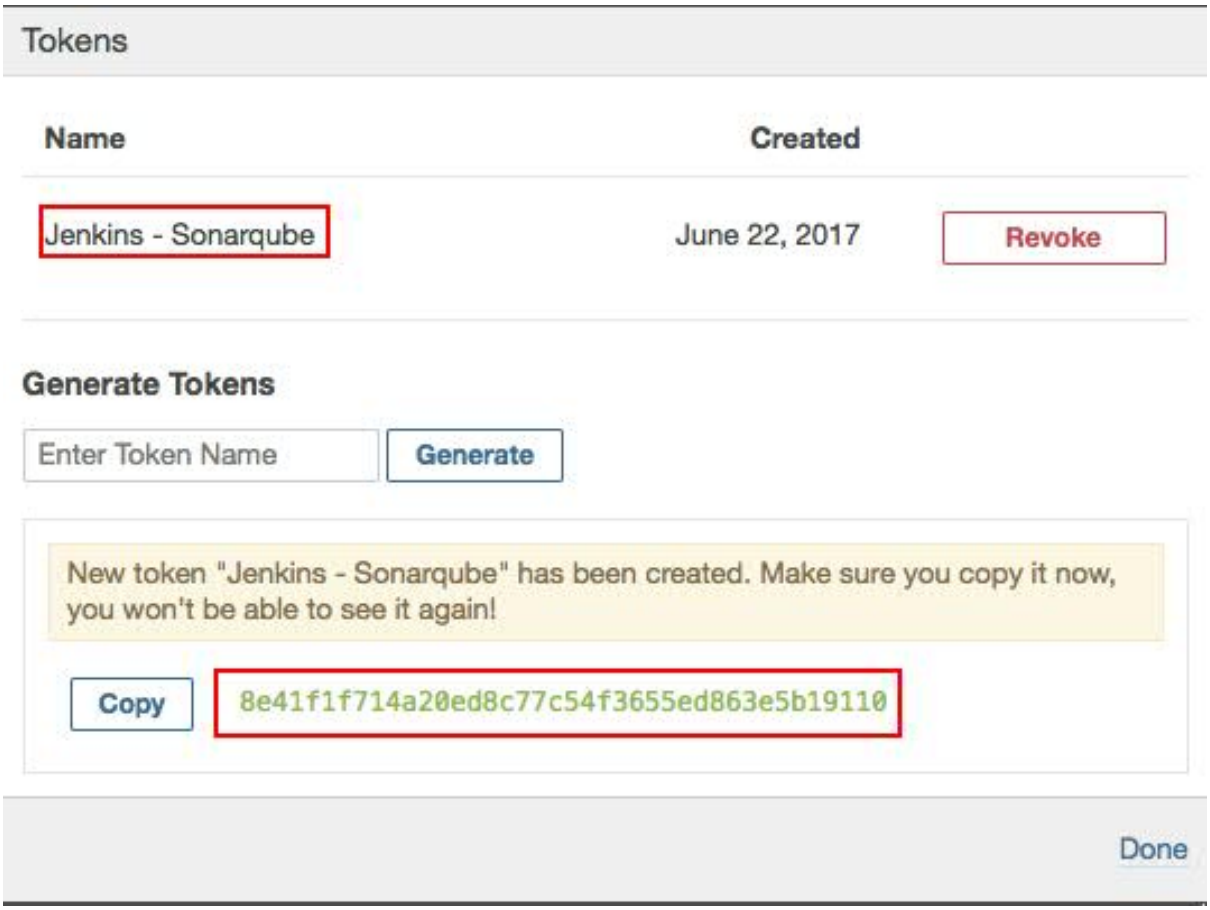


Click on Security dropdown and select the Users

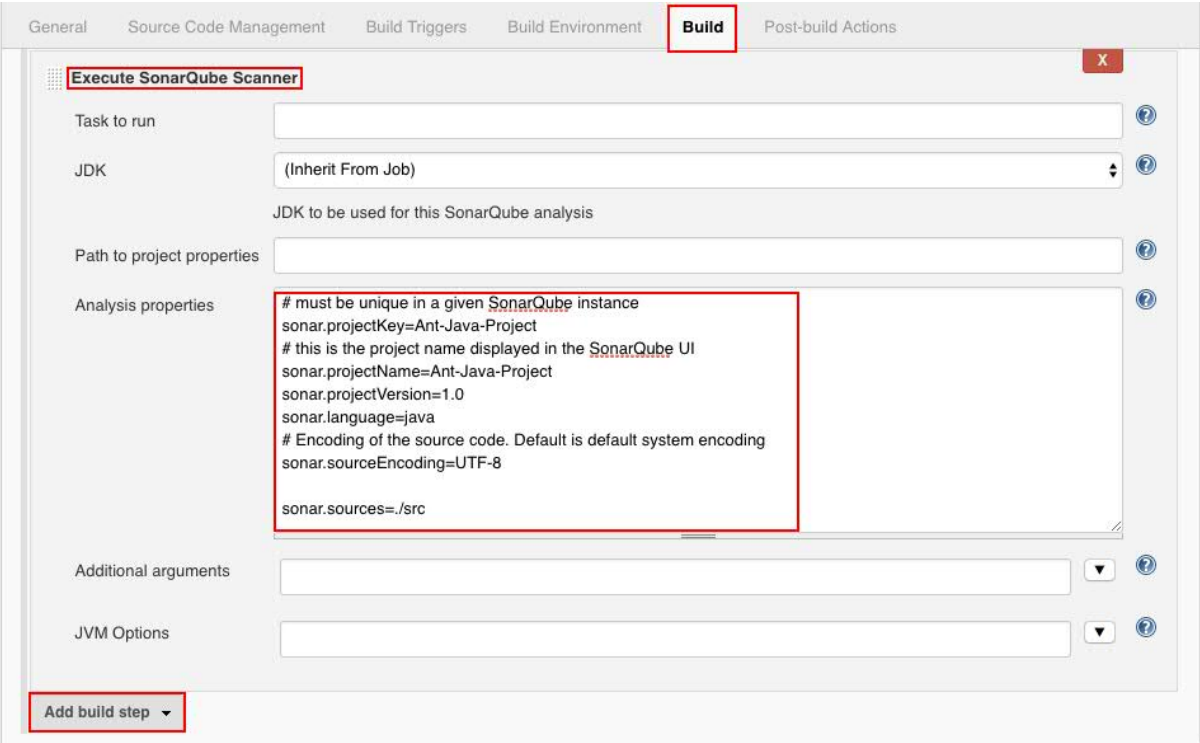


Click on Tokens


Search
SCM Accounts
Groups
Tokens
Administrator admin
sonar-administrators
sonar-users
0
Update Tokens
1/1 shown
Tokens
Name
Created
No tokens
Generate Tokens
Jenkins - Sonarqube
Generate
Done


Tokens
Name
Created
Jenkins - Sonarqube
June 22, 2017
Revoke
Generate Tokens
Enter Token Name
Generate
New token "Jenkins - Sonarqube" has been created. Make sure you copy it now, you won't be able to see it again!
Copy
8e41f1f714a20ed8c77c54f3655ed863e5b19110
Done

Configure SonarQube Scanner for Job/project



To restart Jenkins manually, you can use either of the following URLs:

(jenkins_url)/safeRestart - Allows all running jobs to complete. New jobs will remain in the queue to run after the restart is complete.
Ex: http://localhost:8080/safeRestart
(jenkins_url)/restart - Forces a restart without waiting for builds to complete.
Ex: http://localhost:8080/restart

(OR)

You can install one plug called **SafeRestartPlugin** , once installed it will give one option Jenkins dashboard as follows.

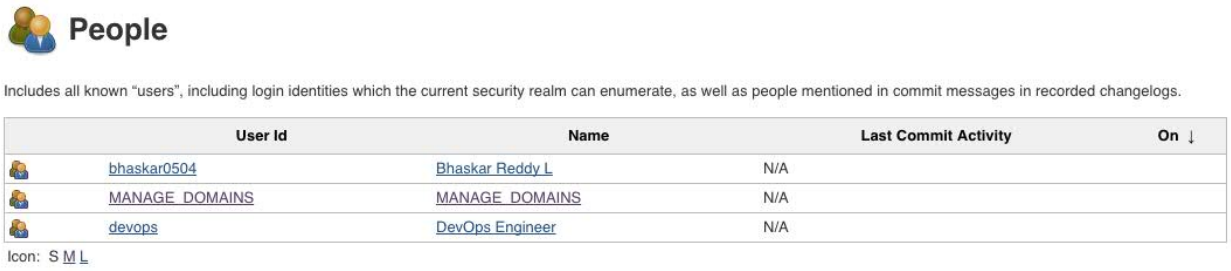
Jenkins - Security

How to create the users in Jenkins?

Click on Manage Jenkins ---> Manage Users ---> Create User ---> Provide the below details

Username:
Password:
Confirm password:
Full name:
E-mail address:
Click on Create User

How to see the list of Users in Jenkins?
Once you logged into Jenkins Dashboard
Go to Left Side Navigation Bar ---> Click on People
You will see list of users available in Jenkins.



How to remove/delete the User in Jenkins?

Click on Manage Jenkins ---> Manage Users ---> click on below Gear icon one circle with cross symbol

It will ask Are you sure about deleting the user from Jenkins? confirmation message Click on ---> Yes

Now User is deleted successfully.

How to change the password for existing users?

Note: Need to document

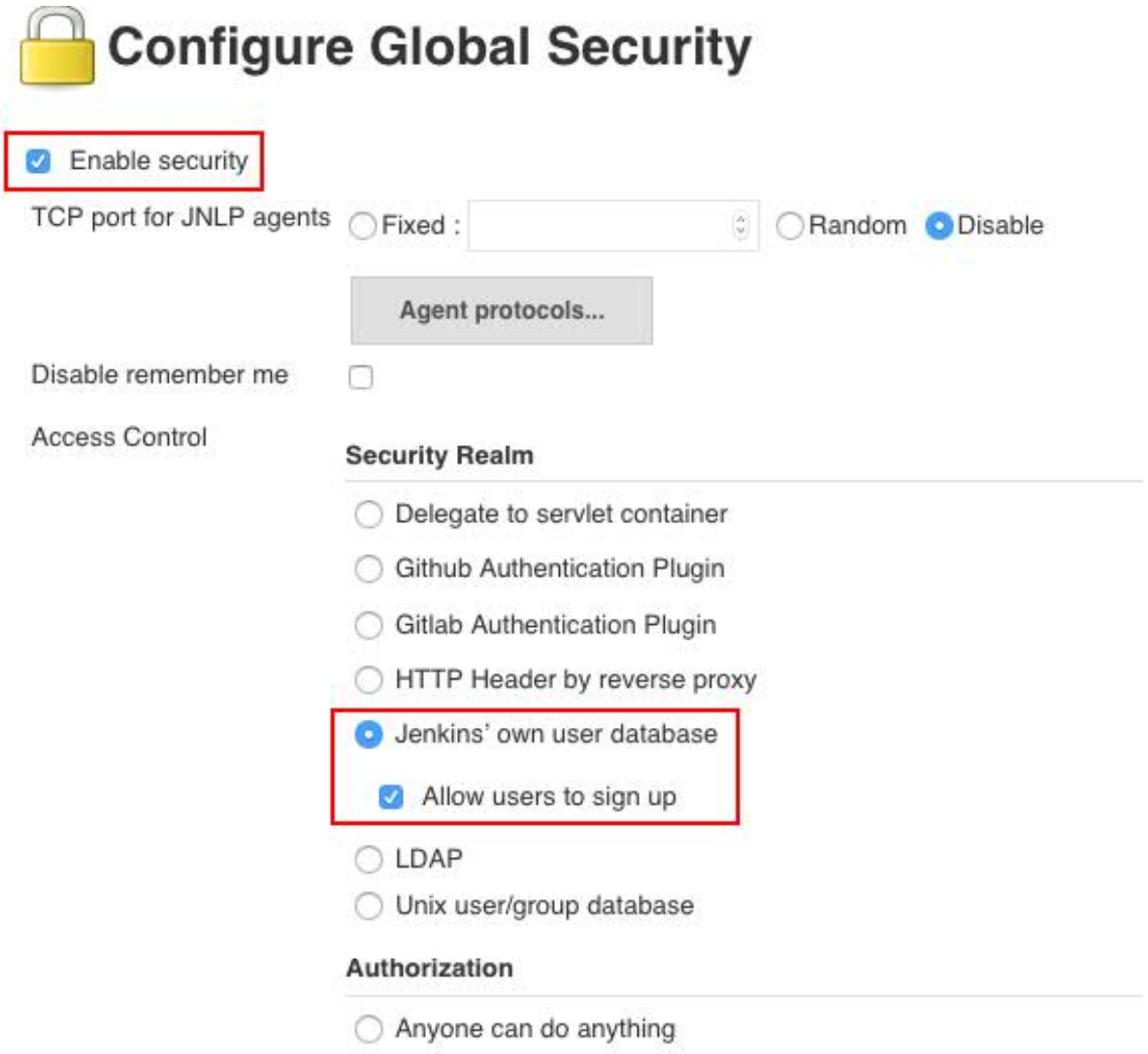
Project-based Matrix Authorization Strategy is an authorization method using which we can define which user or group can do what actions on which job. This gives us a fine-grained control over user/group permissions per project.

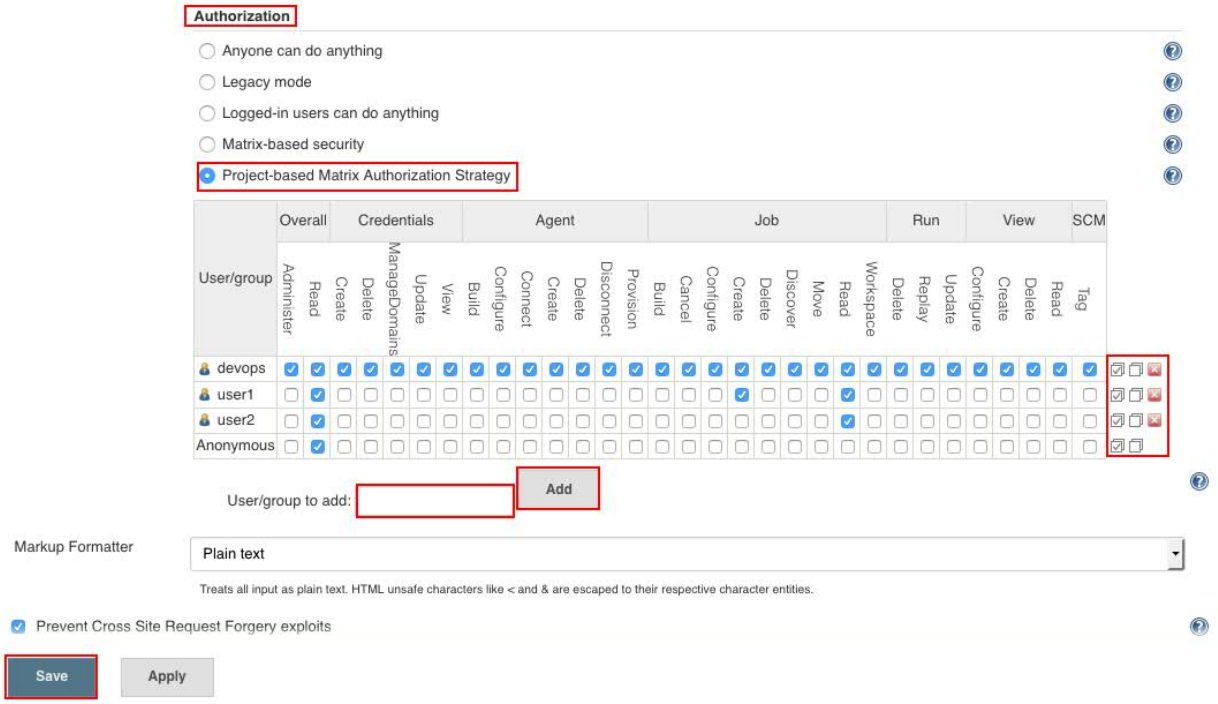
To Enable the Project-based Matrix Authorization Strategy need to configure in Jenkins as follows.

Step 1: Click on Manage Jenkins and choose the 'Configure Global Security' option.

Step 2: Click on Enable Security option.
As an example, let's assume that we want Jenkins to maintain it's own database of users, so in the Security Realm, Select the radio button of 'Jenkins' own user database'.

Step 3: Under Authorization, select "Project-based Matrix Authorization Strategy" and add 2 or 3 users, one administrator (say devops) and a regular user (say user1 and user2).


Configure Global Security
Enable security
TCP port for JNLP agents
Fixed :
Random
Disable
Agent protocols...
Disable remember me
Access Control
Security Realm
Delegate to servlet container
Github Authentication Plugin
Gitlab Authentication Plugin
HTTP Header by reverse proxy
Jenkins' own user database
Allow users to sign up
LDAP
Unix user/group database
Authorization
Anyone can do anything



All the checkboxes present besides users are for setting global permissions. Select all checkboxes against admin user to give admin full permissions.

For user1, we are selecting read permissions under jobs. With this, user1 would now have read permission to view all jobs which we would be creating later on.

We have to provide read permission under “Overall” category to any regular user otherwise the user won’t be able to see anything after login.

All the checkboxes present besides users are for setting global permissions. Select all checkboxes against admin user to give admin full permissions. For user1, we are selecting read permissions under jobs. With this, user1 would now have read permission to view all jobs which we would be creating later on. We have to provide read permission under “Overall” category to any regular user otherwise the user won’t be able to see anything after login.

Finally, you can click on Save button.

Below scenario will applicable in Matrix based security

Error : Access Denied

<<User>> is missing the Overall/Read permission

If you get this error, Pease follow below steps.

Solution:
Click on Manage Jenkins ---> Configure Global Security ---> User/group to add: Enter the user Name and click on Add button and --->
Enable the appropriate feature ---> Click on Save Button.

Jenkins Build Status Icon Colours

 BLUE: Success

 GREY: Disabled, Aborted and Pending

 YELLOW: Unstable

 RED: Failed

Jenkins Job Weather Status Icon Colours

 60-80 of recent builds failed.

 No recent builds failed.

Note: **Need to do more document on these icons.**

Status of the build	Description
	Failed
	Unstable
	Success
	Pending
	Disabled
	Aborted

Figure a: Build status

Job health	Description
	No recent builds failed
	20-40% of recent builds failed
	40-60% of recent builds failed
	60-80% of recent builds failed
	All recent builds failed
	Unknown status

Figure b: Weather reports

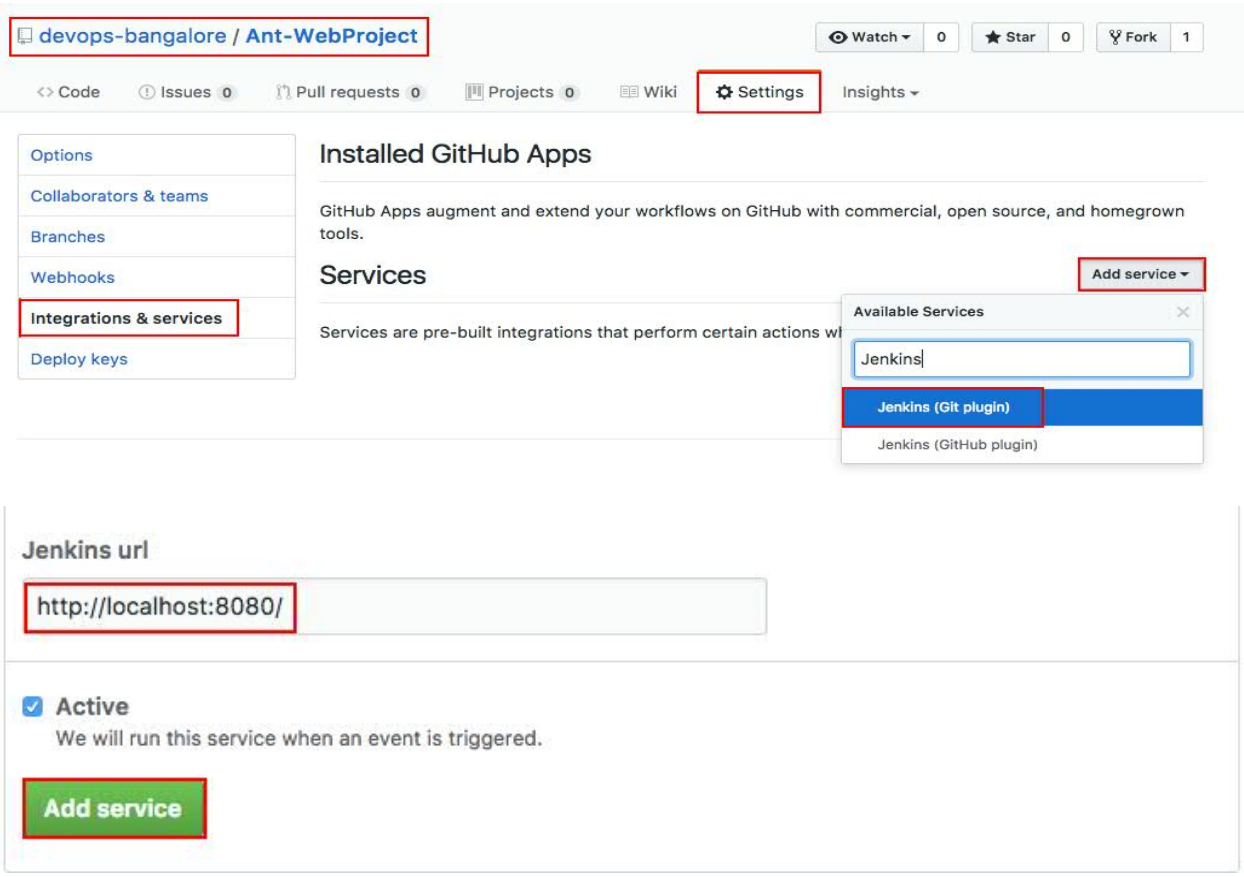
How to enable the hooks in Jenkins?

Step 1: Install the "**Git plugin**" in Jenkins.

Step 2: Select the job which you need to enable hook and click on Configure ---> In **Build Triggers** Section enable the **Poll SCM**
And provide the values as follows.

Step 3: Login into https://github.com

Step 4: Select the project/repository which you need to enable the hooks. ---> Click on Settings ---->Click on Integrations & services ---> Click on Add service and select the Jenkins (Git plugin) from Available Services ---> Provide the Jenkins url (here we are using http://localhost:8080/) and click on **Add service**.


devops-bangalore / Ant-WebProject
Watch 0
Star 0
Fork 1
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Settings
Insights
Options
Collaborators & teams
Branches
Webhooks
Integrations & services
Deploy keys
Installed GitHub Apps
GitHub Apps augment and extend your workflows on GitHub with commercial, open source, and homegrown tools.
Services
Add service
Available Services
Jenkins
Jenkins (Git plugin)
Jenkins (GitHub plugin)
Services are pre-built integrations that perform certain actions wh

Jenkins url
http://localhost:8080/
Active
We will run this service when an event is triggered.
Add service

Jenkins with slack integration

Install the **Slack Notification Plugin** .

Slack:

Go to the Slack App with below url
https://devops-team-bangalore.slack.com/apps

In the search box type Jenkins CI, you will get the Jenkins CI and select that.

Click on Install

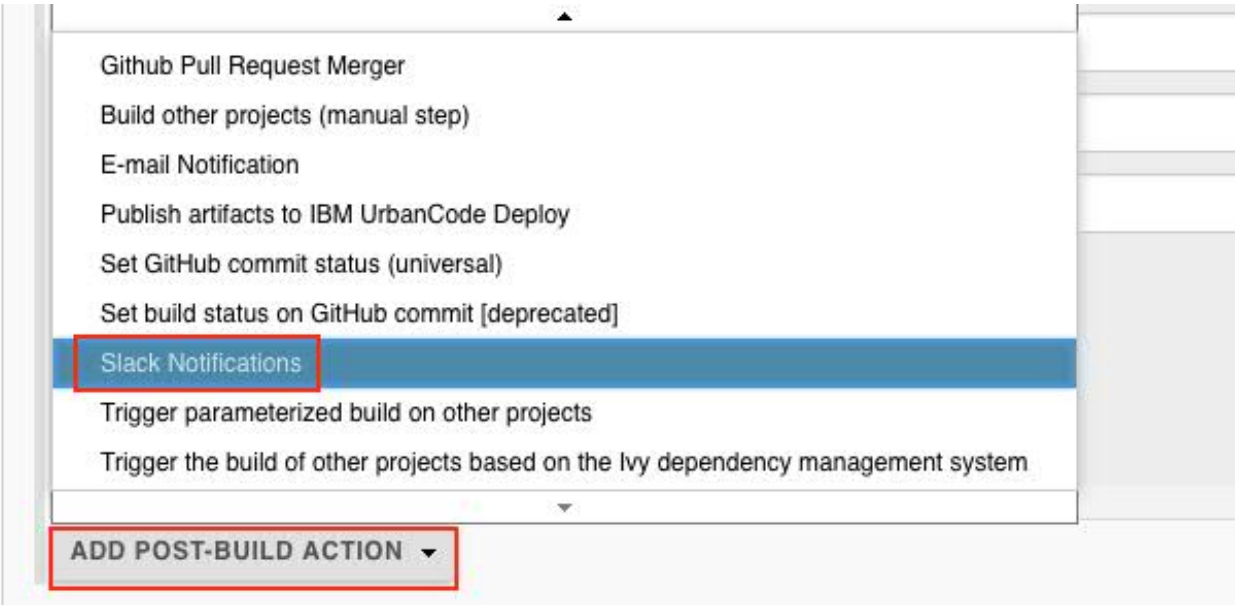
Select the channel in Post to Channel drop down.

Click on Add Jenkins CI Integration

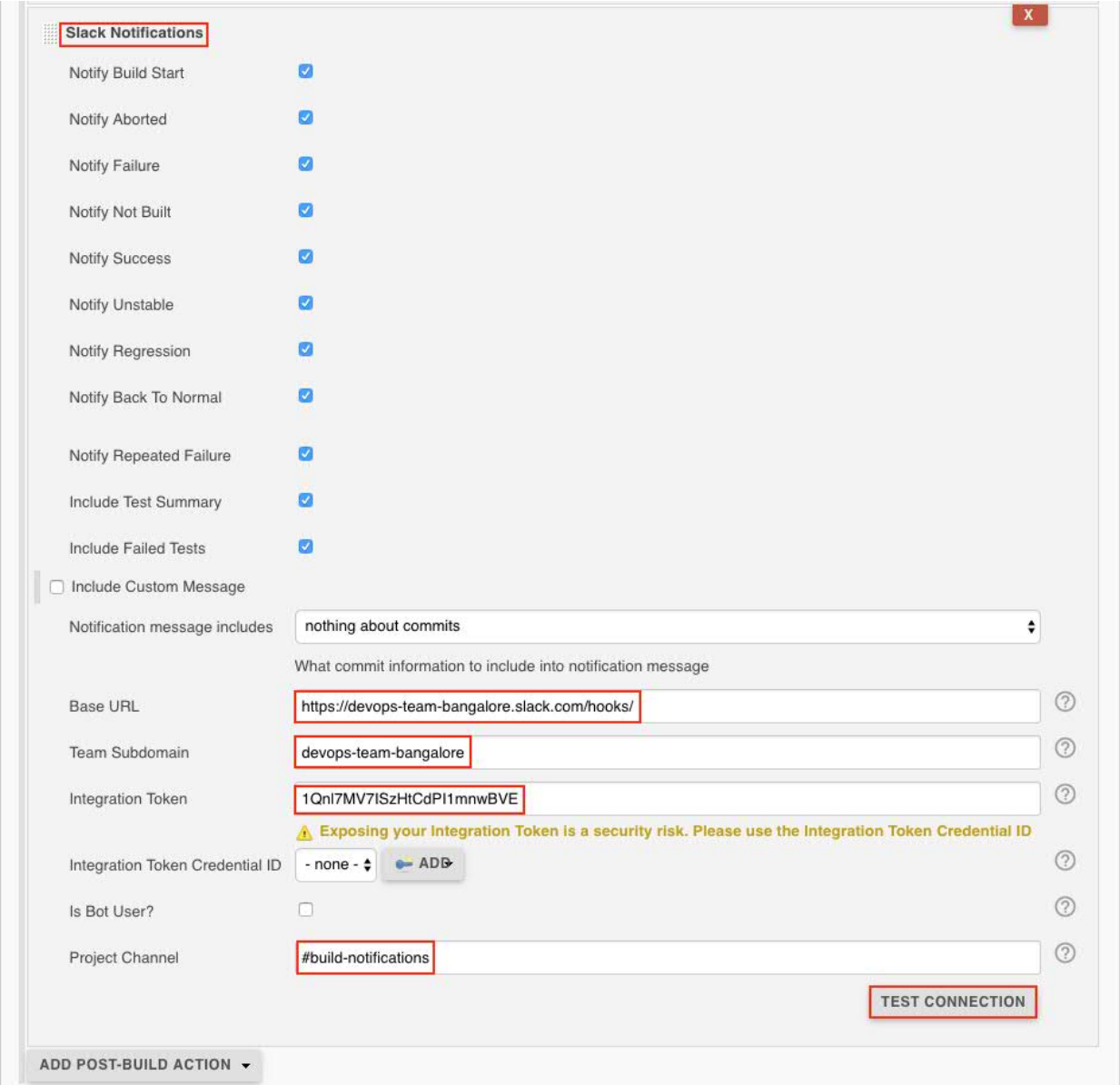
Click on Save Settings.

Jenkins:

Open the job which you want to configure Slack Notifications and click on Configure and in **Post-build actions** tab, click on **ADD POST-BUILD ACTION** and select the **Slack Notifications** as follows.

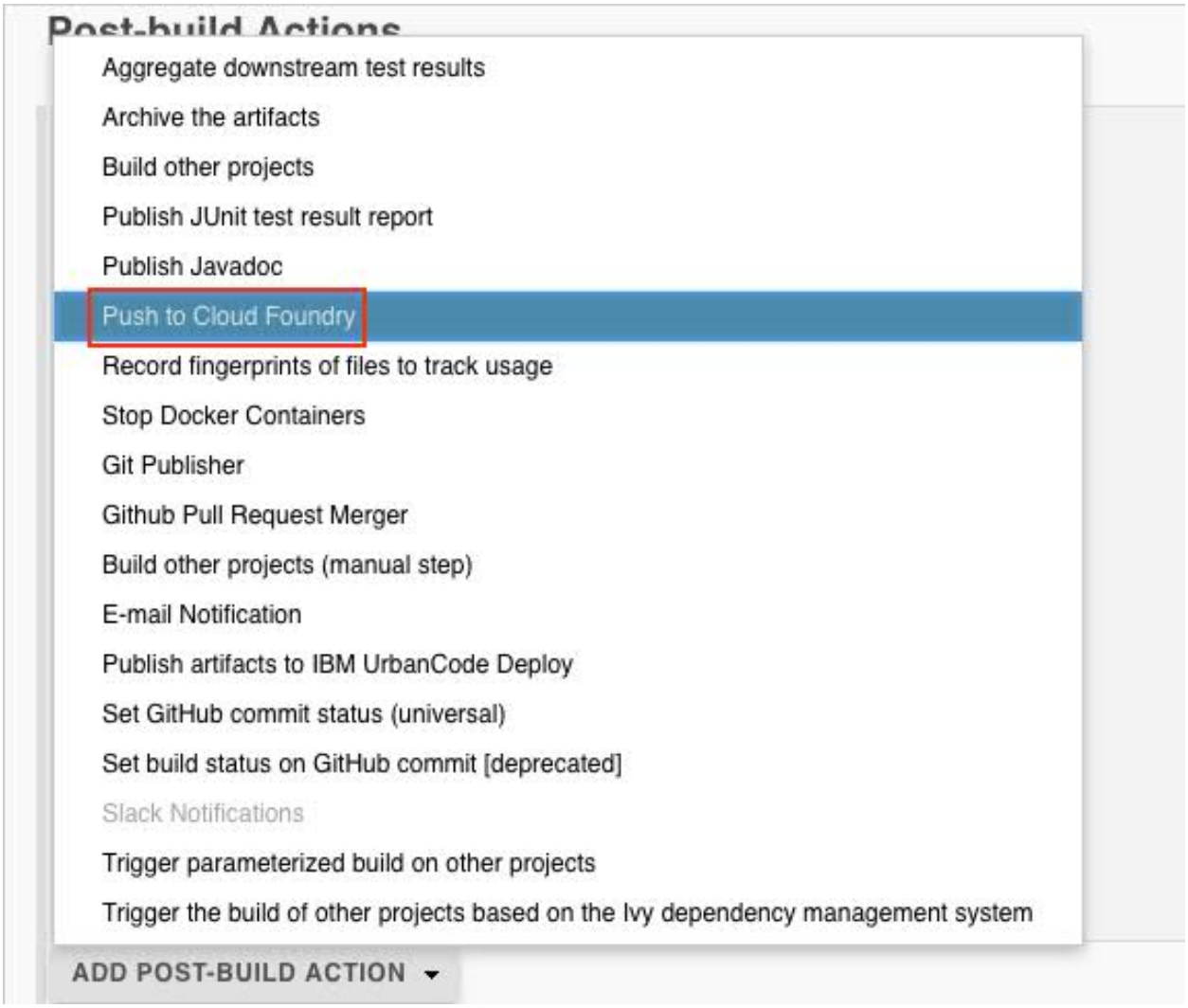


Provide the below details and click on Save.

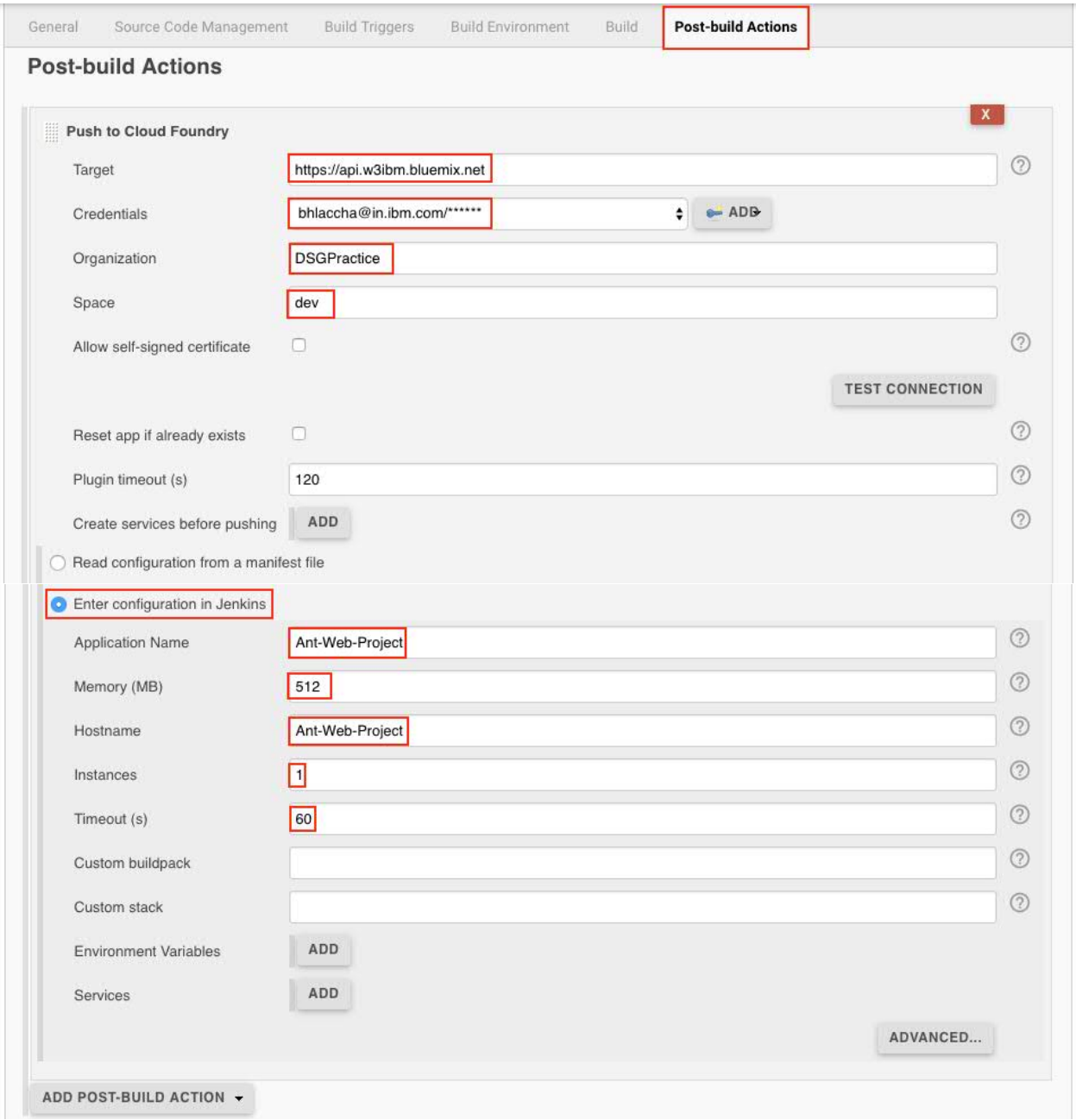


Deploy the application into Bluemix

Install the **"Cloud Foundry Plugin"** plugin.
Open the job which you want to configure deploy, and click on Configure and in **Post-build actions** tab, click on **ADD POST-BUILD ACTION** and select the **Push to Cloud Foundry** as follows.



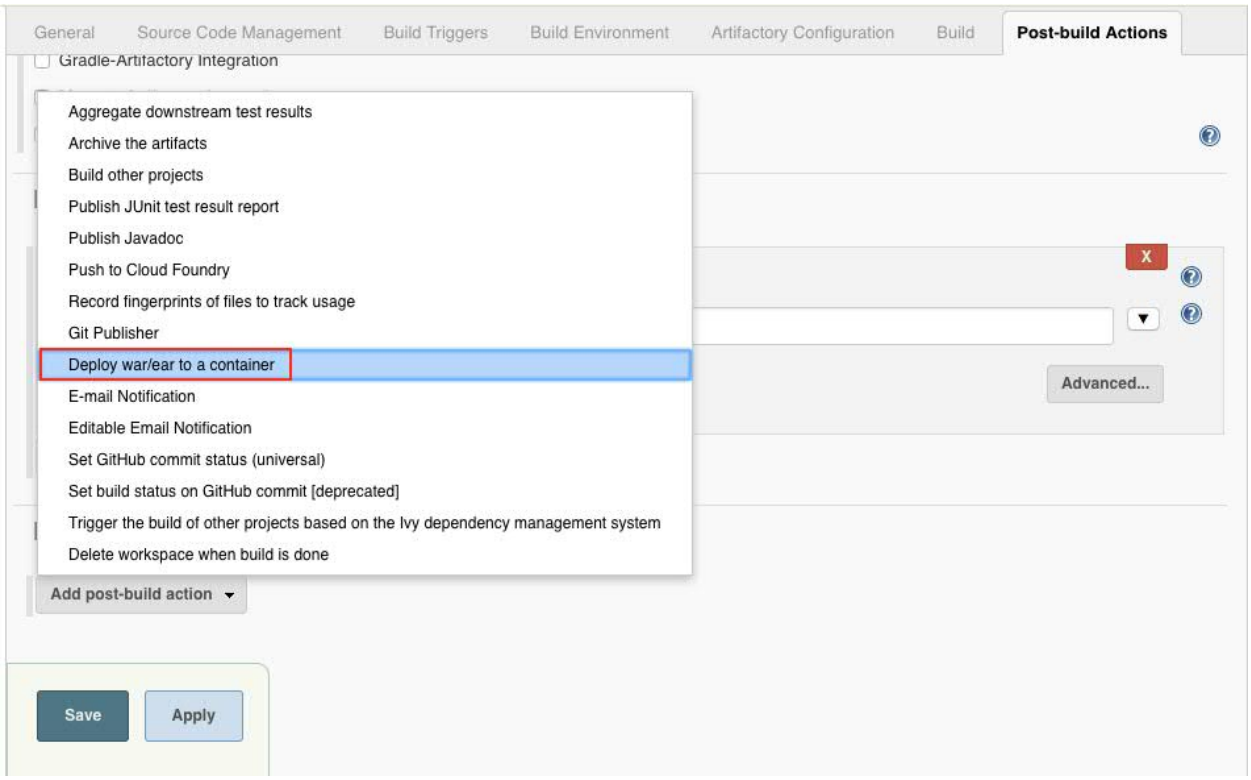
As soon as clicked on Push to Cloud Foundry option you will get the below screen.

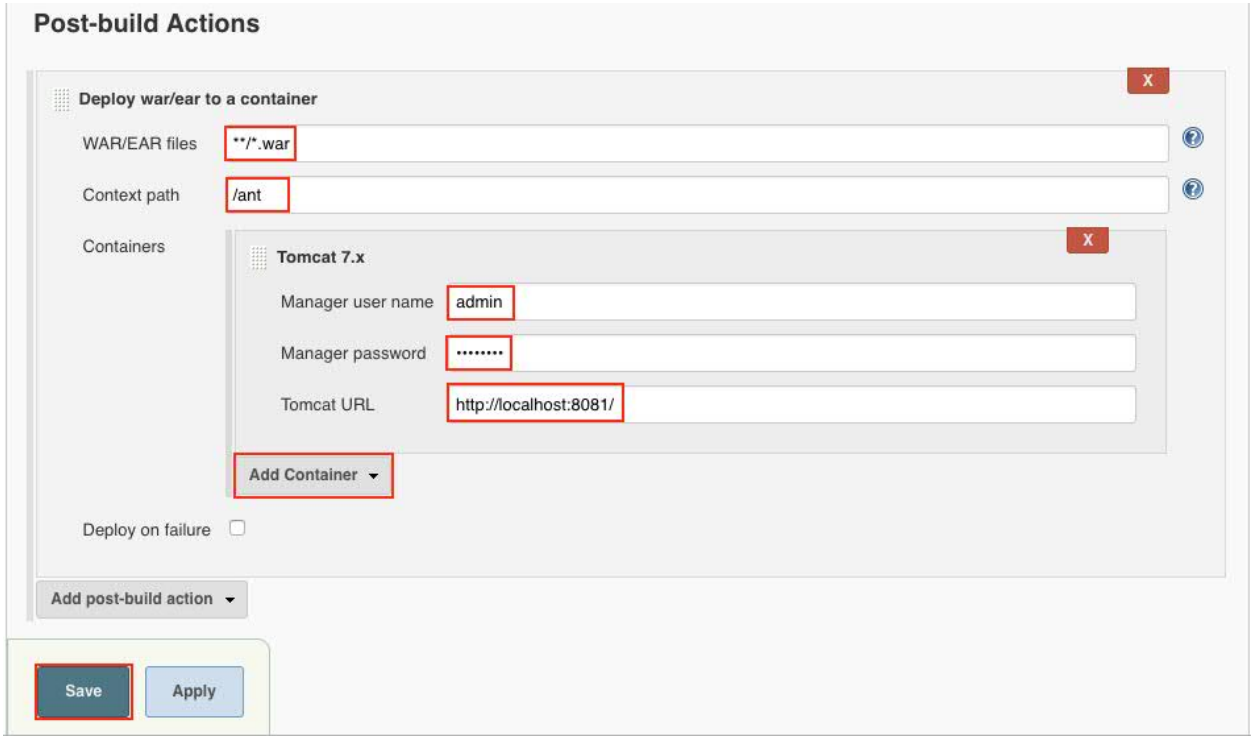


Click on TEST CONNECTION, If connection got succeed and click on Save button.

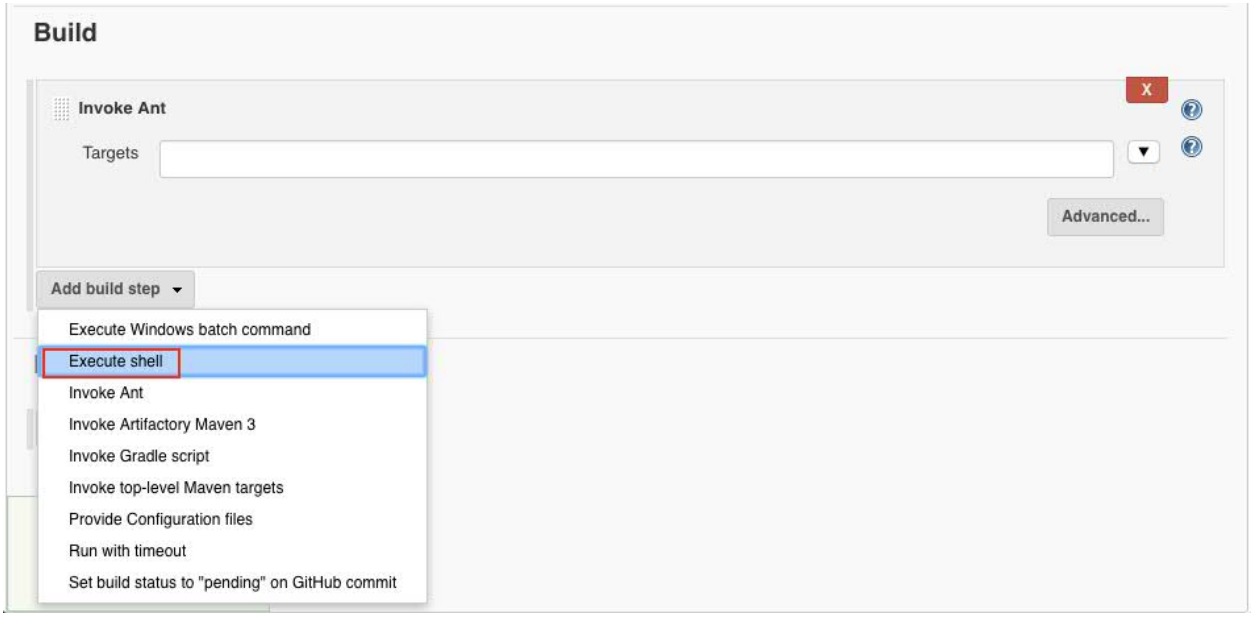
Deploy the application into Bluemix

Install the **"Deploy to container Plugin"** plugin.
Open the job which you want to configure deploy, and click on Configure and in **Post-build actions** tab, click on **ADD POST-BUILD ACTION** and select the **Deploy war/ear to container** as follows.


General
Source Code Management
Build Triggers
Build Environment
Artifactory Configuration
Build
Post-build Actions
Gradle-Artifactory Integration
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Publish Javadoc
Push to Cloud Foundry
Record fingerprints of files to track usage
Git Publisher
Deploy war/ear to a container
E-mail Notification
Editable Email Notification
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Trigger the build of other projects based on the Ivy dependency management system
Delete workspace when build is done
Advanced...
Add post-build action
Save
Apply


Post-build Actions
Deploy war/ear to a container
WAR/EAR files
**/*.war
Context path
/ant
Containers
Tomcat 7.x
Manager user name
admin
Manager password
Tomcat URL
http://localhost:8081/
Add Container
Deploy on failure
Add post-build action
Save
Apply

(OR)



Add the below script in **Execute shell**

```
#!/bin/sh
 echo “Starting to copy the build”
 scp $WORKSPACE/war/SampleAntProject.war /opt/apache-tomcat-7.0.78/webapps
 echo “Copied the build to tomcat”
```

Note: If we want to deploy in Tomcat, which is deployed any remote machine, use below line og code.

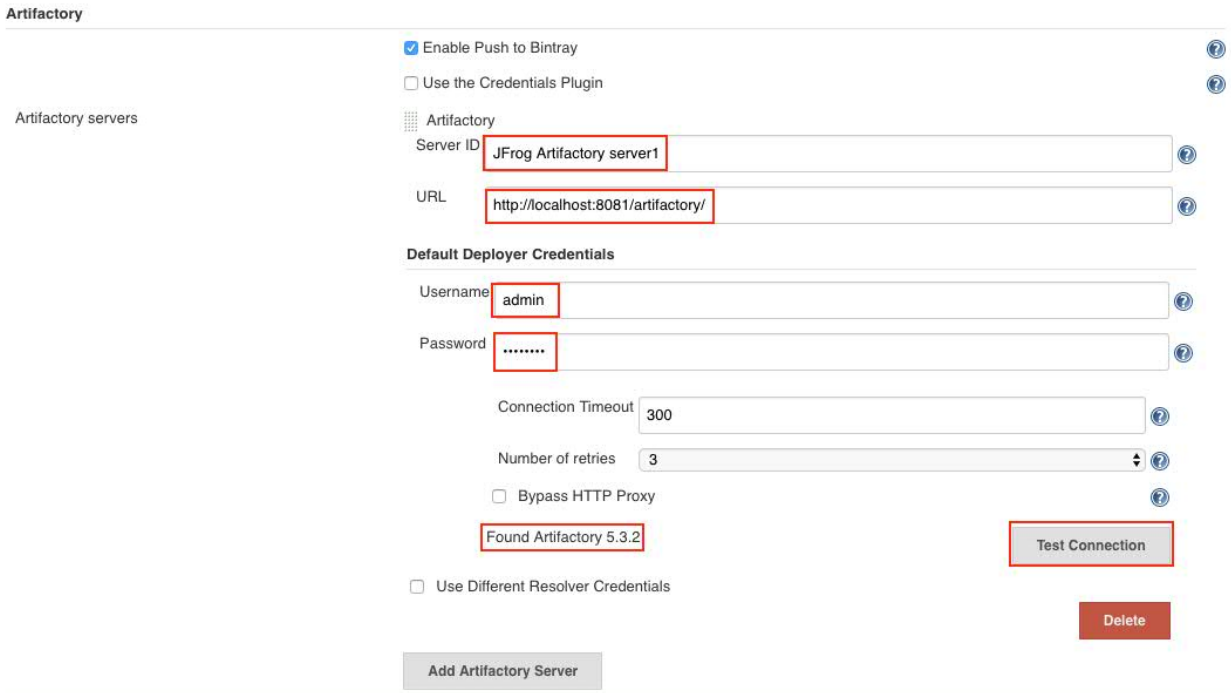
$WORKSPACE/war/SampleAntProject.war <<User Name>>@<<ServerIP>>:/opt/apache-tomcat-7.0.78/webapps

---

Integrate JFrog Artifactory with Jenkins

Install **"Artifactory Plugin"** plugin.

Got to the Manage Jenkins ---> Configure System --->
In the **Artifactory** section fill the below details and click on Save.



Note: Once you entered all the details click on **TEST CONNECTION**. IF connection is succeed you will see the message like **Found Artifactory <<Version>>**.

Jenkins – Metrics and Trends

There are various plugins which are available in Jenkins to showcase metrics for builds which are carried out over a period of time. These metrics are useful to understand your builds and how frequently they fail/pass over time. As an example, let's look at the '**Build History Metrics plugin'**. This plugin calculates the following metrics for all of the builds once installed

Mean Time To Failure (MTTF)
Mean Time To Recovery (MTTR)
Standard Deviation of Build Times

Enable LDAP security to Jenkins

---

http://www.scmgalaxy.com/tutorials/complete-guide-to-use-jenkins-cli-command-line

Jenkins CLI

Jenkins has a built-in command line interface (CLI) that allows users and administrators to access Jenkins from a script or shell environment. This can be convenient for scripting of routine tasks, bulk updates, troubleshooting, and more.

Advantages of Jenkins CLI:

- Easier
- Faster
- Memory management
- Automation tasks.

Pre-Requisites

a) Jenkins server is running

b) Enable security as follows.

Go to jenkins dashboard in Home page (e.g http://localhost:8080/) -> Manage Jenkins -> Configure Global Security -> Click on “Enable security” checkbox

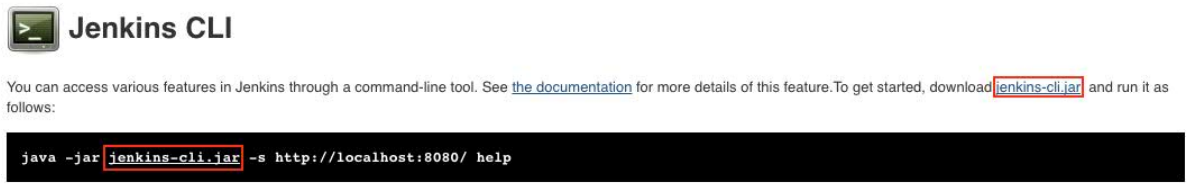
You can also configure “Access Control” and “Authorization” option in Global Security page.

Download the Jenkins CLI jar file as follows.

Method 1

Open the below url

http://localhost:8080/cli/



Click on Jenkins-cli.jar.

Method 2

Click on below url, it will automatically download the jar file.

http://<<Jenkins Server URL>>/jnlpJars/jenkins-cli.jar

Example: http://localhost:8080/jnlpJars/jenkins-cli.jar

Here
Copy into any folder as follows

#cp jenkins-cli.jar /opt/jenkins/

Go to the directory where Jenkins-cli.jar is there and run the below command to get the help.

Login Jenkins using username and Password

java -jar jenkins-cli.jar -s http://localhost:8080/ help --username devops --password passw0rd

Get the Version of Jenkins
#java -jar jenkins-cli.jar -s http://localhost:8080/ version --username devops --password passw0rd

Get all the jobs of Jenkins

#java -jar jenkins-cli.jar -s http://localhost:8080/ list-jobs --username devops --password passw0rd

Delete the Job

#java -jar jenkins-cli.jar -s http://localhost:8080/ delete-job ant-java-job-dev --username devops --password passw0rd

#java -jar jenkins-cli.jar -s http://localhost:8080/ disable-job ant-web-job-dev --username devops --password passw0rd

#java -jar jenkins-cli.jar -s http://localhost:8080/ enable-job ant-web-job-dev --username devops --password passw0rd

While executing above command if you see Enter passphrase, follow the below configuration.
(Manage Jenkins ---> Configure Global Security ---> enable the Enable Security ---> Apply and Save.)

Manage Jenkins ---> Configure System --->SSH Public Keys (Enter here any value, same value u can use in CLI)

UrbanCode Deploy with Jenkins

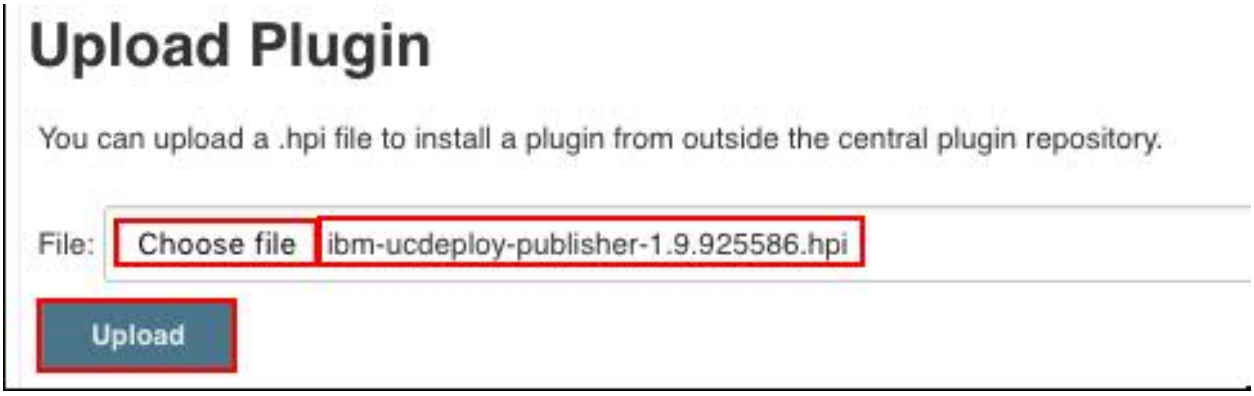
Download the UCD Plugin from below url.

https://developer.ibm.com/urbancode/plugin/jenkins/ (ibm-ucdeploy-publisher-1.9.925586.hpi) ---> Old version URL

https://developer.ibm.com/urbancode/plugin/jenkins-2-0/ (ibm-ucdeploy-build-steps-2.6.929921.hpi) ---> New version URL

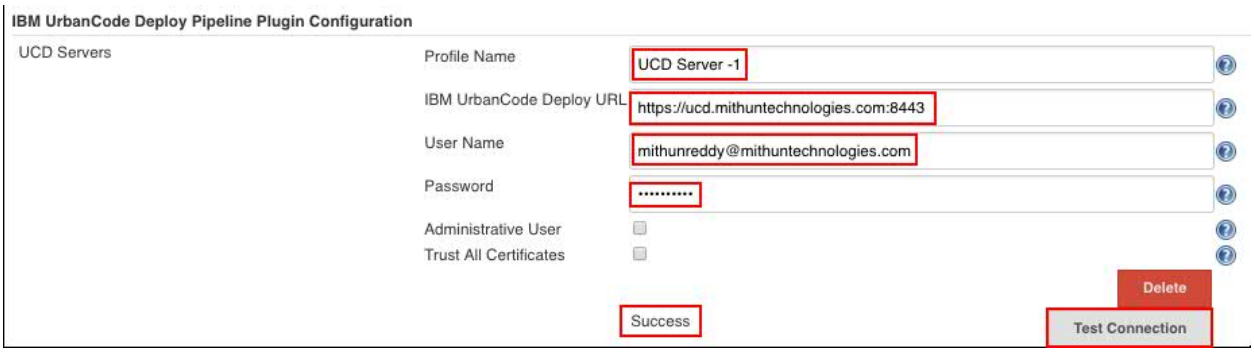
Once you download the UCD plugin, install the that plugin in Jenkins as follows.

Manage Jenkins ---> **Manage Plugins** ---> Got to the "**Advanced**" tab ---> Go to the "**Upload Plugin**" section and click on **Choose file** and select the UCD plugin (ibm-ucdeploy-publisher-1.9.925586.hpi OR ibm-ucdeploy-build-steps-2.6.929921.hpi) ---> Click on "**Upload**"



Integrate the UrbanCode Deploy Server as follows.

Manage Jenkins ---> **Configure System** --->Got the "**IBM UrbanCode Deploy Pipeline Plugin Configuration"** section and click on **Add** button and provide the UCD details as follows, and click on **Save** button.



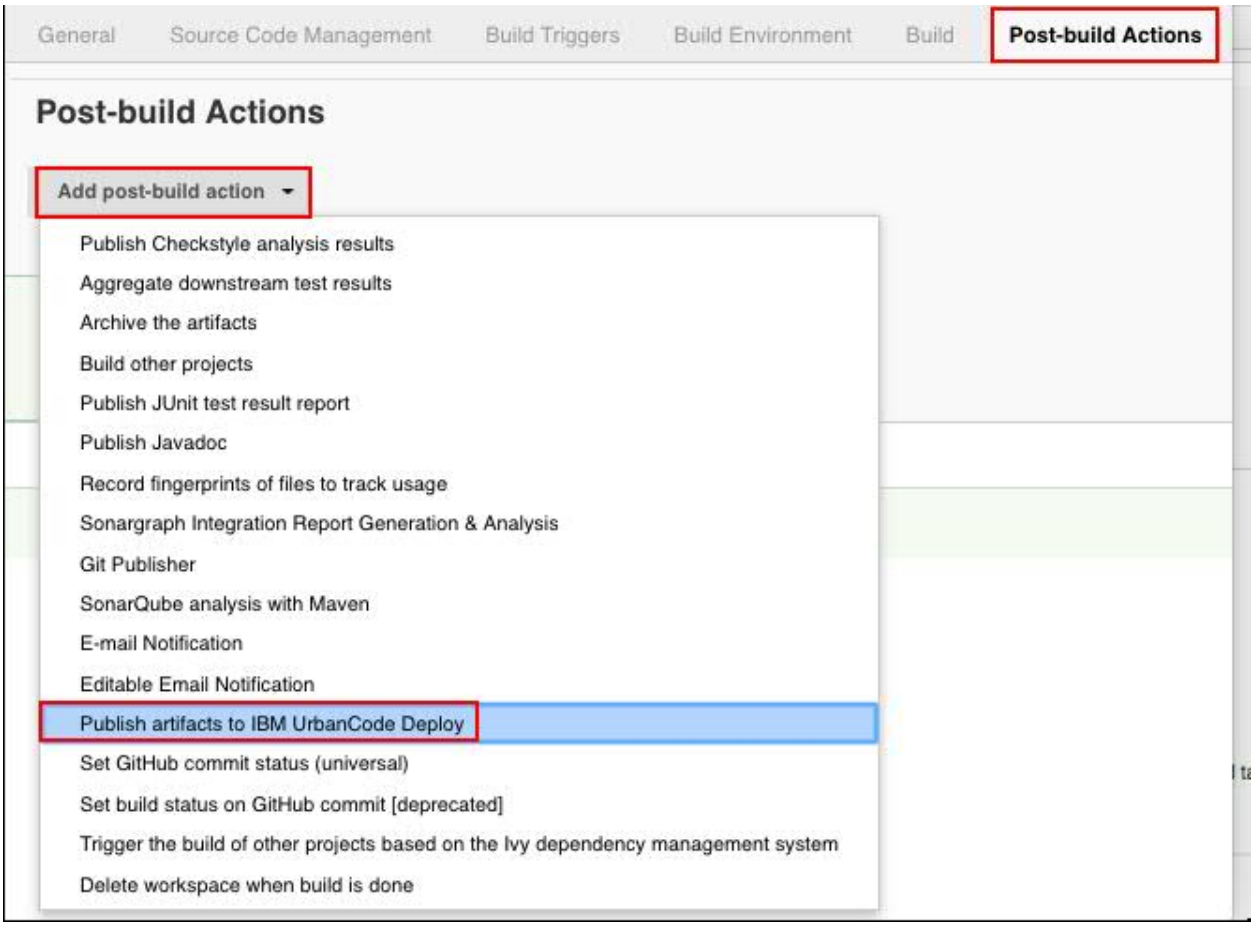
If you provide wrong UCD server, you will get the below error message.

If you provide the wrong Credentials, you will get the below error message.

Once you have integrated UCD with Jenkins, you can enable UCD for each job as follows.
Go to the **Post-build Actions** section and select the **Publish artifacts to IBM UrbanCode Deploy**.

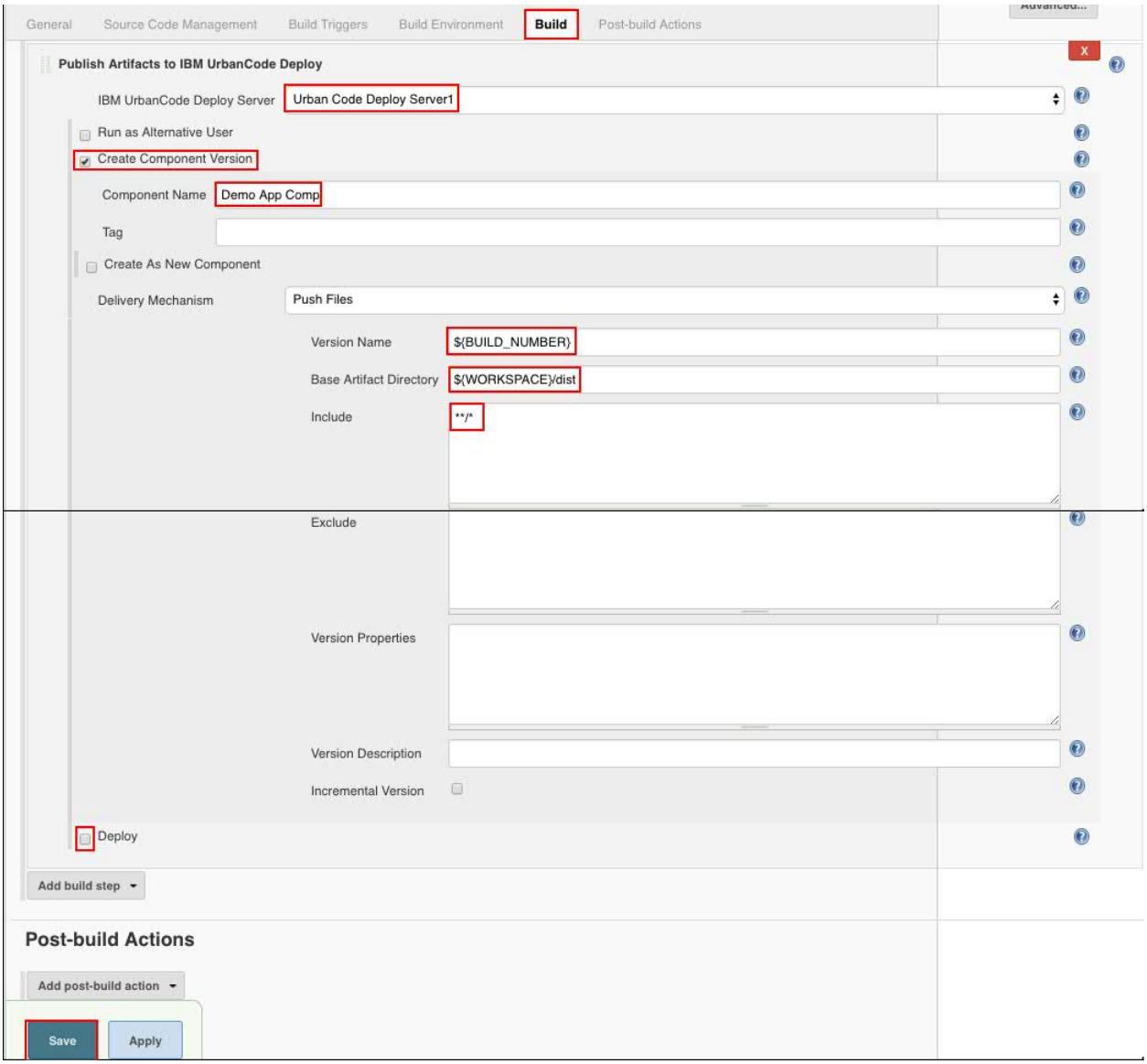
Note: If you have install **ibm-ucdeploy-publisher-1.9.925586.hpi** Plugin, you will see the **Publish artifacts to IBM UrbanCode Deploy** option in **Post-build Actions** section.

If you have install ibm-ucdeploy-build-steps-2.6.929921.hpi Plugin, you will see the **Publish artifacts to IBM UrbanCode Deploy** option in Build section.


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Post-build Actions
Add post-build action
Publish Checkstyle analysis results
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Publish Javadoc
Record fingerprints of files to track usage
Sonargraph Integration Report Generation & Analysis
Git Publisher
SonarQube analysis with Maven
E-mail Notification
Editable Email Notification
Publish artifacts to IBM UrbanCode Deploy
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Trigger the build of other projects based on the Ivy dependency management system
Delete workspace when build is done

As soon as you select the **Publish artifacts to IBM UrbanCode Deploy** option you will get the below screen.

Provide the UCD Details.



Jenkins Pipeline Project

Required Plugins

1) Pipeline Maven Integration Plugin
2) JUnit Attachments Plugin'
3) Task Scanner Plugin

In Jenkins Pipeline project, we will use one file called Jenkinsfile, in this file we will write groovy code to build process.
We will write Jenkinsfile in 2 ways.
1) Declarative way
2) Scripted way.

1) Declarative Pipeline Syntax

Jenkinsfile

```
pipeline {
   agent any

   stages {
      stage('Compile Stage') {
         steps {
            echo 'Comple Stage starts...'
                                withMaven(maven : 'maven_3_5_2'){
                                 sh 'mvn clean compile'
                                echo 'Comple Stage ends...'
                                }
         }
      }
      stage('Testing Stage') {
         steps {
            echo 'Testing Stage starts...'
                                withMaven(maven : 'maven_3_5_2'){
                                 sh 'mvn test'
                                echo 'Testing Stage ends...'
                                }
         }
      }
      stage('Install Stage') {
         steps {
            echo 'Install Stage starts...'
                                withMaven(maven : 'maven_3_5_2'){
                                 sh 'mvn install'
                                echo 'Install Stage ends...'
                                }
         }
      }
          stage('Deploy Stage') {
         steps {
            echo 'Install Stage starts...'
                                withMaven(maven : 'maven_3_5_2'){
                                 sh 'mvn deploy'
                                echo 'Deploy Stage ends...'
                                }
         }
      }
      stage('SonarQube Scanner Stage') {
         steps {
            echo 'Deploymnet Stage starts...'
                                 sh 'mvn sonar:sonar'
                                echo 'Deploymnet Stage ends...'
         }
      }
   }
```

}

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.bt</groupId>
    <artifactId>maven-web-project</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>maven-web-project Maven Webapp</name>
    <url>http://maven.apache.org</url>

    <properties>
        <!-- Spring test requires 1.7 onwards. don't down grade version -->
        <!-- <jdk.version>1.6</jdk.version> -->
        <spring.version>4.1.4.RELEASE</spring.version>
        <junit.version>4.11</junit.version>
        <log4j.version>1.2.17</log4j.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20160212</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring.version}</version>
```

```
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<dependency>
			<groupId>javax</groupId>
			<artifactId>javaee-web-api</artifactId>
			<version>7.0</version>
			<scope>provided</scope>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
		</dependency>
		<!-- http://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-jdbc</artifactId>
			<version>${spring.version}</version>
		</dependency>

	</dependencies>

	<repositories>
	  <repository>
	    <id>myMavenRepo.read</id>
	    <url>https://mymavenrepo.com/repo/Kc9khPUGgDzV9gruoyls/</url>
	  </repository>
	</repositories>

	<distributionManagement>
	  <repository>
	    <id>myMavenRepo.write</id>
	    <url>https://mymavenrepo.com/repo/xtzDXsryp1OKYd3hZ4iE/</url>
	  </repository>
	</distributionManagement>


	<build>
		<plugins>
			<plugin>
				<groupId>org.sonarsource.scanner.maven</groupId>
				<artifactId>sonar-maven-plugin</artifactId>
				<!--<version>3.3.0.603</version>-->
			</plugin>
		</plugins>
	</build>


	<profiles>
```

```
            <profile>
                <id>sonar</id>
                <activation>
                    <activeByDefault>true</activeByDefault>
                </activation>
                <properties>
                    <sonar.host.url>http://localhost:9001/</sonar.host.url>
                </properties>
            </profile>
        </profiles>
    </project>
```

2) Scripted Pipeline Syntax

```
#!groovy

node {
   currentBuild.result = "SUCCESS"

   try {

      stage('Checkout'){

         checkout scm
      }

      stage('Compiling'){

         sh 'mvn deploy'
      }

     stage('Sonar') {
                //add stage sonar
                sh 'mvn sonar:sonar'
            }
      stage('mail'){

        mail body: 'project build successful',
                 from: 'devopstrainingblr@gmail.com',
                 replyTo: 'mithunreddytechnologies@gmail.com',
                 subject: 'project build successful',
                 to: 'mithunreddytechnologies@gmail.com'
      }

   }
   catch (err) {

       currentBuild.result = "FAILURE"

          mail body: "project build error is here: ${env.BUILD_URL}" ,
          from: 'devopstrainingblr@gmail.com',
          replyTo: 'mithunreddytechnologies@gmail.com',
```

```
            subject: 'project build failed',
            to: 'mithunreddytechnologies@gmail.com'

        throw err
    }
}
```

Jenkins Multi Branch Pipeline Project

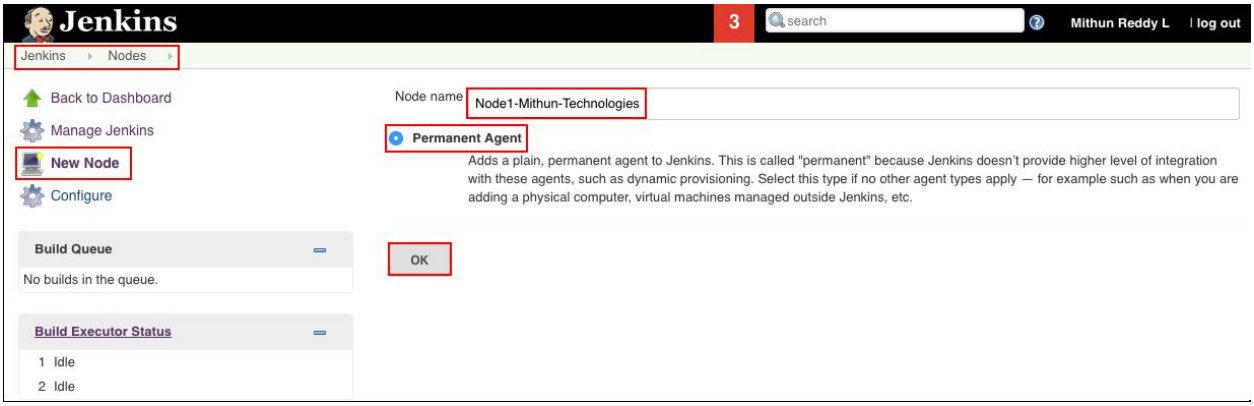
Required Plugins

1) Pipeline: Multibranch

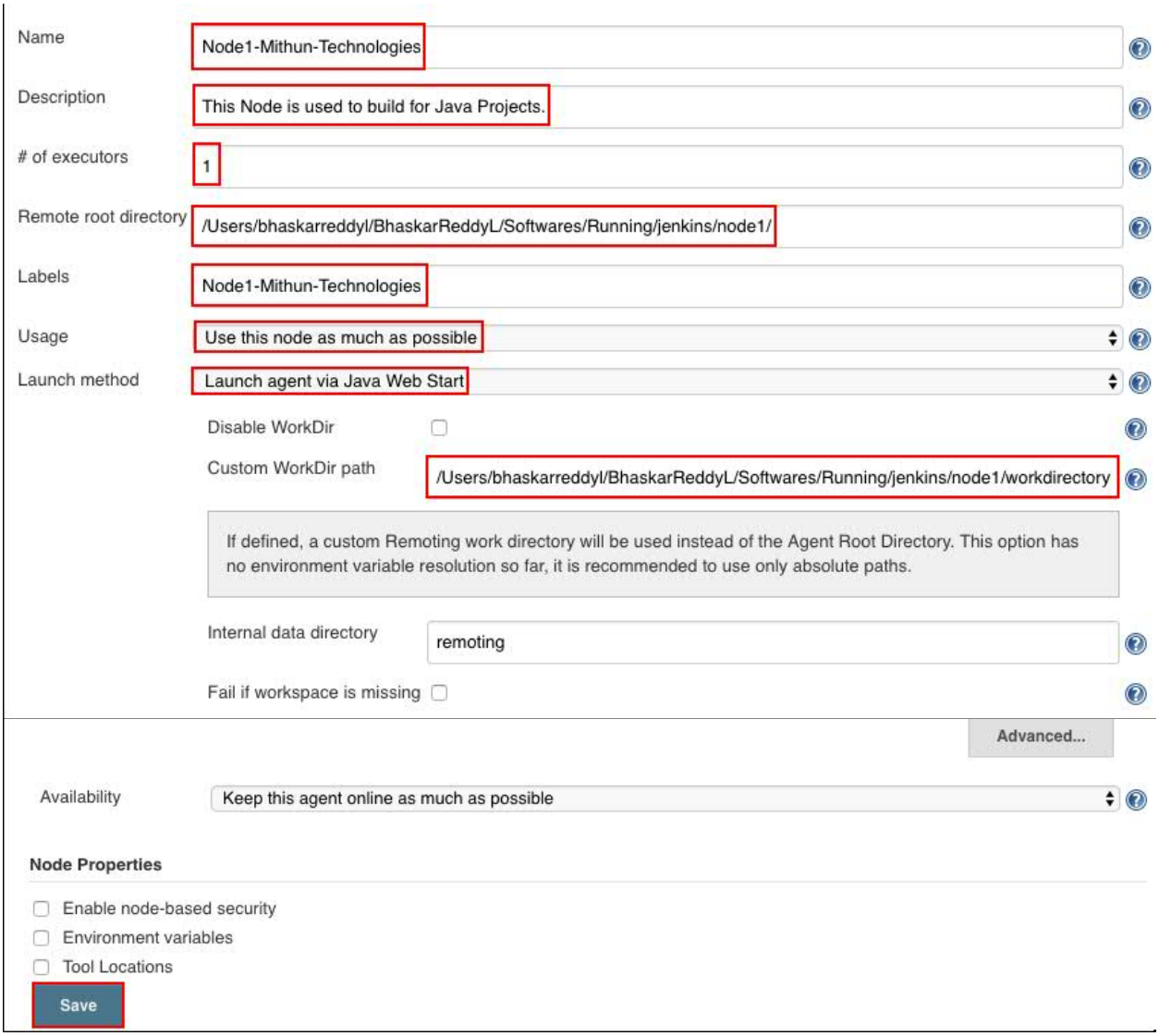
Blue Ocean

Jenkins Master-Slave setup

Manage Jenkins ---> Manage Nodes ---> New Node
Provide the Node name and click on **OK** button.

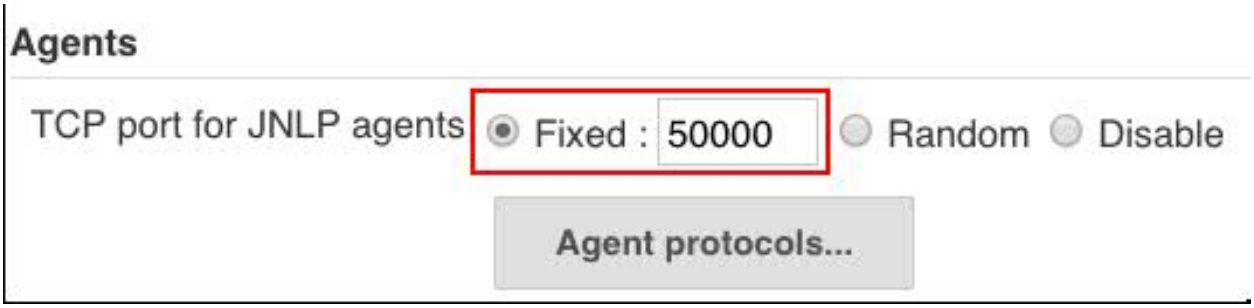


Provide all the details as follows and click on **Save** button.

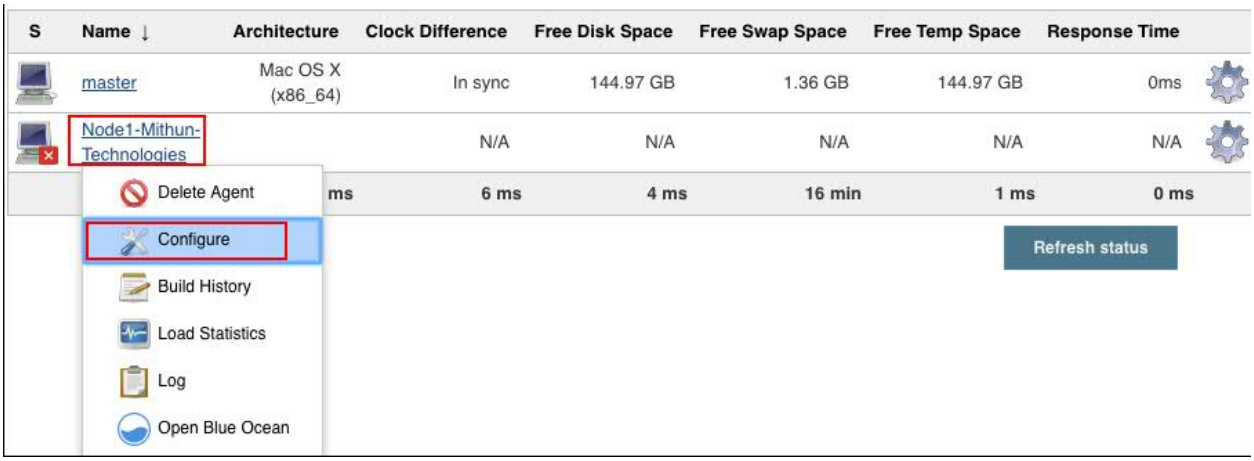


Note: Suppose if you don't see "**Launch agent via Java Web Start**" option, do the below configurations.

Manage Jenkins ---> Configure Global Security ---> enable the **TCP port for JNLP agents** (by default, it is Disabled.)



Once you click on Save you will see the Nodes and Master detail, and select the Node which we have created and click on configure.



You will see below screen and click download the slave.jar file.

Copy slave.jar file into any directory (/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1)

Go to the path where slave.jar copied and run the below command.

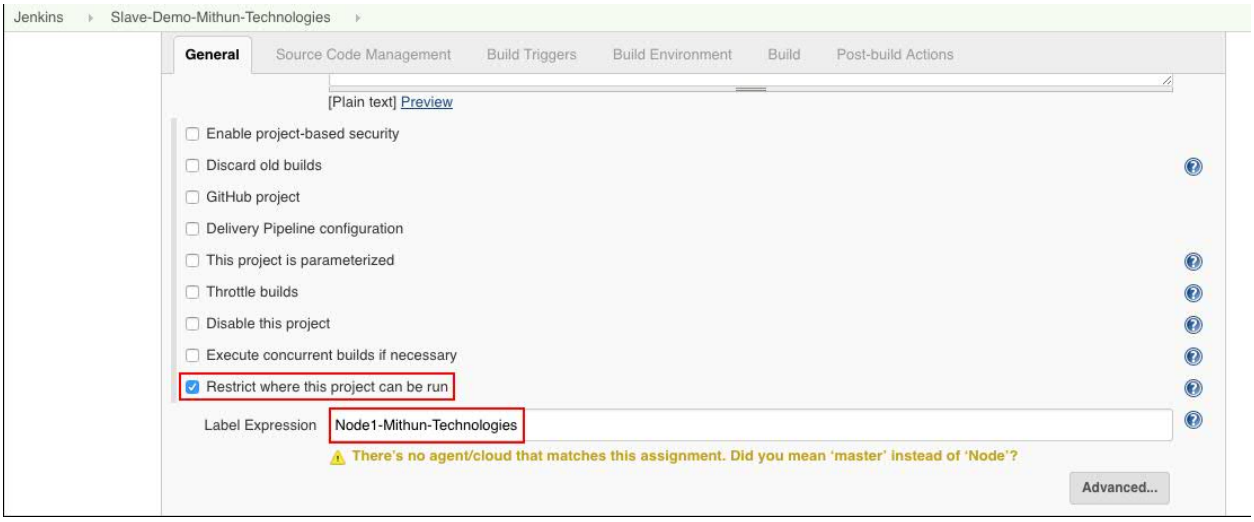
java -jar slave.jar -jnlpUrl http://localhost:8080/computer/Node1-Mithun-Technologies/slave-agent.jnlp -secret 8e6c24c3e977342073d2184d051b1fb87f30d57acd0c63ae0a913008e65ad86f -workDir "/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory"

```
Bhaskars-MacBook-Air:node1 bhaskarreddyl$ java -jar slave.jar -jnlpUrl http://localhost:8080/computer/
Node1-Mithun-Technologies/slave-agent.jnlp -secret 8e6c24c3e977342073d2184d051b1fb87f30d57acd0c63ae0a9
13008e65ad86f -workDir "/Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirecto
ry"
Nov 26, 2017 9:48:30 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory/remoting
as a remoting work directory
Both error and output logs will be printed to /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jen
kins/node1/workdirectory/remoting
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main createEngine
INFO: Setting up slave: Node1-Mithun-Technologies
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener <init>
INFO: Jenkins agent is running in headless mode.
Nov 26, 2017 9:48:31 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /Users/bhaskarreddyl/BhaskarReddyL/Softwares/Running/jenkins/node1/workdirectory/remoting
as a remoting work directory
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://localhost:8080/]
Nov 26, 2017 9:48:31 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, JNLP-connect, Ping, JNLP2-conne
ct]
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: localhost
  Agent port:    50000
  Identity:      96:6e:10:60:c1:c4:f2:e8:7e:4c:d9:c7:01:b3:e1:a3
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to localhost:50000
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
Nov 26, 2017 9:48:31 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 96:6e:10:60:c1:c4:f2:e8:7e:4c:d9:c7:01:b3:e1:a3
Nov 26, 2017 9:48:32 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

Now slave become communicating to node and it is live.

Now you can use this slave for job creation.

Create one Freestyle project/any kind of project and select the Restrict where this project can be run and select the Node which you have crated.



Provide the Git url and click on **Save** button.

Resources:

https://jenkins.io/ ---> Download software
https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service

http://www.tothenew.com/blog/jenkins-implementing-project-based-matrix-authorization-strategy/ ---> User Access

https://support.cloudbees.com/hc/en-us/articles/216118748-How-to-Start-Stop-or-Restart-your-Instance