Setup Jenkins from Scratch

- Beginners guide

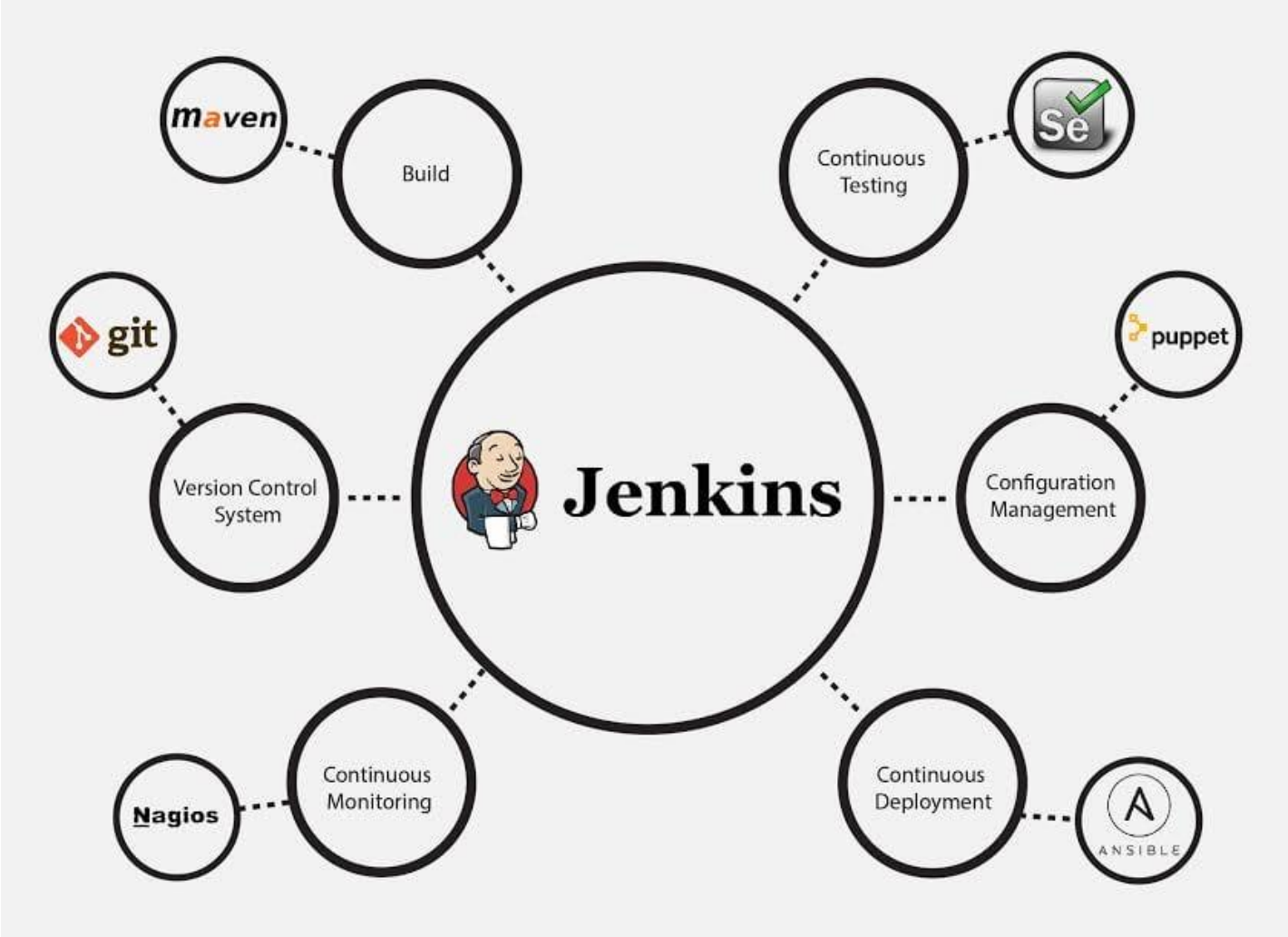
Ver. No.	Ver. Date	Revised By	Description
1.0	May 2020	Kedar Pavaskar	Initial Document

CONTENTS

Continuous integration – Continuous Delivery (CI – CD)

Earlier, there were so many different types of engineers involved in the development process. Such as Developers, Testers, Middleware engineers, release engineers etc.

Most of the tasks done by these various team members mentioned in the above process, is automated by CI/CD tool called Jenkins.



Stage 1 – Continuous Download

Developers create some code and upload it to the version control system (Example GIT0. Jenkins will be integrated with the version control system (using a plugin) in such a way that when the developers upload the modified code Jenkins will receive the notification and it will download the code.

Stage 2 – Continuous Build

The code downloaded in the Stage 1, has to be converted into an artifact. This artifact can be in the format of exe, jar, war, ear file etc. (different types of files in the format of the developing technology they are using). to trigger this build process, Jenkins will use build tools like ANT, MAVEN, MSbuild.

Stage 3 – Continuous Deployment

The artifact created in the Stage 2, has to be deployed into the testing servers. These testing servers might be running on some application server like TOMCAT, JBOSS, weblogic, etc. Jenkins will deploy the artefact into these application servers. So that Testers will start testing the application.

Stage 4 – Continuous Testing

Testers create automation test scripts using tools like Selenium, codeui etc. Jenkins will download this automation scripts and execute them. If testing fails Jenkins will send automated email notifications to developers and testers, Developers will fix the defects and upload the modified code into the version controlling system, then again Jenkins will start from stage 1.

Stage 5 – Continuous Delivery

If testing passes in the previous stage, Jenkins will deploy the artifact into the production servers where the end user or the client can start accessing the application.

The above 5 stages are called as CI/CD.

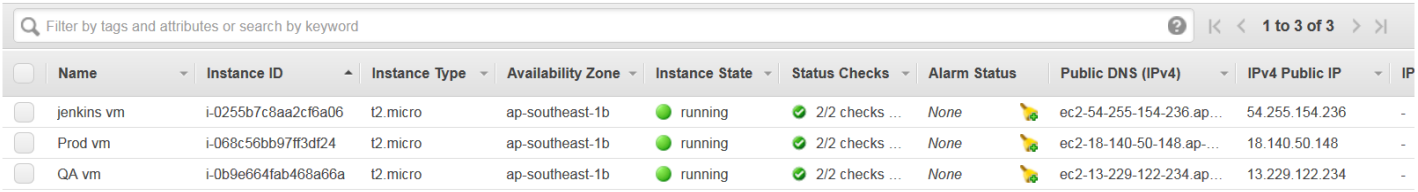
There are few tasks Jenkins cannot automate or takes longer time. Those kind of activities, could be automated via python.

Example: There might be 100 Jobs listed in Jenkins, which belong to a particular client/project. Jenkins may not be able to delete those jobs in one go and you have to do manual tasks of deleting the jobs. In such cases, Python scripting can come handy.

LAB testing - Installing Jenkins

3 Vms have been setup on AWS for the lab setup perspective.

- Jenkins vm
- Prod VM
- QA vm



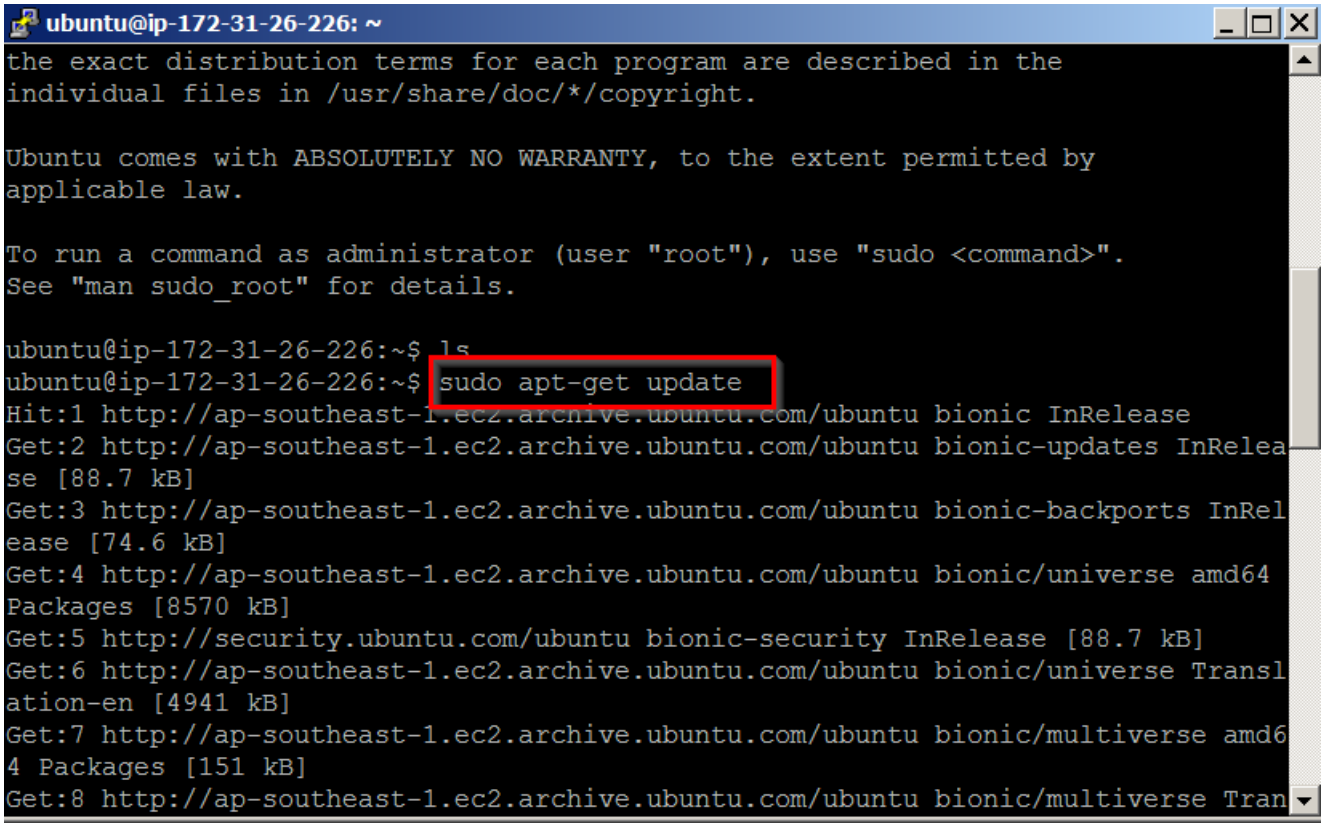
NOTE: this installation of Jenkins instance on the cloud is not ideal, as every time we start the jenkins server we'd need to start the Jenkins.war file manually by running Java –jar jenkins.war.

Refer to Alternative installation(permanent) methods in the document below.

Connect to the Jenkins VM via putty or git bash.

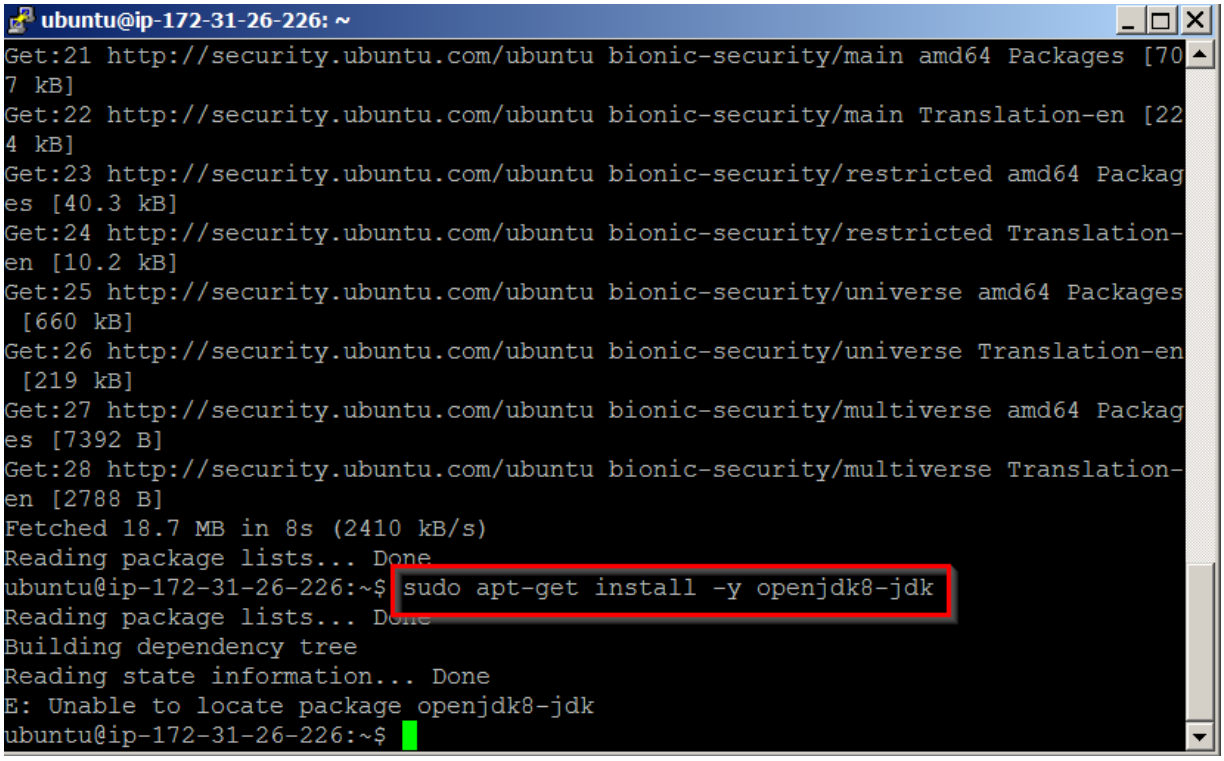
1. Update the apt repository

sudo apt-get update



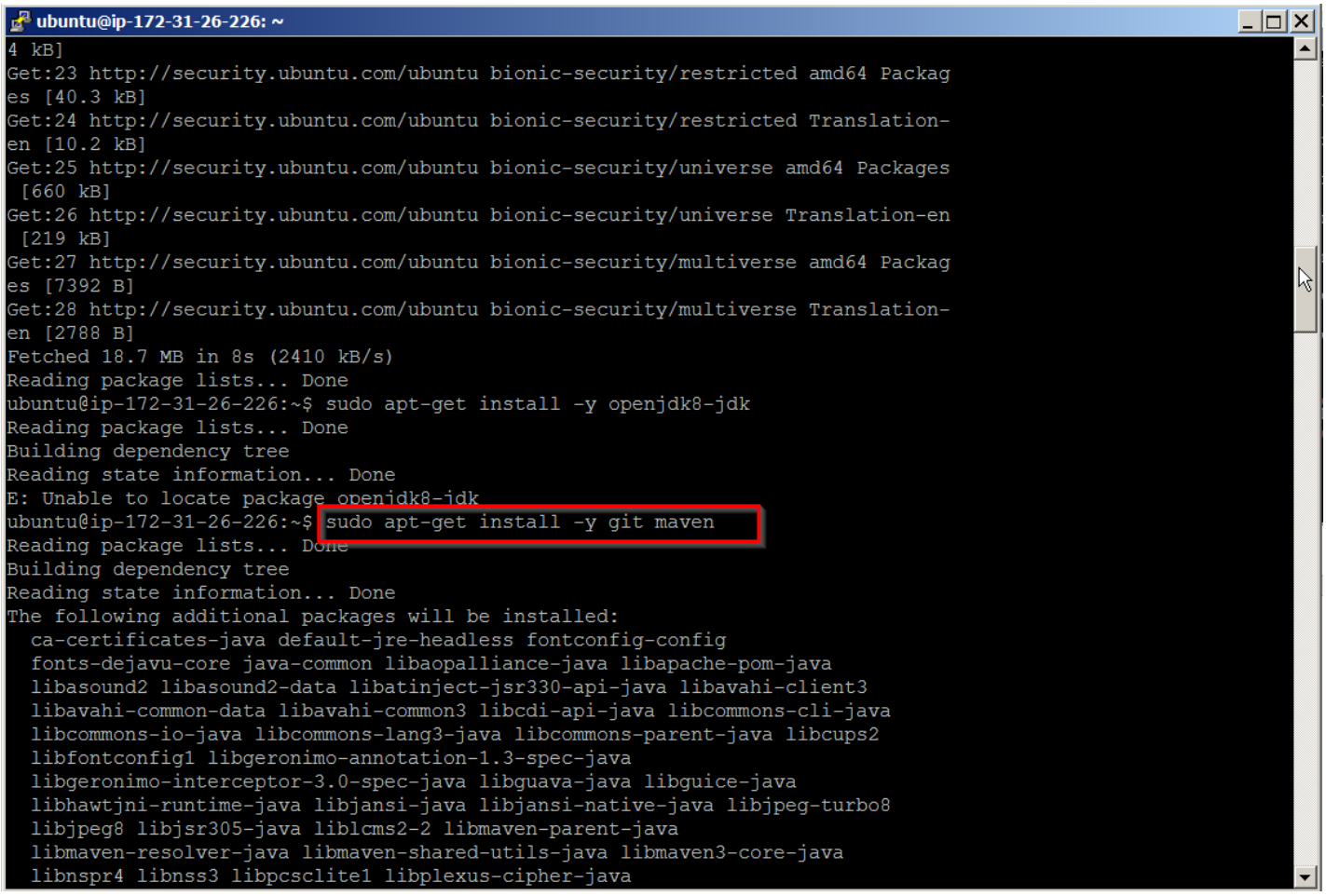
2. Install java

sudo apt-get install -y openjdk-8-jdk



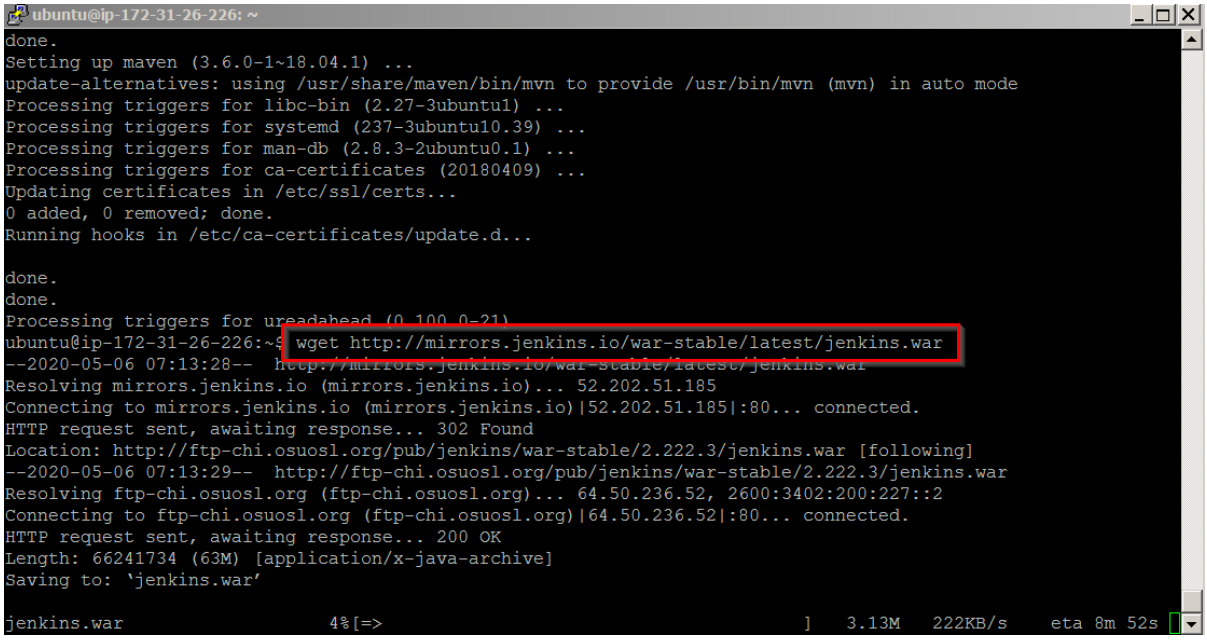
3. Install git and maven

sudo apt-get install -y git maven



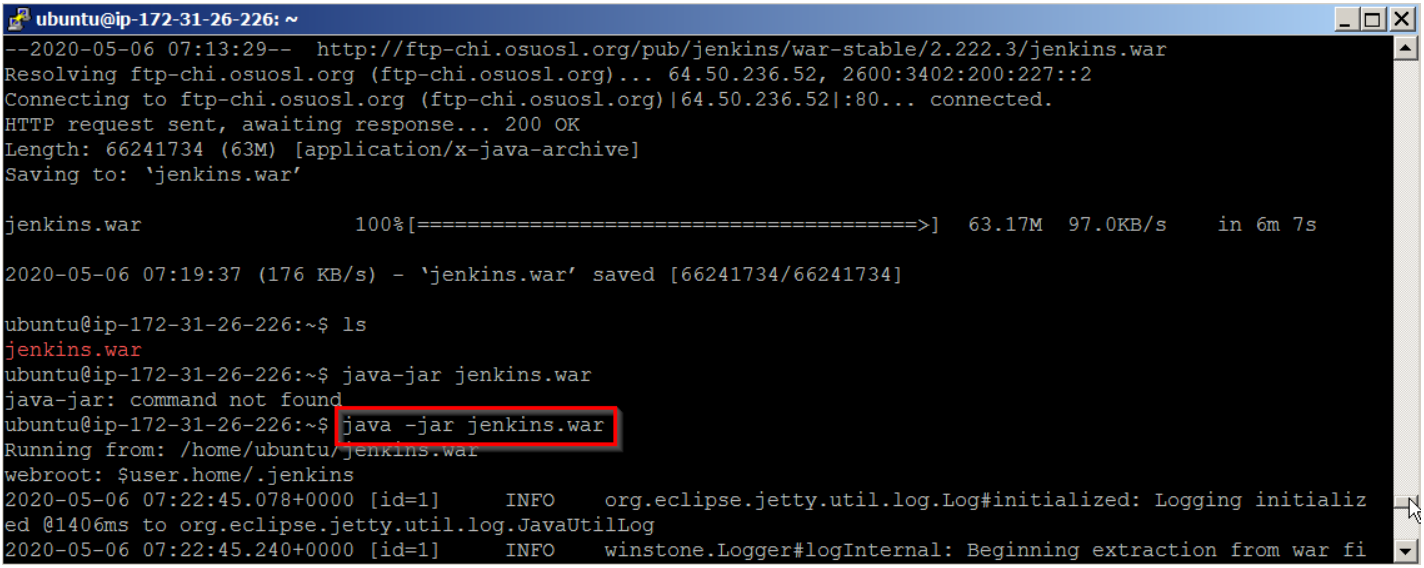
4. Download jenkins.war

wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war



5. To start jenkins

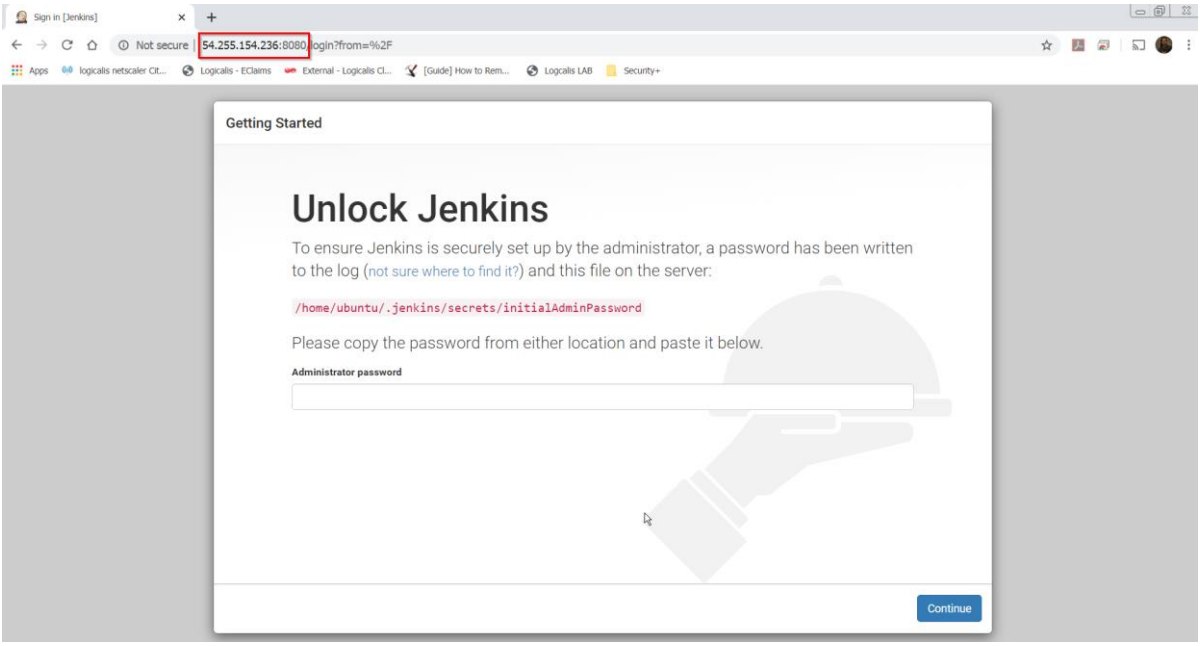
Java -jar jenkins.war



6. To access Jenkins

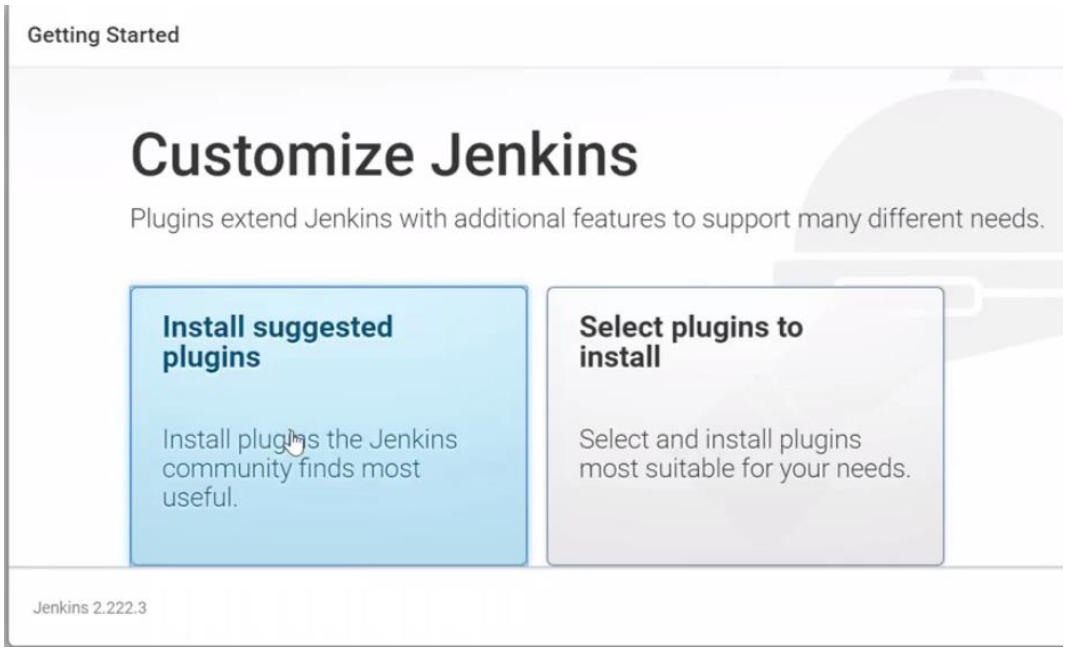
a) Launch any browser

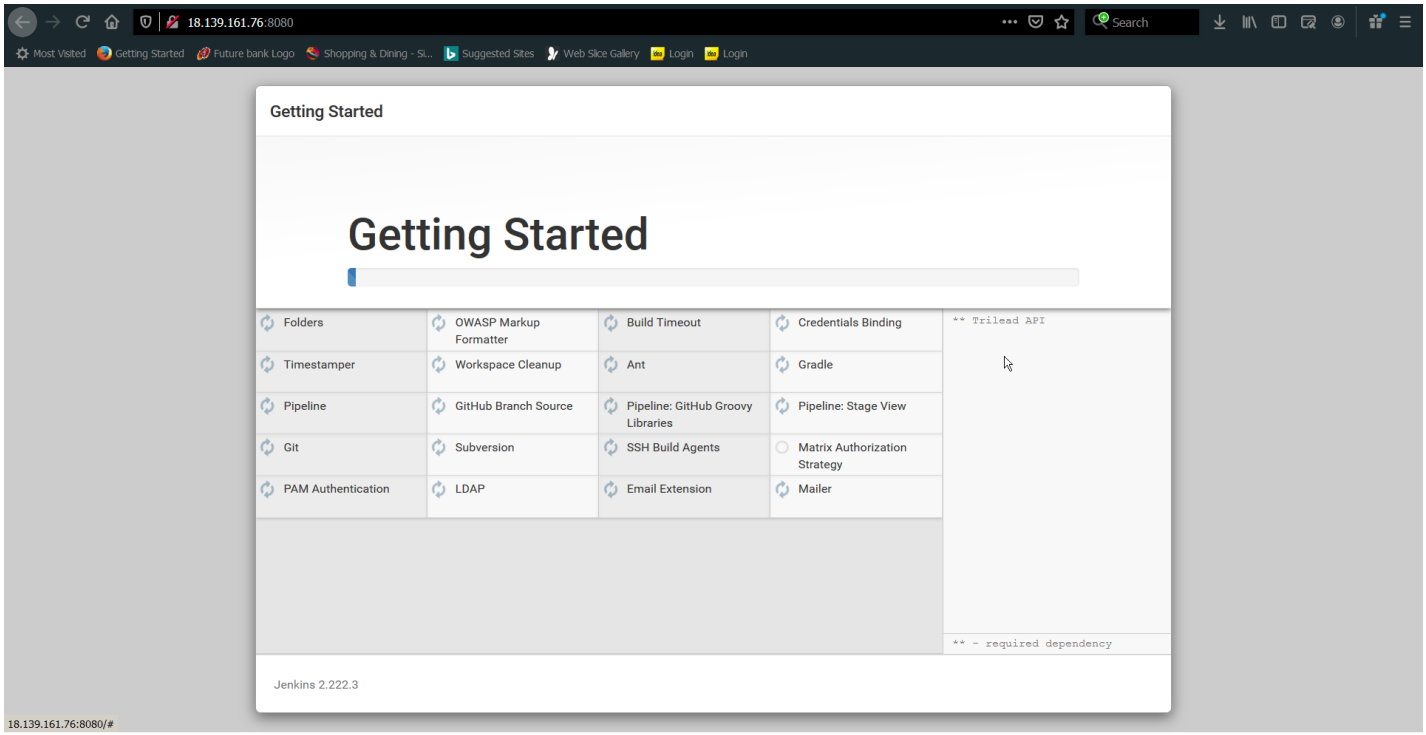
b) publicip_of_jenkinsServer:8080



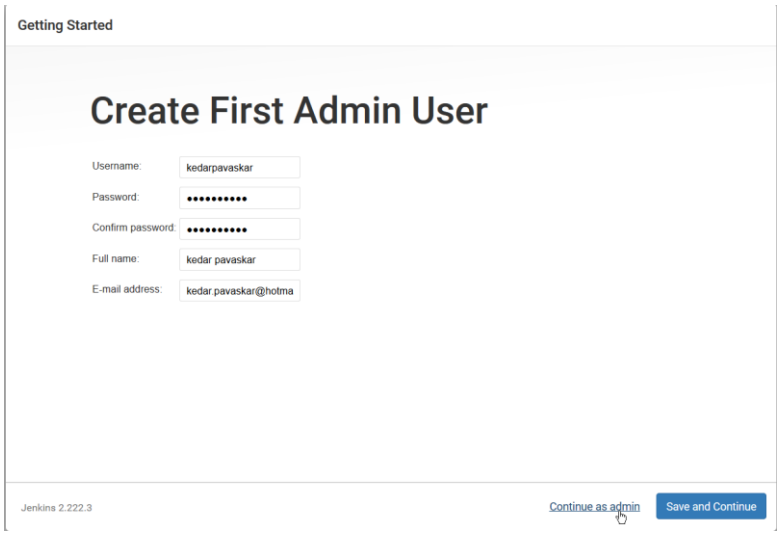
7. In the Unlock Jenkins Screen enter the admin password

8. Click on Install suggested plugins





Create first admin user--->Continue



Click on Finish

LAB testing - Installing Production and QA server

Setting up tomcat on QAServer and ProdServers

1. Connect to QAserver AWS instance using Gitbash

2. Update the apt repository

sudo apt-get update

3. Install tomcat8

sudo apt-get install -y tomcat8

4. Install tomcat8-admin

sudo apt-get install -y tomcat8-admin

5. Edit the tomcat-users.xml file

sudo vim /etc/tomcat8/tomcat-users.xml file

Delete all the content of the file and copy the below code

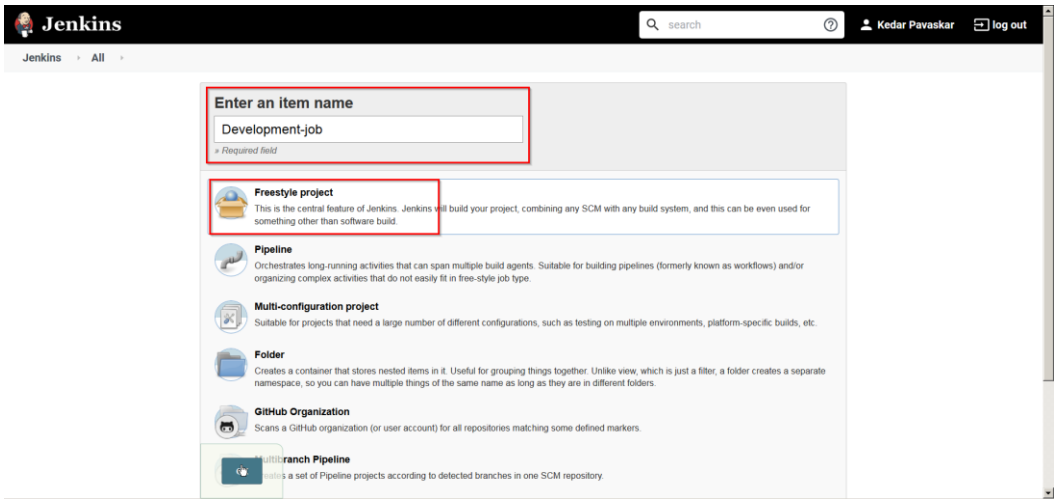
```
<tomcat-users>
    <user username="intelliqit" password="intelliqit" roles="manager-script"/>
</tomcat-users>
```

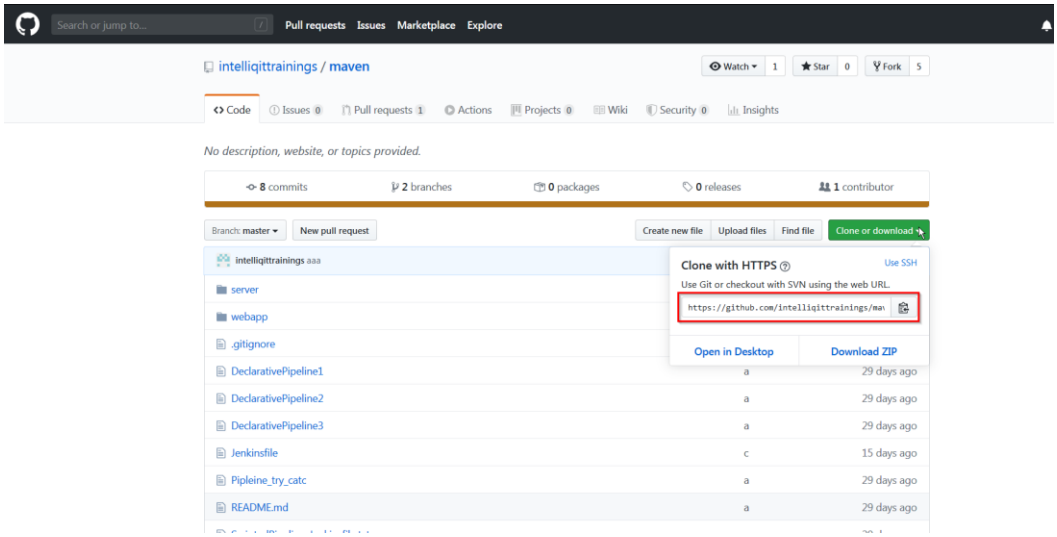
6. Restart tomcat8

sudo service tomcat8 restart

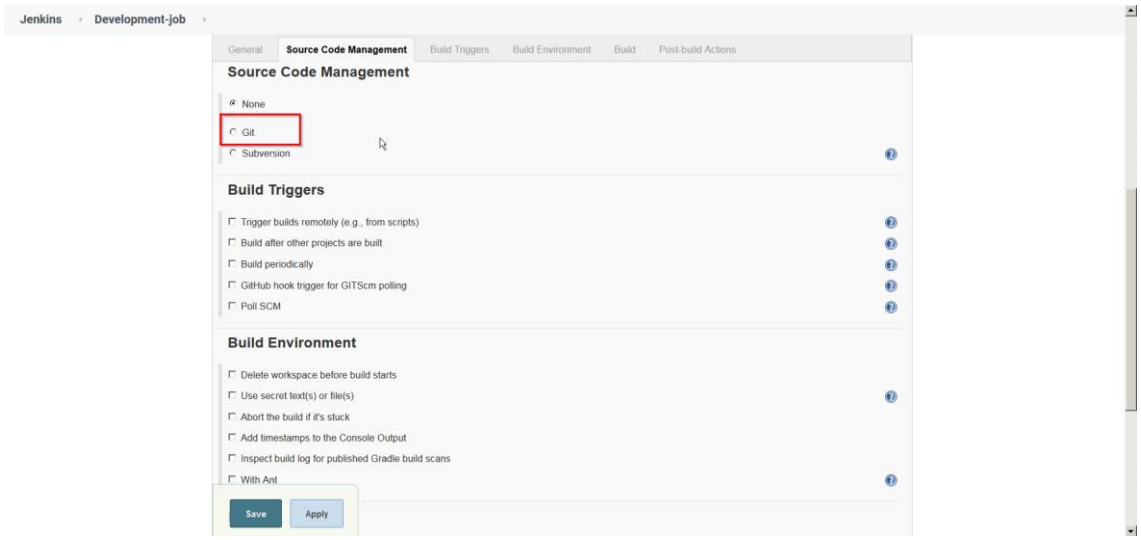
Repeat the above 6 steps on the ProdServer AWS instance

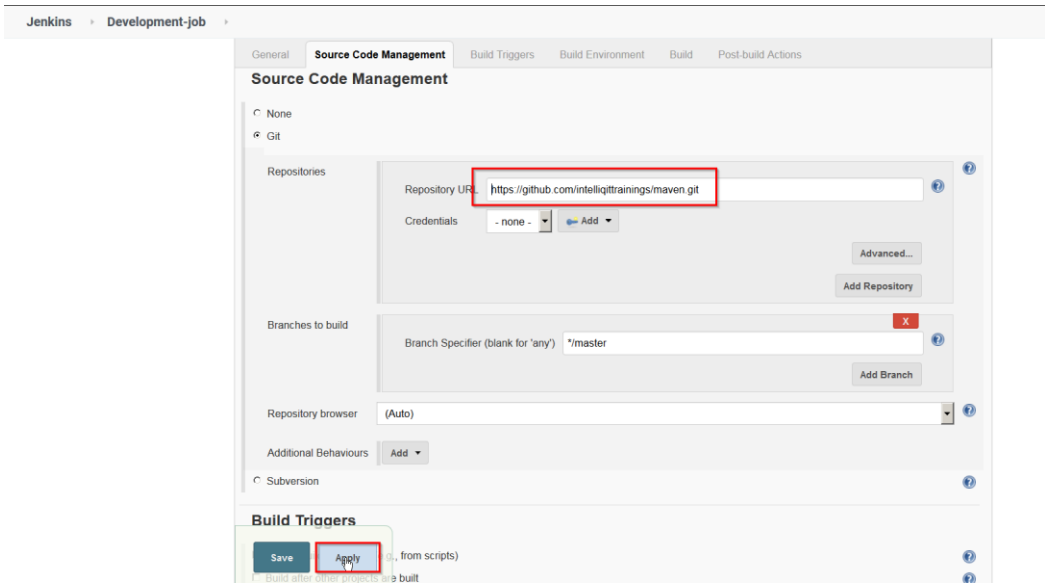
LAB testing Stage 1 - Continuous Download configuration



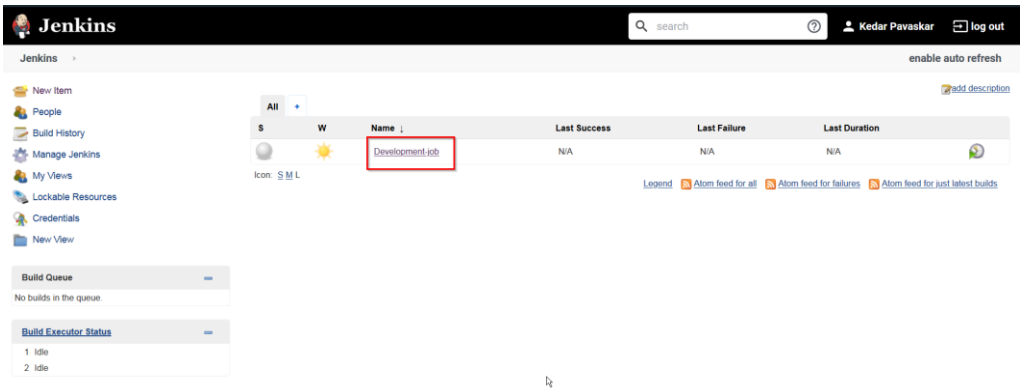


Example: https://github.com/intelliqittrainings/maven.git



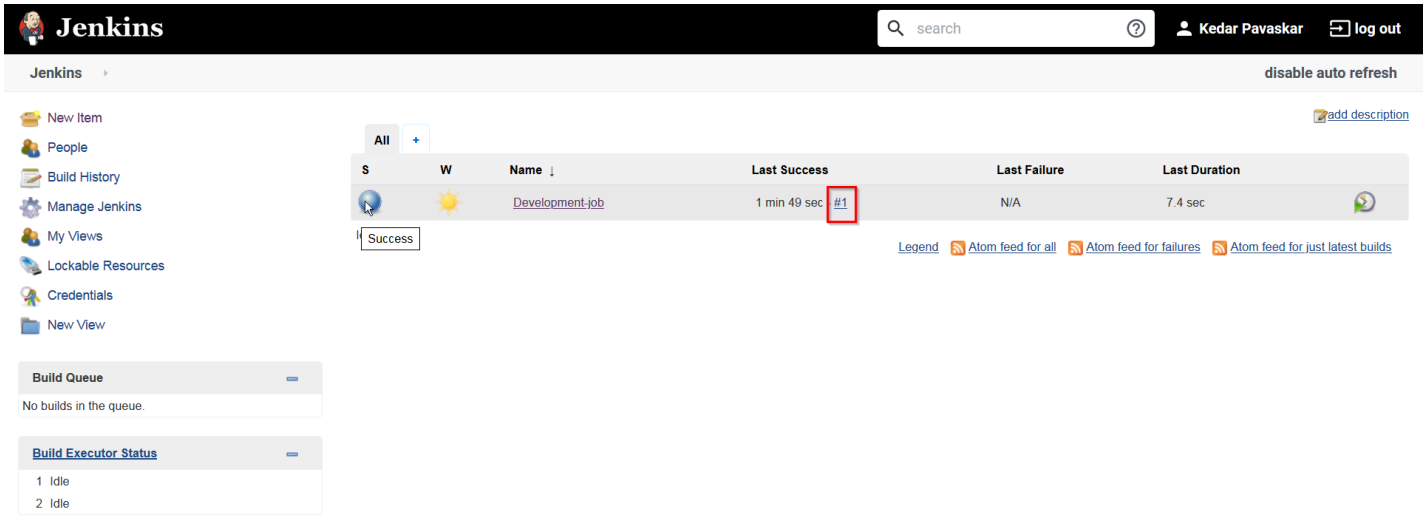


Mentioned the path of the Remote repository that the developer supposedly uploads the code.



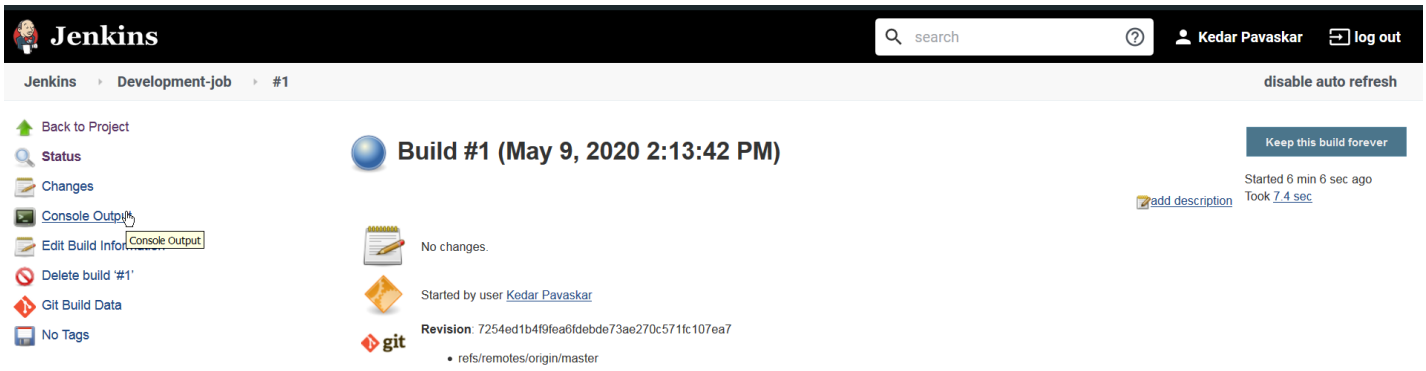
“Development-job” you created earlier must be visible on the dashboard now.

This is stage1 of the CI/CD process. “Continuous Download”

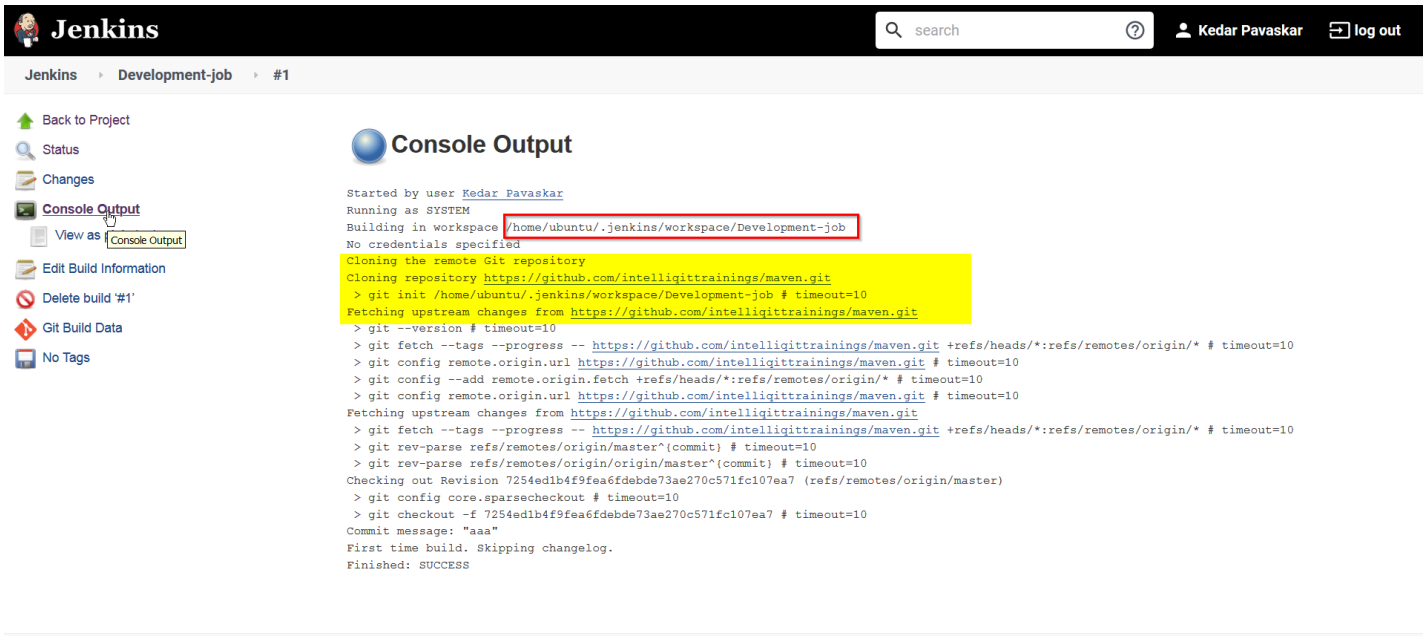


Running the build task will run the job and “build” the code on the Jenkins server.

“#1“ – in the above screenshot shows that the code has been built once. Click on the #1 will take you to the screen where you can check the code’s console output as seen below.



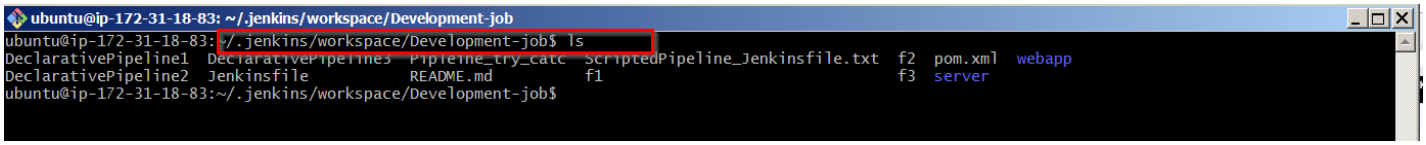
Click on console output as seen below.



You can see that the code has been downloaded locally on the jenkins vm, under the location.

`/home/ubuntu/.jenkins/workspace/Development-job`

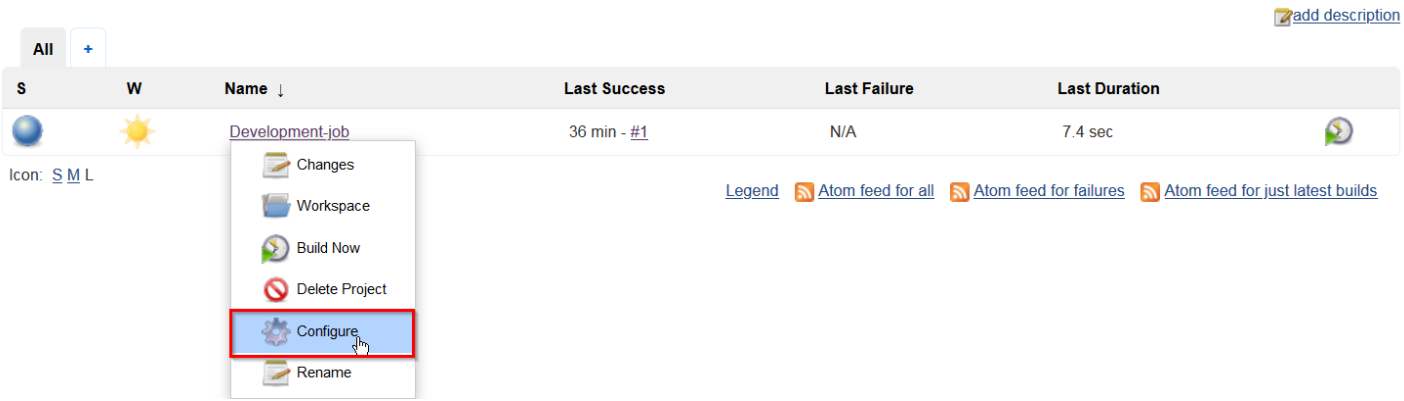
You can verify the same on the Jenkins VM by navigating to the path mentioned above.



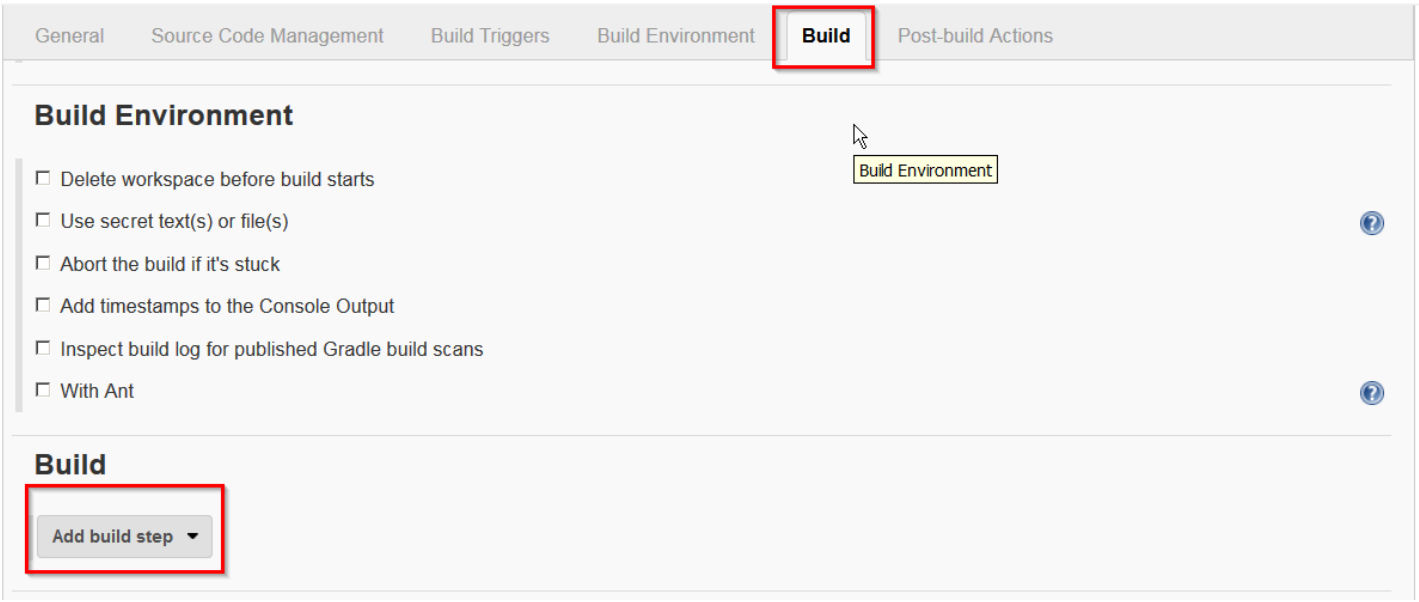
Basically the first stage is Continuous Download, which has been completed above. Jenkins has successfully cloned/Downloaded the GitHub repository (Remote Repository server in our case) we mentioned.

LAB testing Stage 2 - Continuous Build configuration

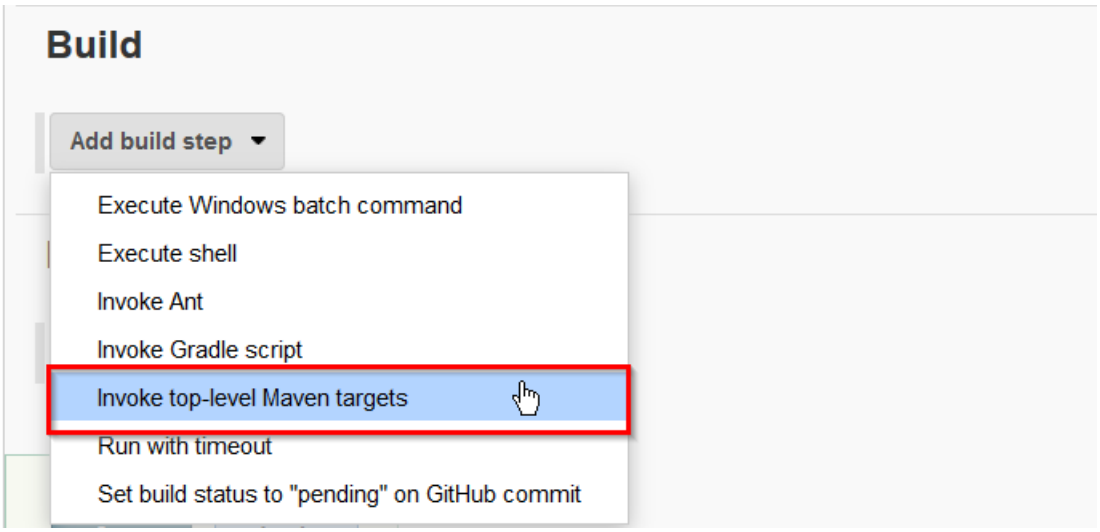
To proceed with the build stage config for jenkins, goto the same job and click on configure.



Select the build tab and click “add build step”



Click Invoke top-level Maven targets.

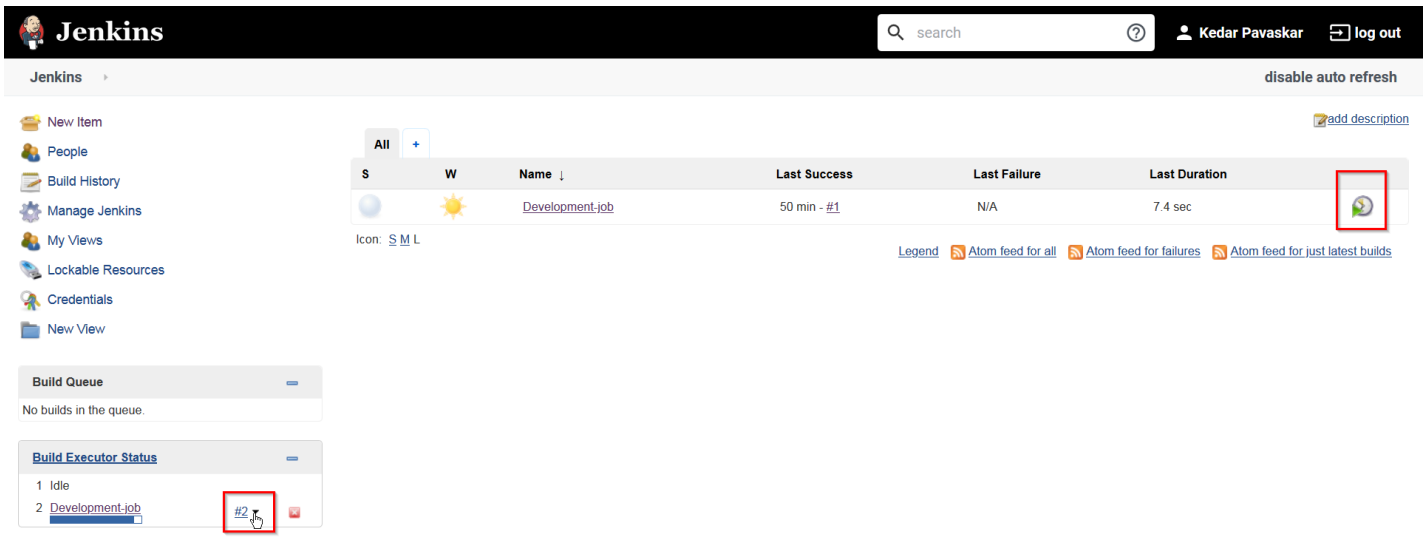


Under build tab in jenkins, "Invoke-top level maven targets" is for maven build tasks for java based projects.

Enter ‘package’ as shown above in the goals text box. And click on Save.

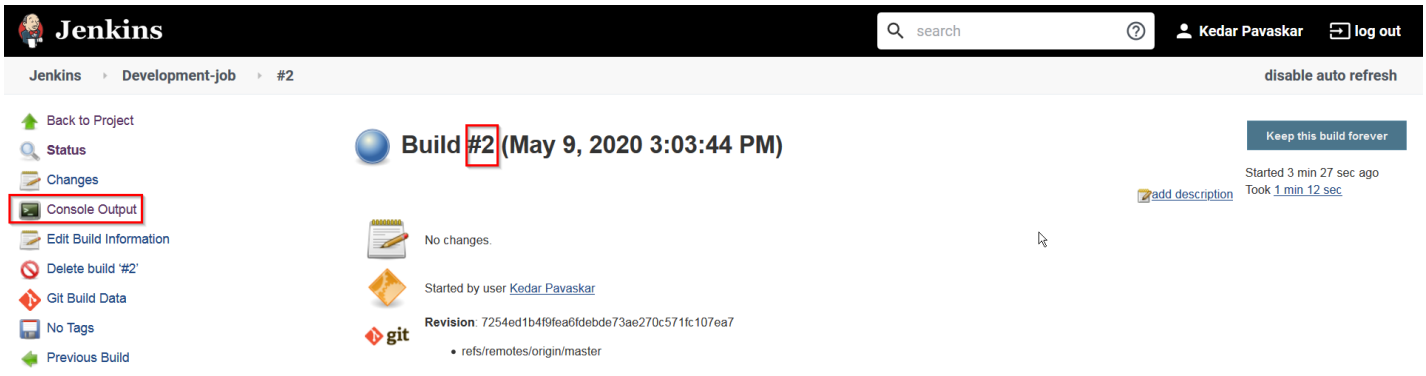
Note: *Maven has 7 stages in its lifecycle, the 5th stage is called as “package” which is used to build an artifact.*

Package is a stage in the Maven Lifecycle, where the code developed by the developer, will converted into an artifact or setup file.

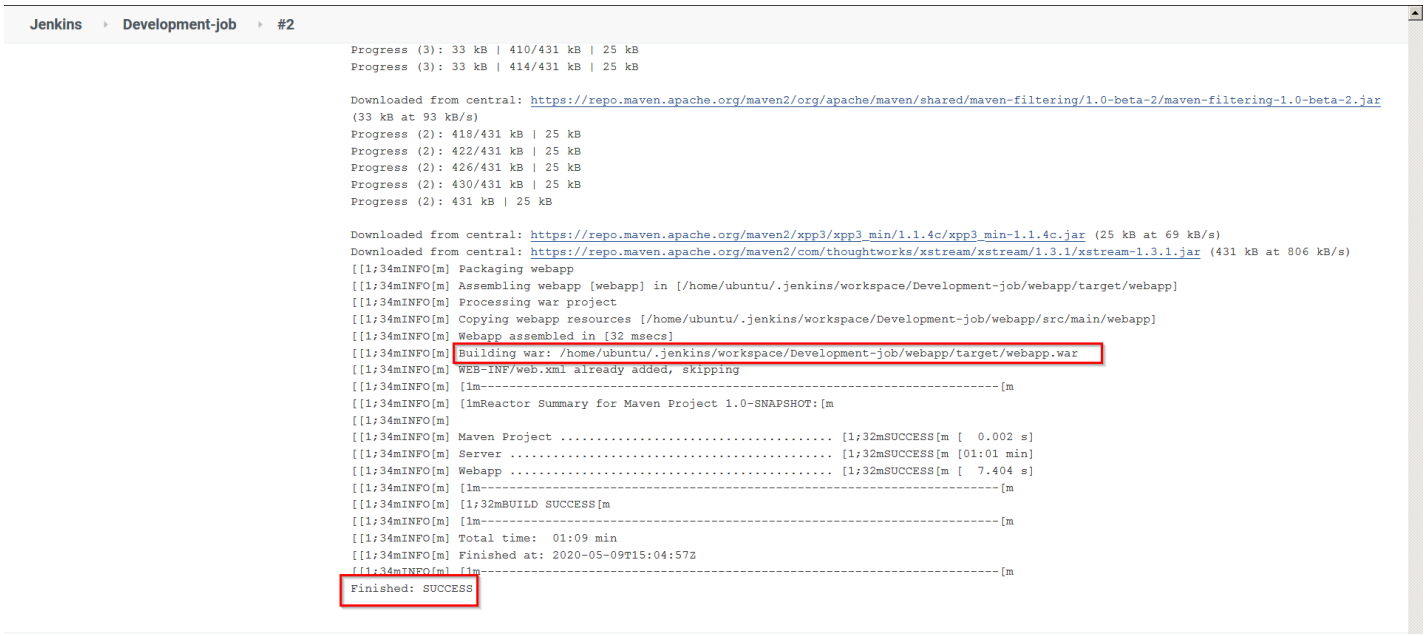


Now click on the build button, doing so will run the job twice which is continuous Download and continuous build.

But since our repository hasn’t changed, the stage1 download doesn’t happen since jenkins doesn’t detect any modifications on the code in the repository.

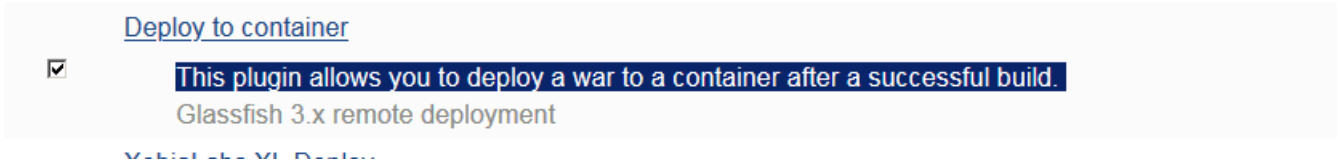


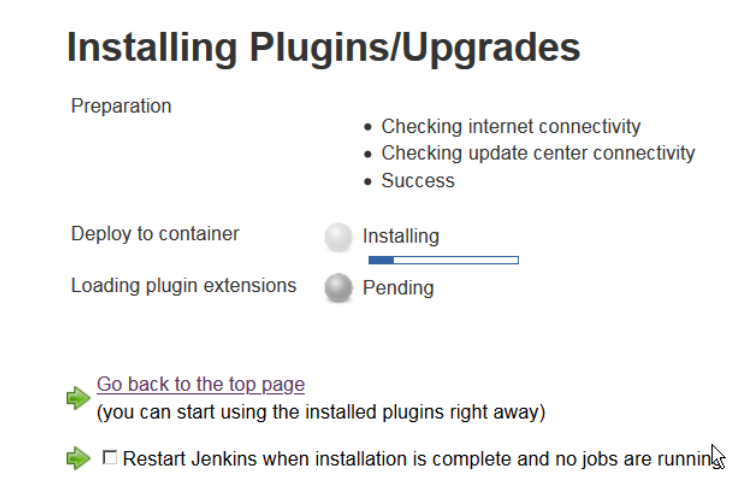
Click on the console output.

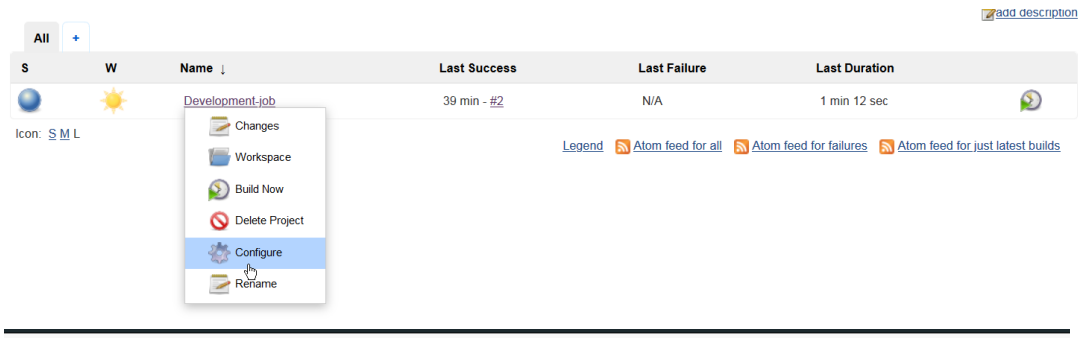


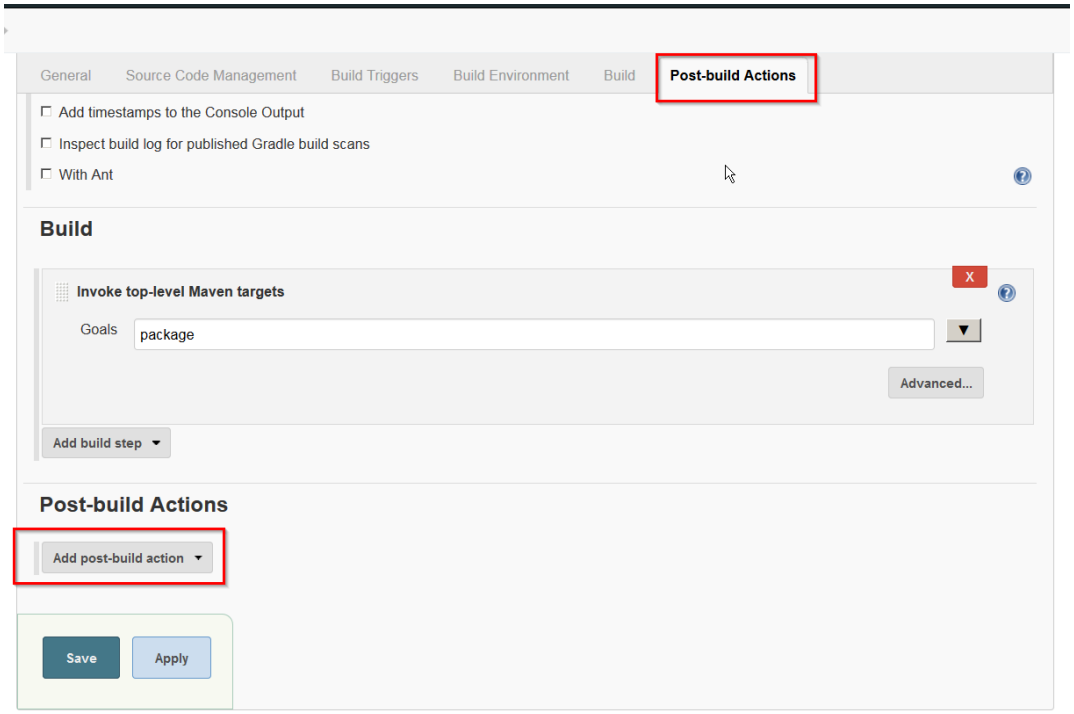
Which will build the Artifact in form of a War file and download it to the jenkins server. This concludes the Continuous Build process.

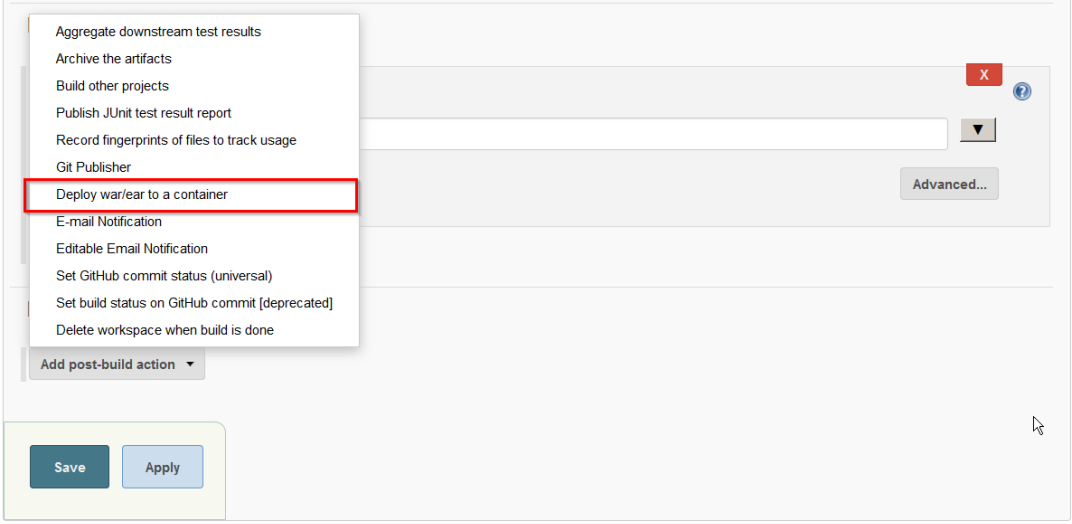
LAB testing Stage 3 - Continuous Deploy configuration on jenkins



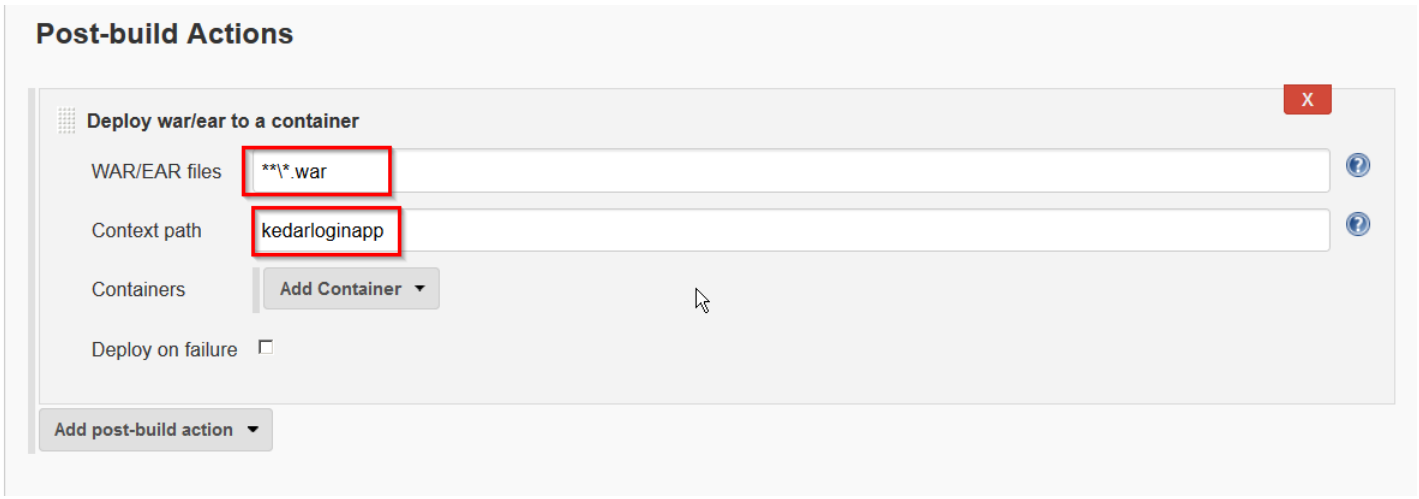




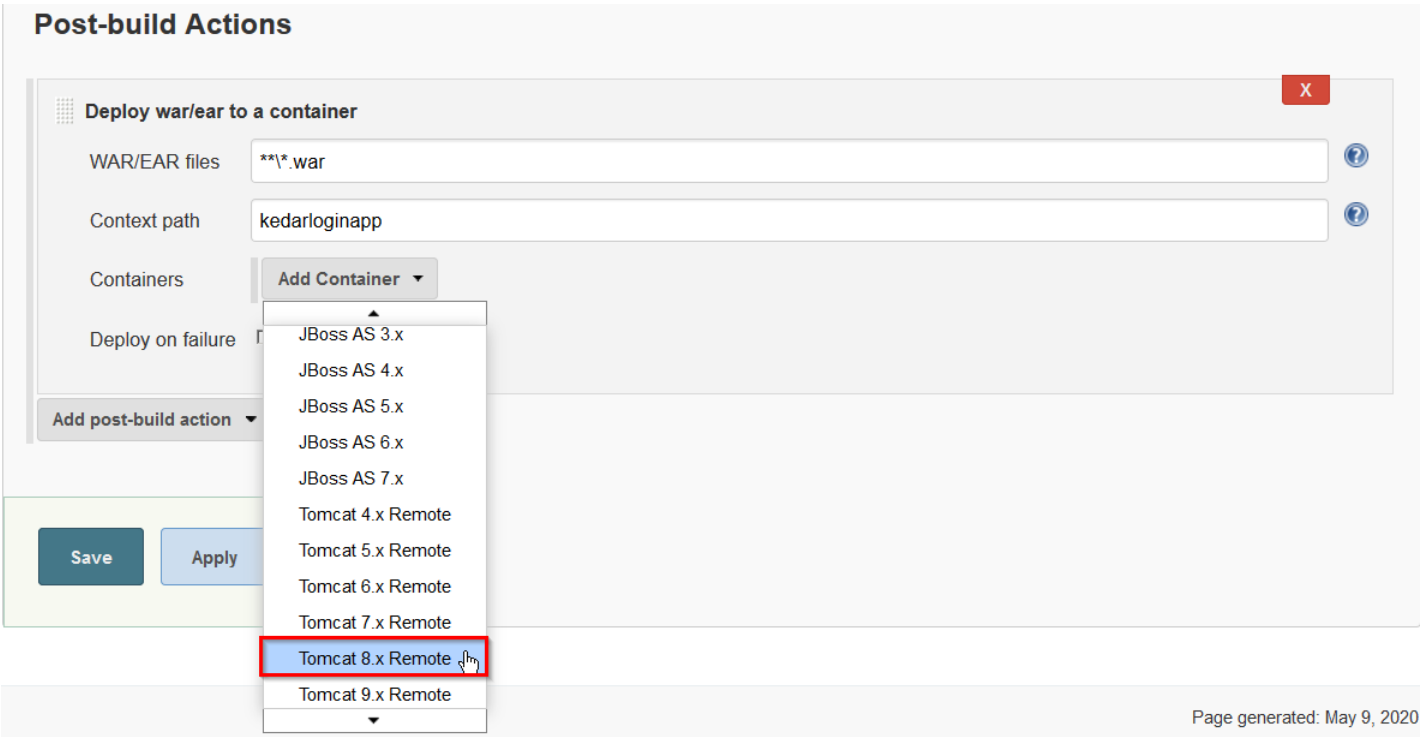


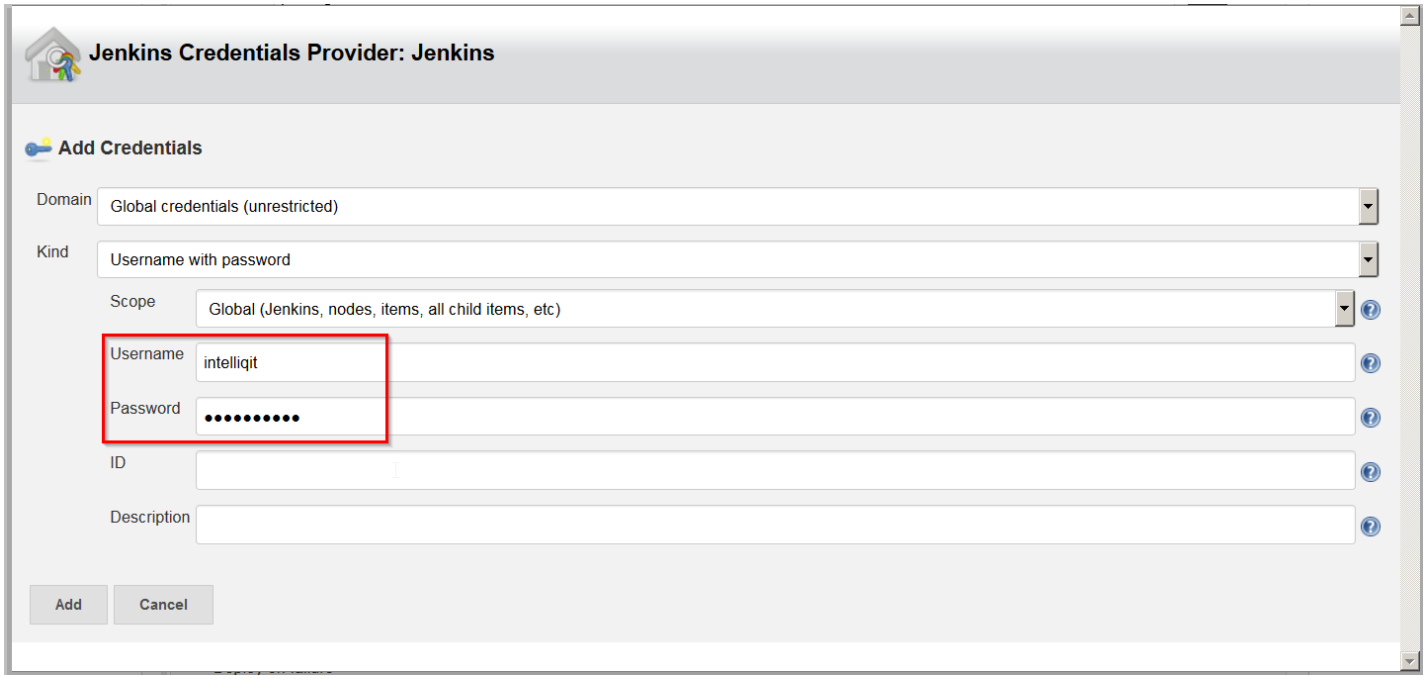


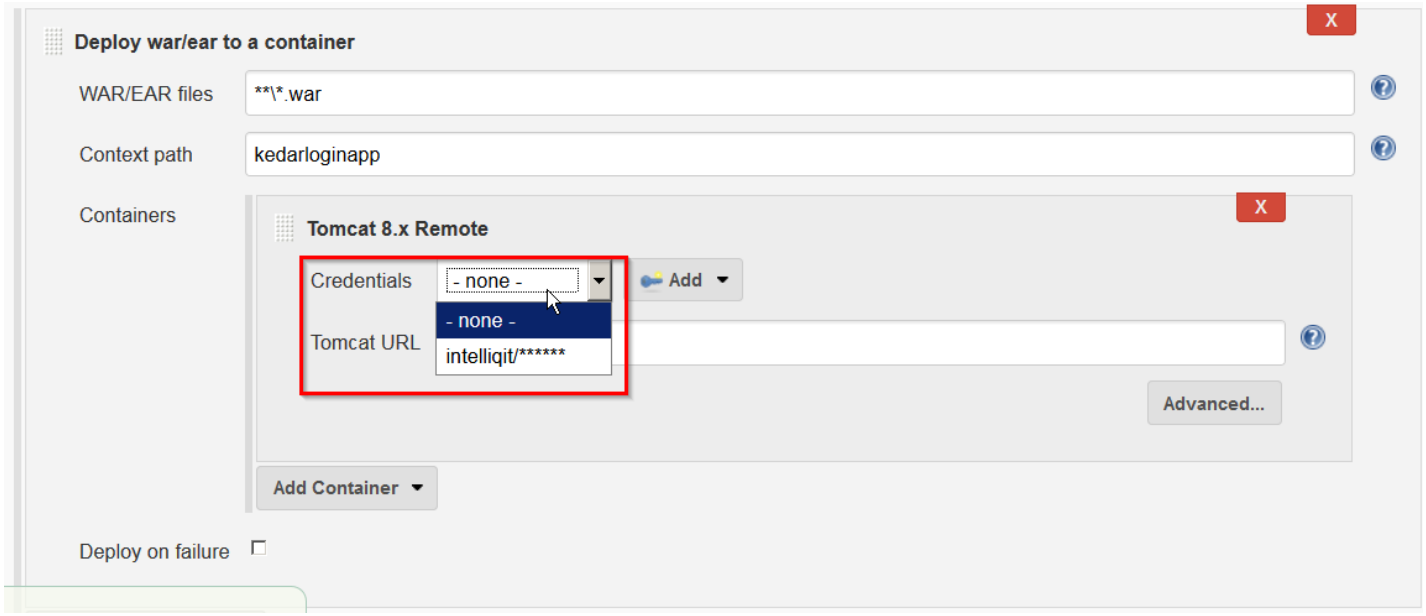
The deploy War/ear to a container option is available only after the “Deploy with Container” plugin is installed. This is used to deploy war files into a tomcat container.

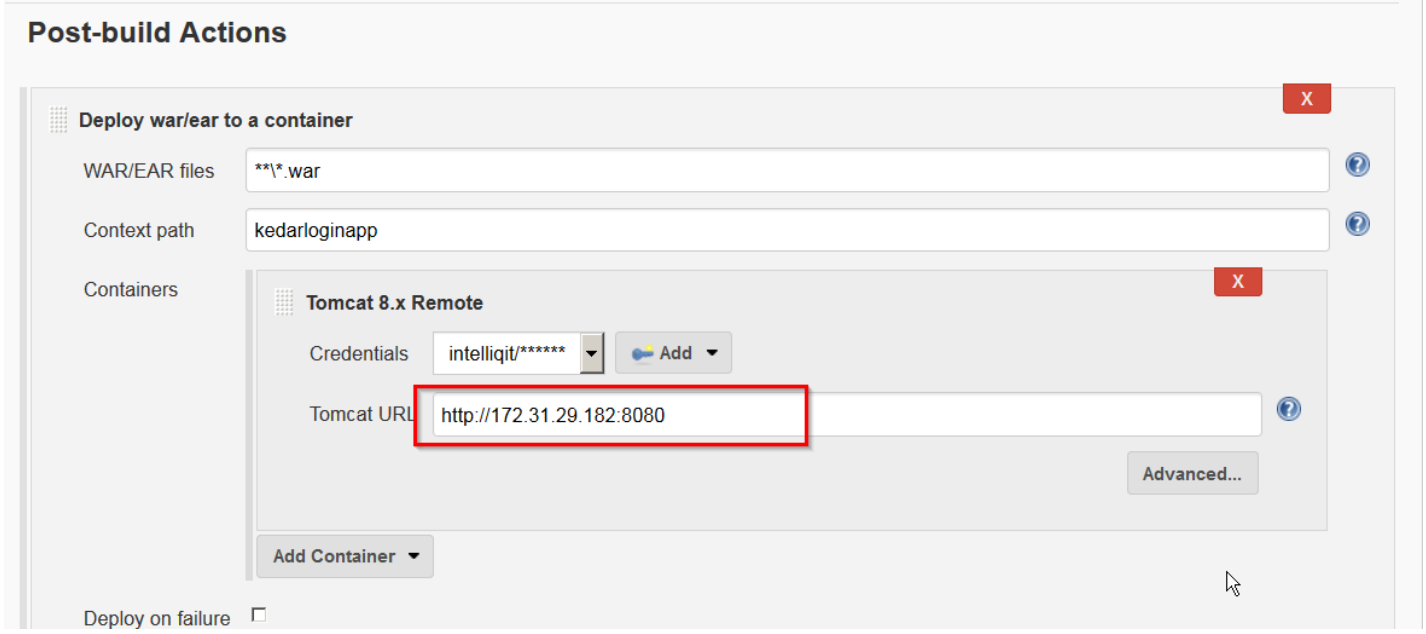


***.war** means in the current working directory it will look for any file with Extention .war file. This essentially is asking jenkins to pick up all war files from the current working directory, since this is a basic test deployment there is only one war file.





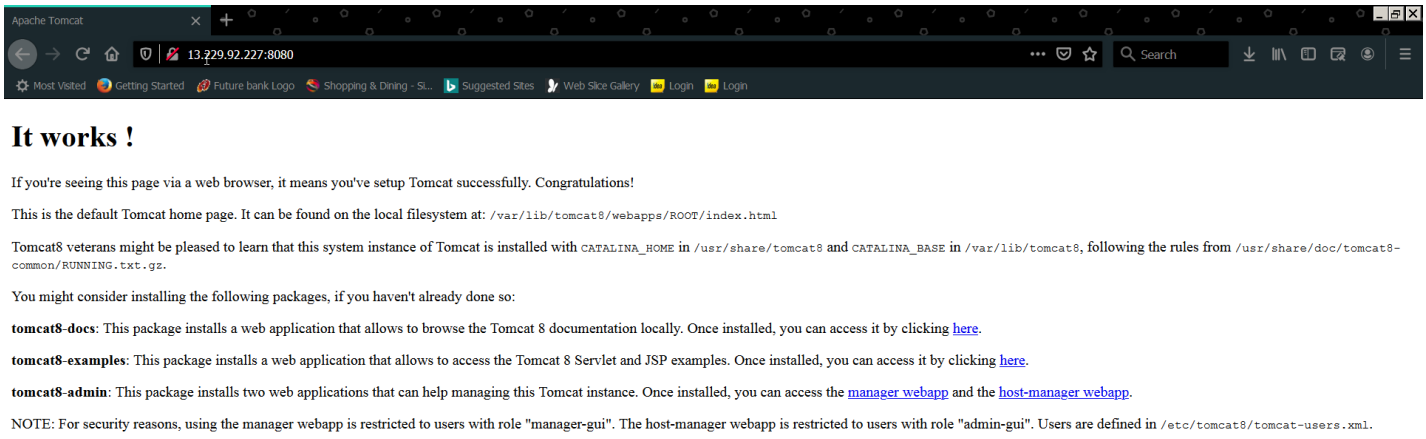




Select the QA server Internal IP and click save. And goto the Dashboard, select the job and click build.

```
-------------------------------------------------------

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[[1;34mINFO[m]
[[1;34mINFO[m] [1m--- [0;32mmaven-war-plugin:2.2:war[m [1m(default-war)[m @ [36mwebapp[0;1m ---[m
[[1;34mINFO[m] Packaging webapp
[[1;34mINFO[m] Assembling webapp [webapp] in [/home/ubuntu/.jenkins/workspace/Development-job/webapp/target/webapp]
[[1;34mINFO[m] Processing war project
[[1;34mINFO[m] Copying webapp resources [/home/ubuntu/.jenkins/workspace/Development-job/webapp/src/main/webapp]
[[1;34mINFO[m] Webapp assembled in [34 msecs]
[[1;34mINFO[m] Building war: /home/ubuntu/.jenkins/workspace/Development-job/webapp/target/webapp.war
[[1;34mINFO[m] WEB-INF/web.xml already added, skipping
[[1;34mINFO[m] [1m------------------------------------------------------------------------[m
[[1;34mINFO[m] [1mReactor Summary for Maven Project 1.0-SNAPSHOT:[m
[[1;34mINFO[m]
[[1;34mINFO[m] Maven Project ..................................... [1;32mSUCCESS[m [  0.002 s]
[[1;34mINFO[m] Server ............................................ [1;32mSUCCESS[m [  2.406 s]
[[1;34mINFO[m] Webapp ............................................ [1;32mSUCCESS[m [  0.878 s]
[[1;34mINFO[m] [1m------------------------------------------------------------------------[m
[[1;34mINFO[m] [1;32mBUILD SUCCESS[m
[[1;34mINFO[m] [1m------------------------------------------------------------------------[m
[[1;34mINFO[m] Total time:  3.620 s
[[1;34mINFO[m] Finished at: 2020-05-09T16:21:19Z
[[1;34mINFO[m] [1m------------------------------------------------------------------------[m
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /home/ubuntu/.jenkins/workspace/Development-job/webapp/target/webapp.war to container Tomcat 8.x Remote with
context kedarloginapp
  [/home/ubuntu/.jenkins/workspace/Development-job/webapp/target/webapp.war] is not deployed. Doing a fresh deployment.
  Deploying [/home/ubuntu/.jenkins/workspace/Development-job/webapp/target/webapp.war]
Finished: SUCCESS
```

On the console output, you can see the build is successful. Now try browsing the App on the public IP of the QA server we just deployed.

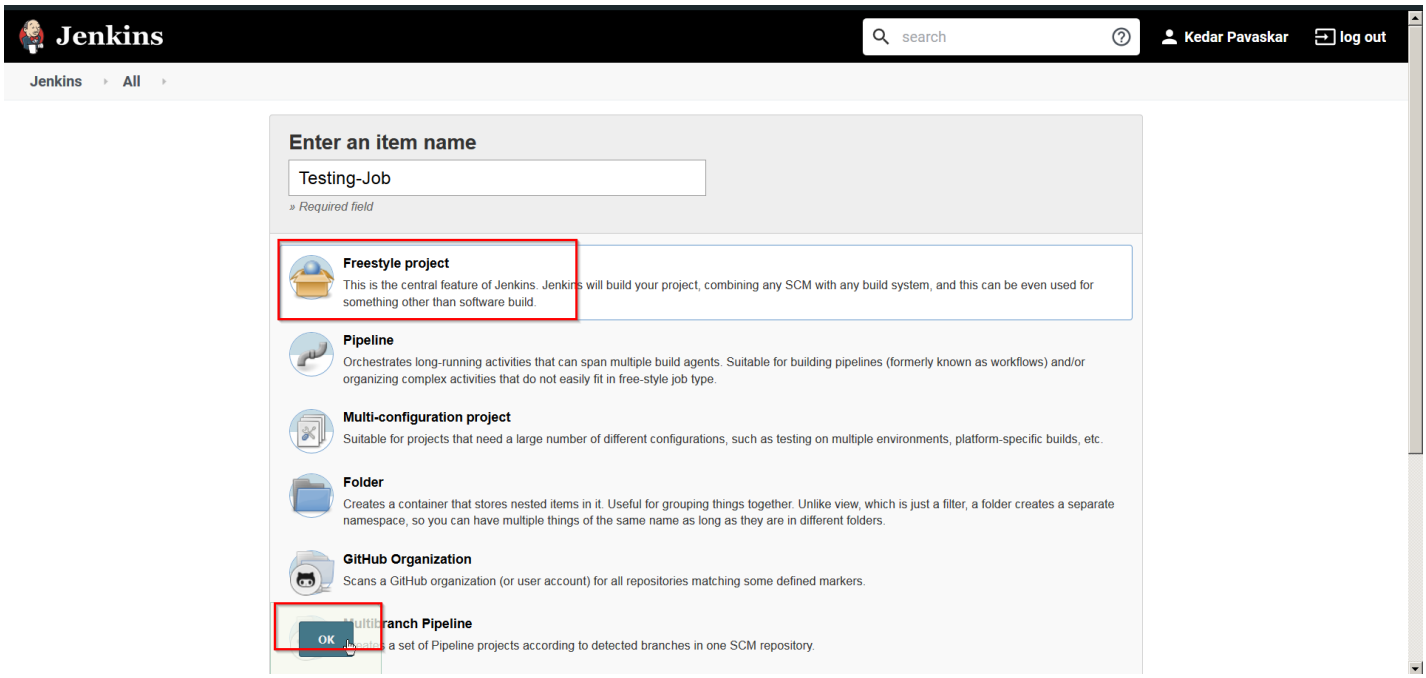


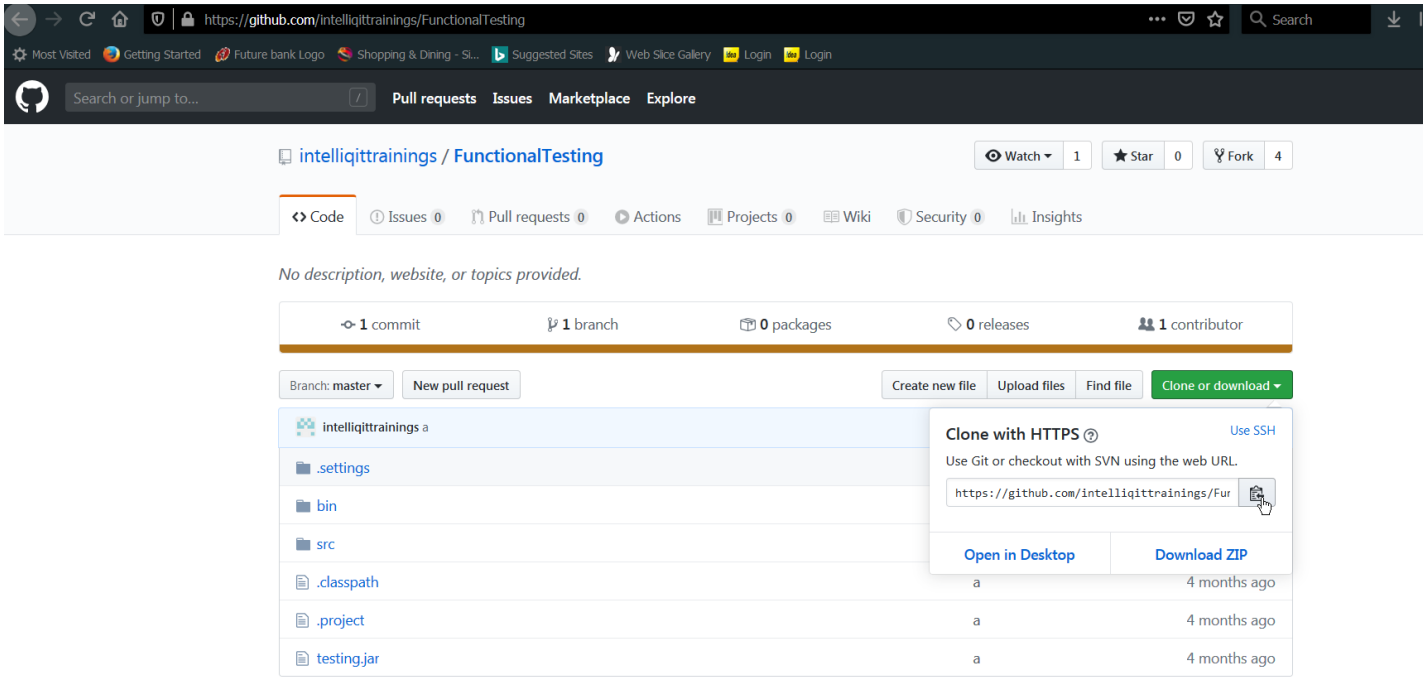
You can see the webpage is working. Now add the contextpath we mentioned to browse the java servlet we have deployed.

After adding the context path we mentioned in the config, we are able to successfully connect to the app.

LAB testing Stage 4 - Continuous Testing configuration on jenkins

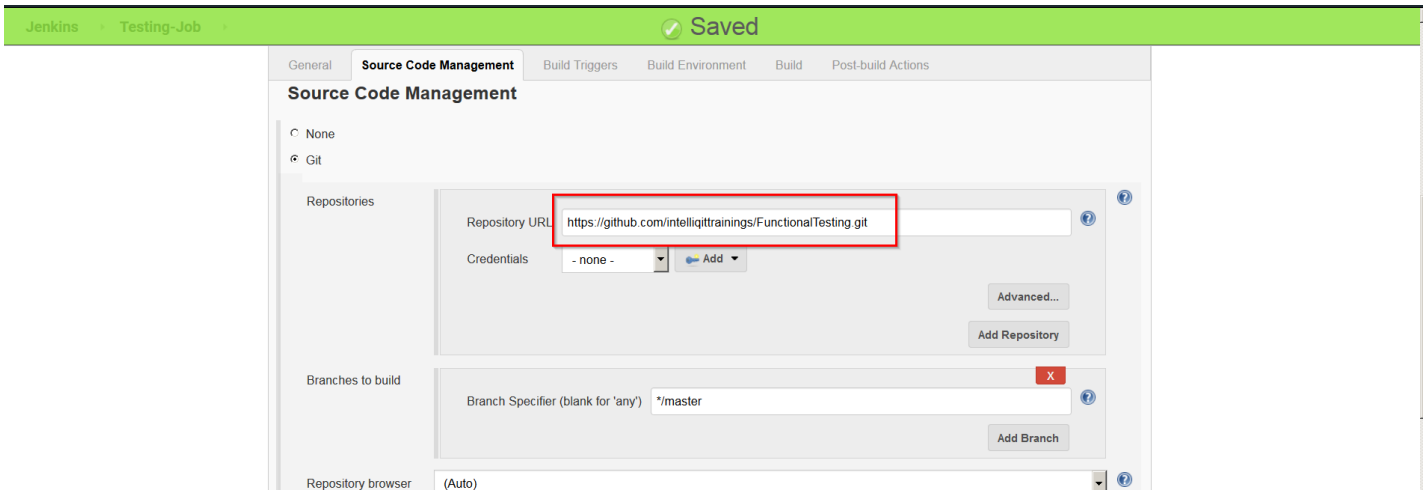
Open the dashboard of Jenkins, click on New item--->Enter item name as Testing

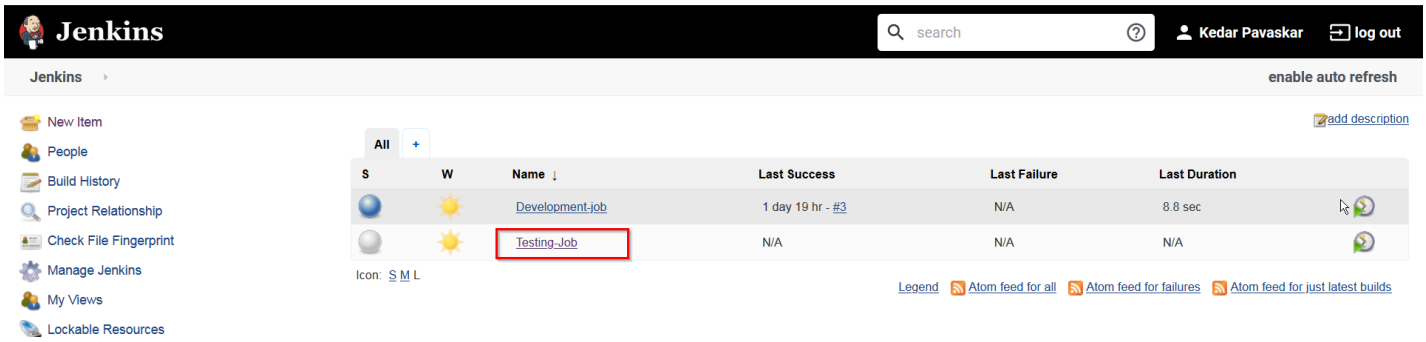




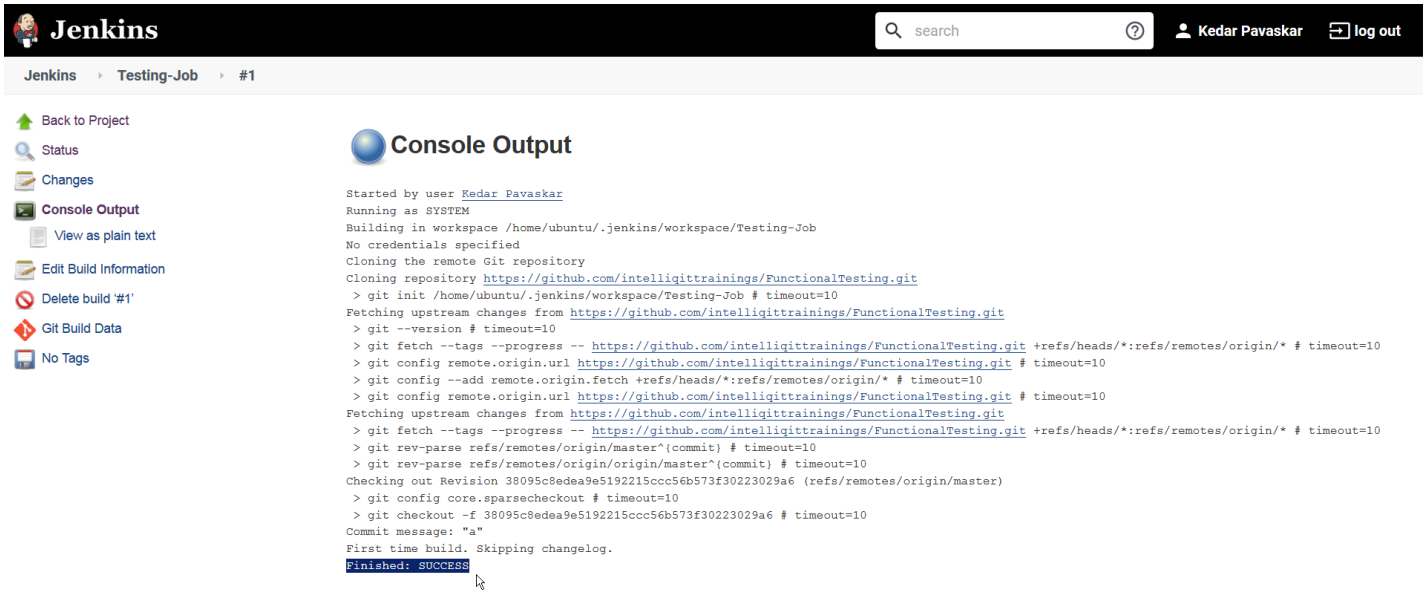
https://github.com/intelliqittrainings/FunctionalTesting.git

Go to Source Code Management, click on Git and Enter the GitHub url where the Testers have uploaded the selenium scripts. In our case it's the above GitHub repository.

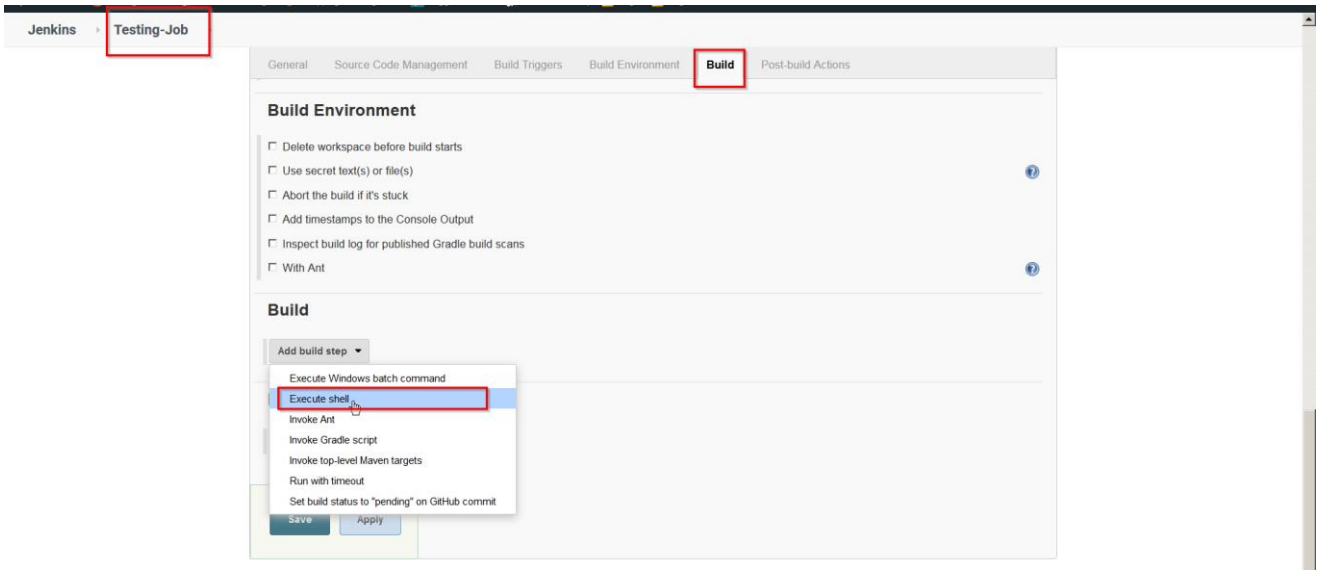




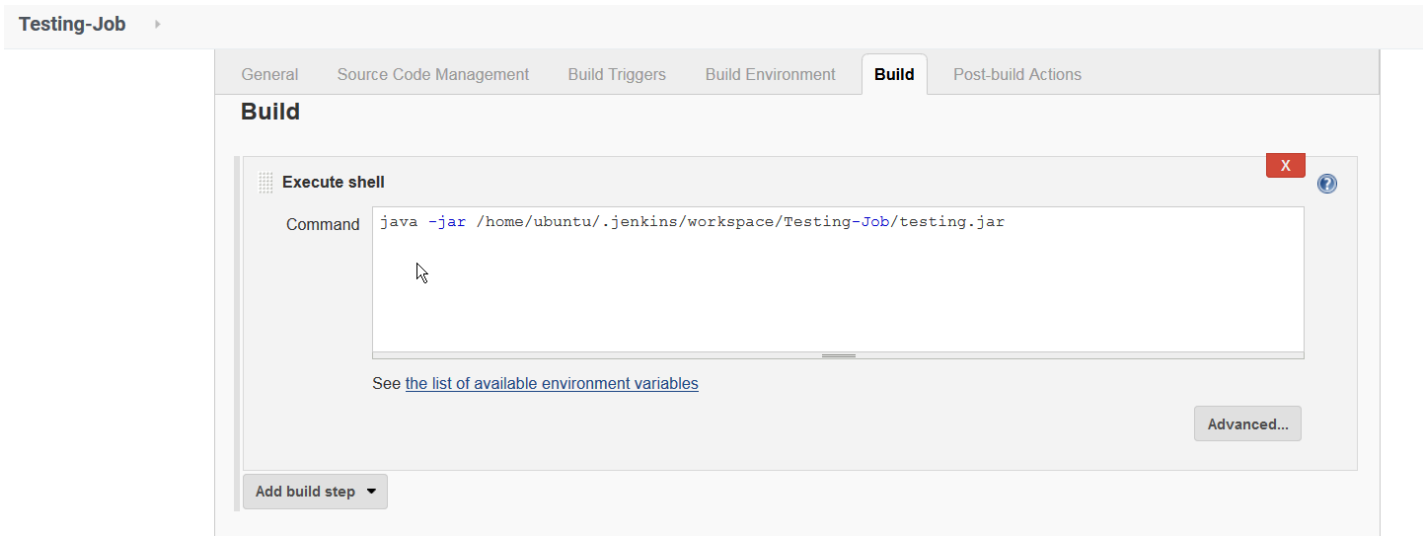
Click build



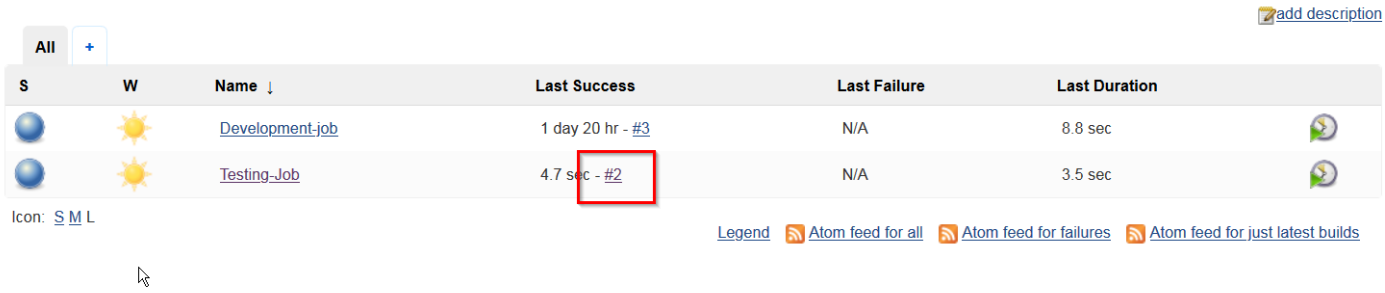
This console output shows that the Testing selenium script has been downloaded to the jenkins server from the GIT repository.

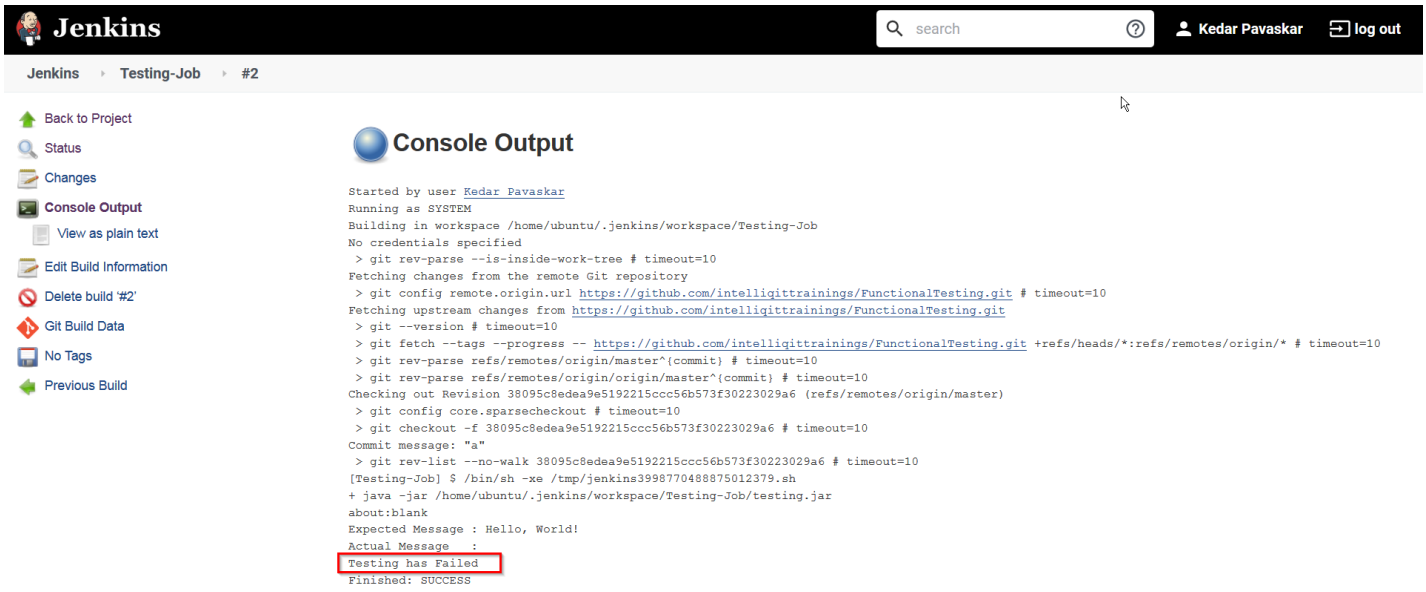


Go to Build section, click on Add build step, Click on Execute shell



Click on Apply--->Save, and run the Testing job again. This time it will execute the jar file (selenium test scripts).

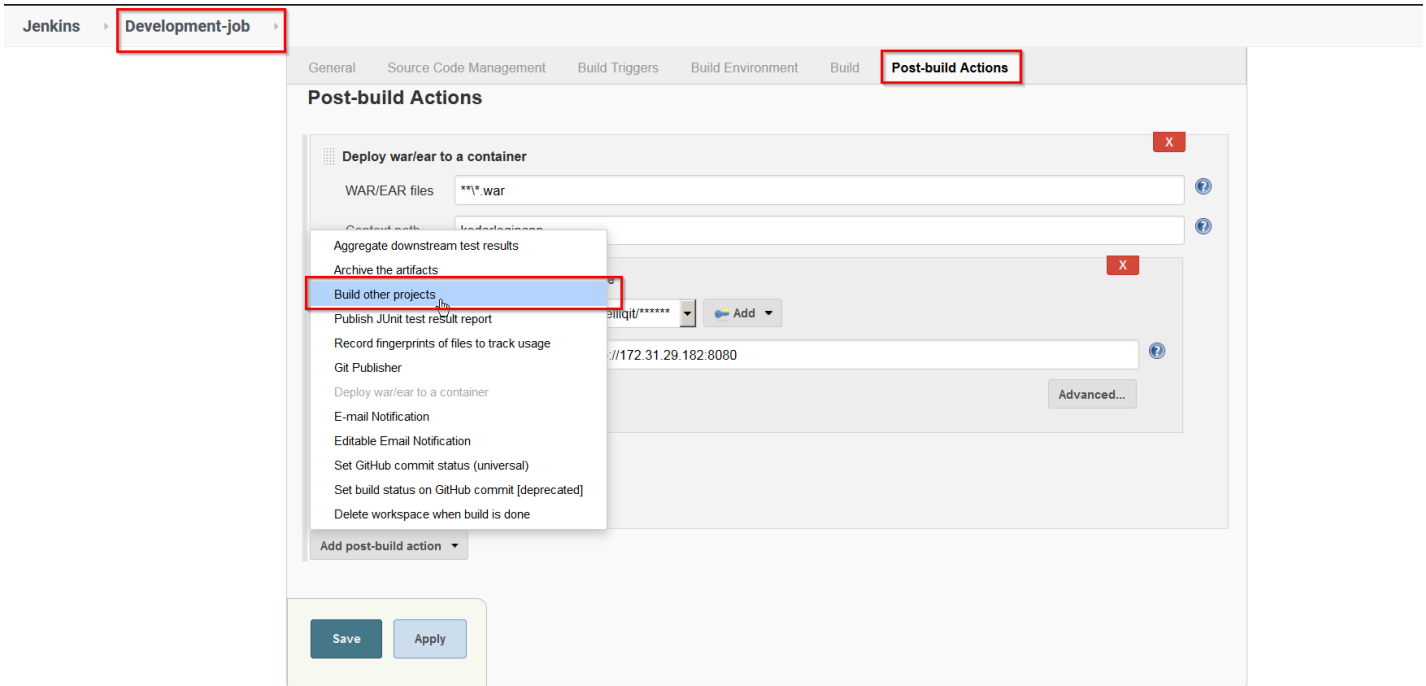


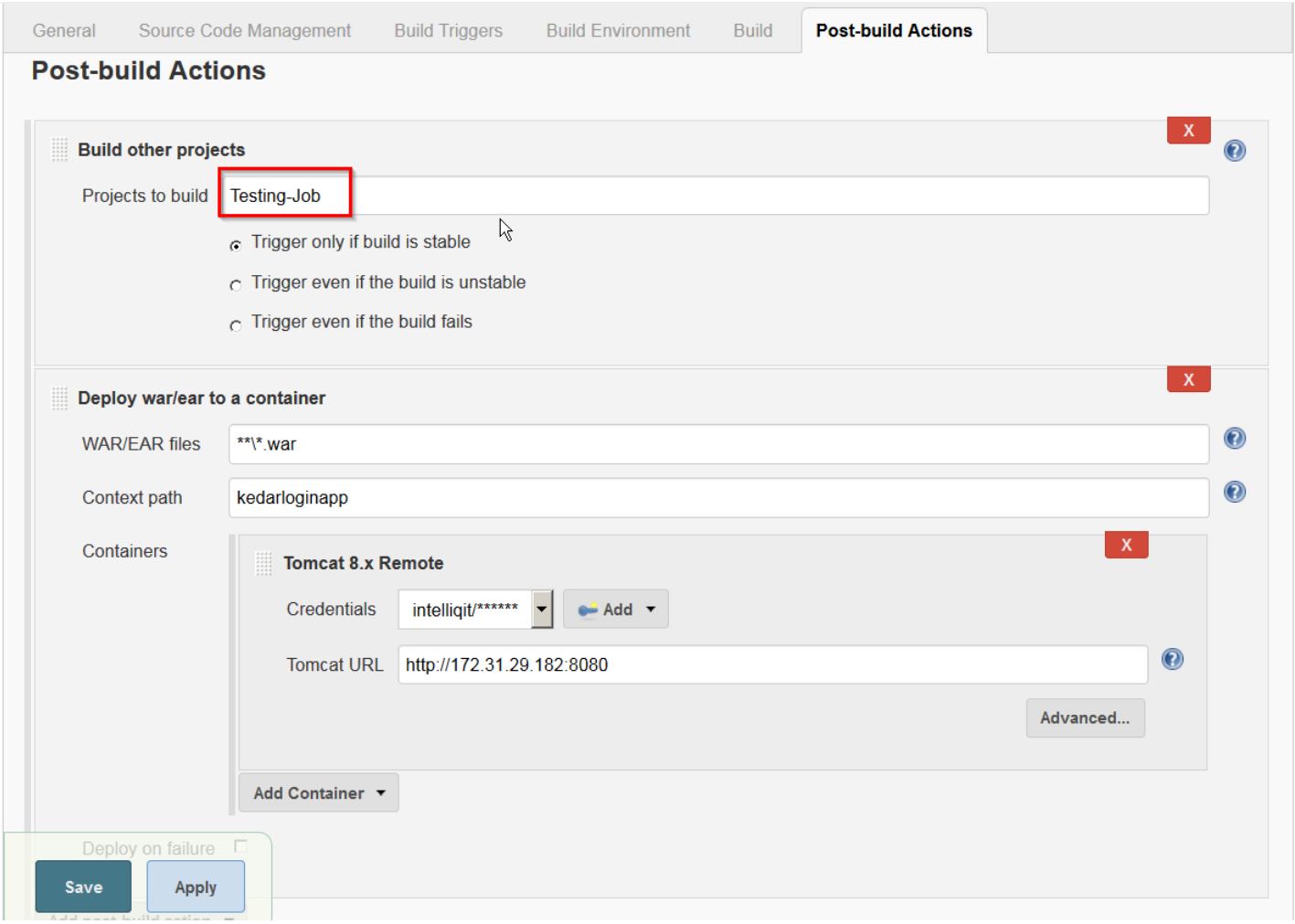


The test job though runs, will not actually test the app, as there is no Link between the two Jobs.

Linking Development job with the Testing job

So far we have manually scheduled the development-job and testing Jobs, but we need to schedule the jobs in such a way that the Jobs are linked in a serial manner to perform the CI/CD stages accordingly.





Mentioned “testing-job” to link the jobs.

Now Running the Development Job will also run the Testing Job in queue.

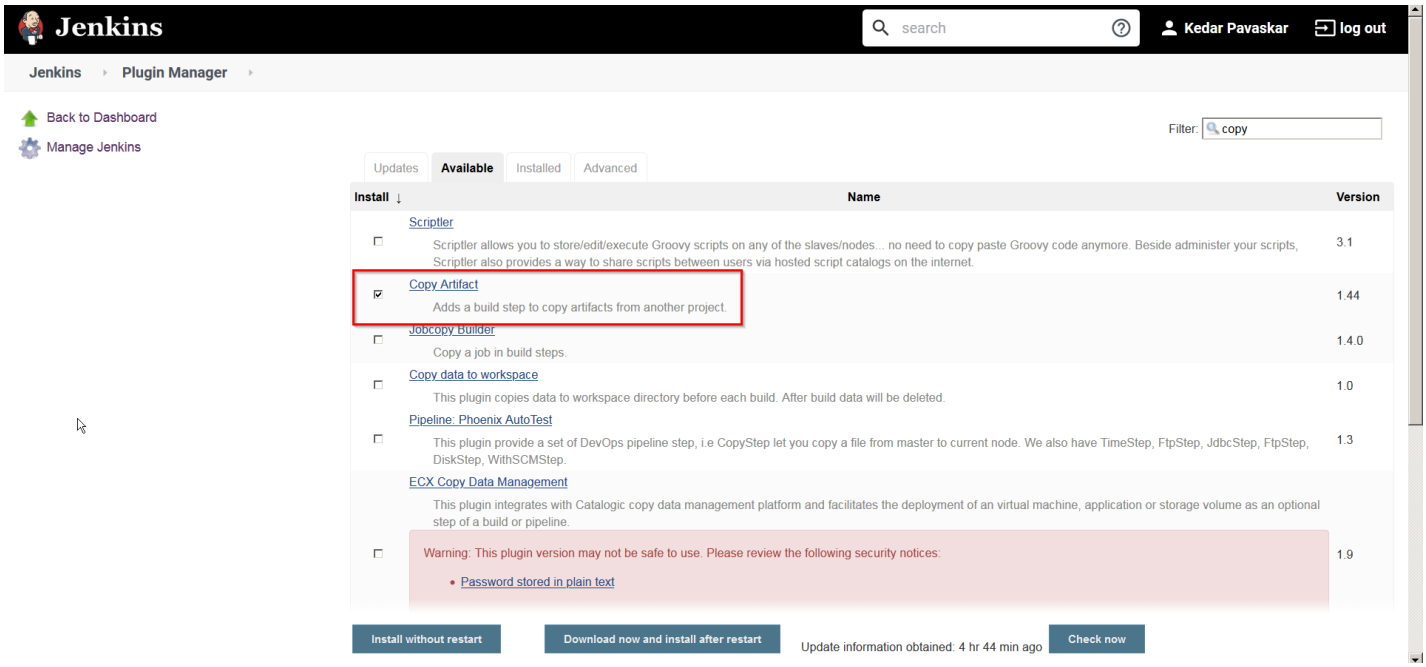
Workspace created by Jenkins Jobs.

We have 2 jobs in the above CI/CD example, Development-Job and Testing-Job. Both the above jobs are creating a workspace (paths mentioned below) where the code and the artifact is either getting downloaded or Created. The challenge here is to deploy the artifact from the Testing Job to production servers as the artifact is sitting inside the Development Workspace. To mitigate this, we need to copy the artifact from development workspace to testing.

Install the Copy Artifact Plugin on Jenkins.

/home/ubuntu/.jenkins/workspace/Development

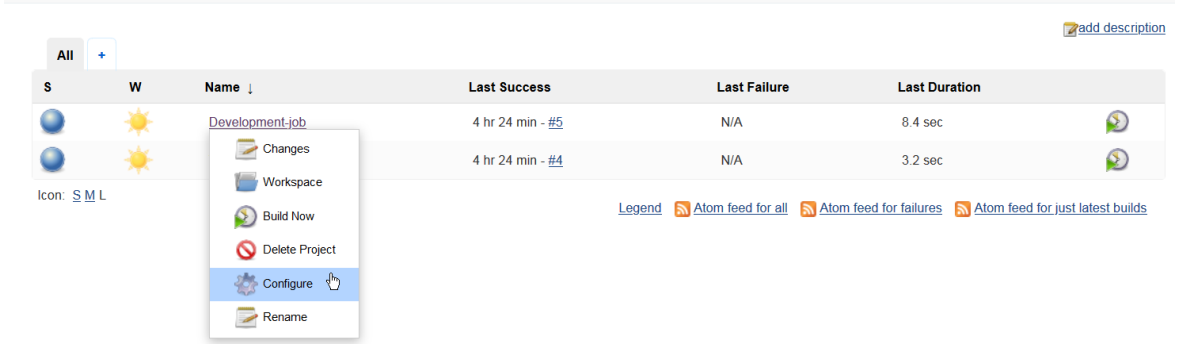
/home/ubuntu/.jenkins/workspace/Testing

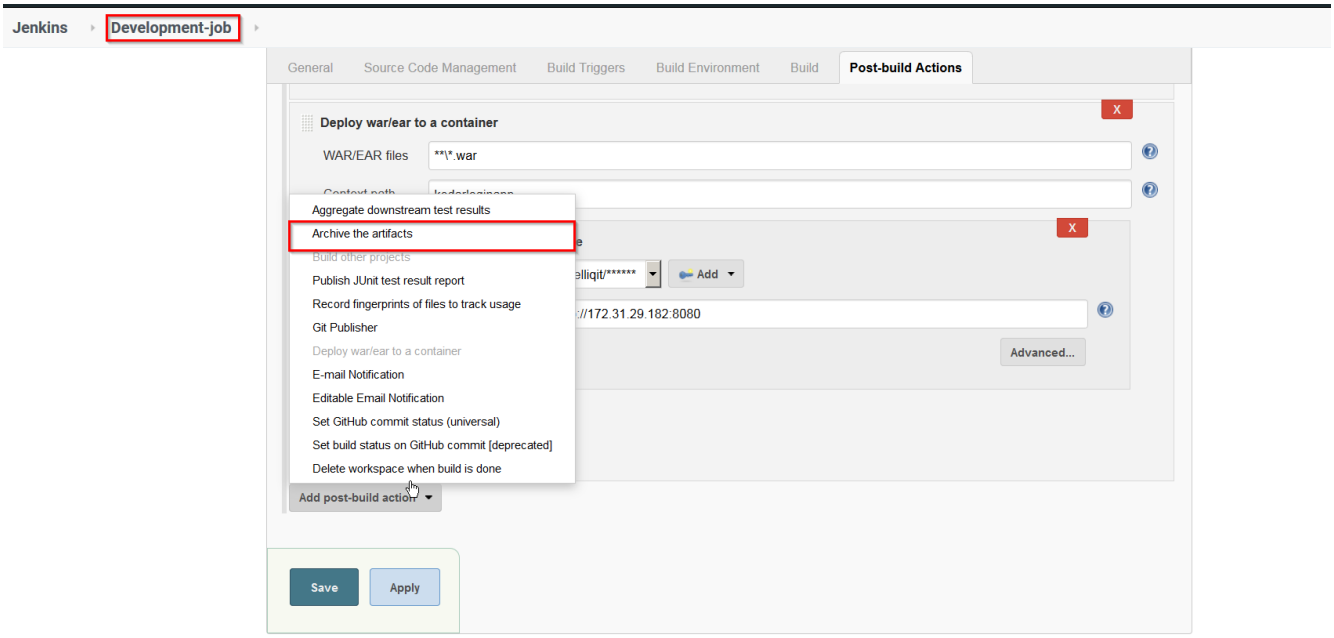


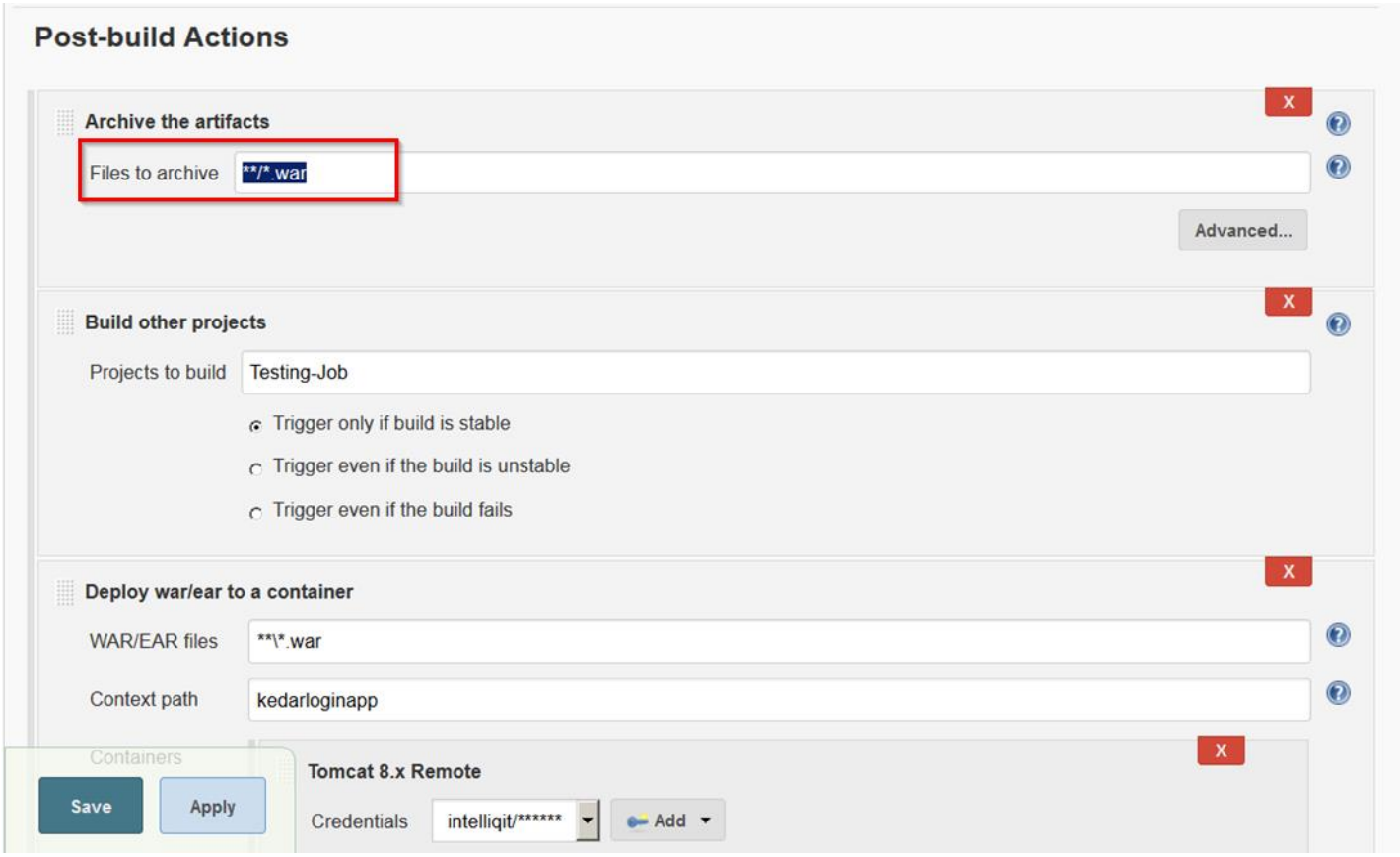
Once installed proceed to configure Jenkins, to copy the artifact to testing job.

Copying artifacts from Development job to Testing job

- Open the dashboard of Jenkins
- Click on Manage Jenkins--->Manage Plugins
- Go to Available section--->Search for Copy artifact plugin
- Click on Install without restart
- Go to the dashboard of Jenkins
- Go to the Development job--->Click on Configure
- Go to Post Build actions--->Click on Add Post build actions
- Click on Archive the artifacts
- enter files to be archived as: ***.war
- Click on Apply--->Save
- Go to the dashboard of Jenkins
- Go to the Testing job--->Click on configure
- Go to Build section--->Click on Copy artifacts from another project
- Enter the project name as Development
- Click on Apply-->Save

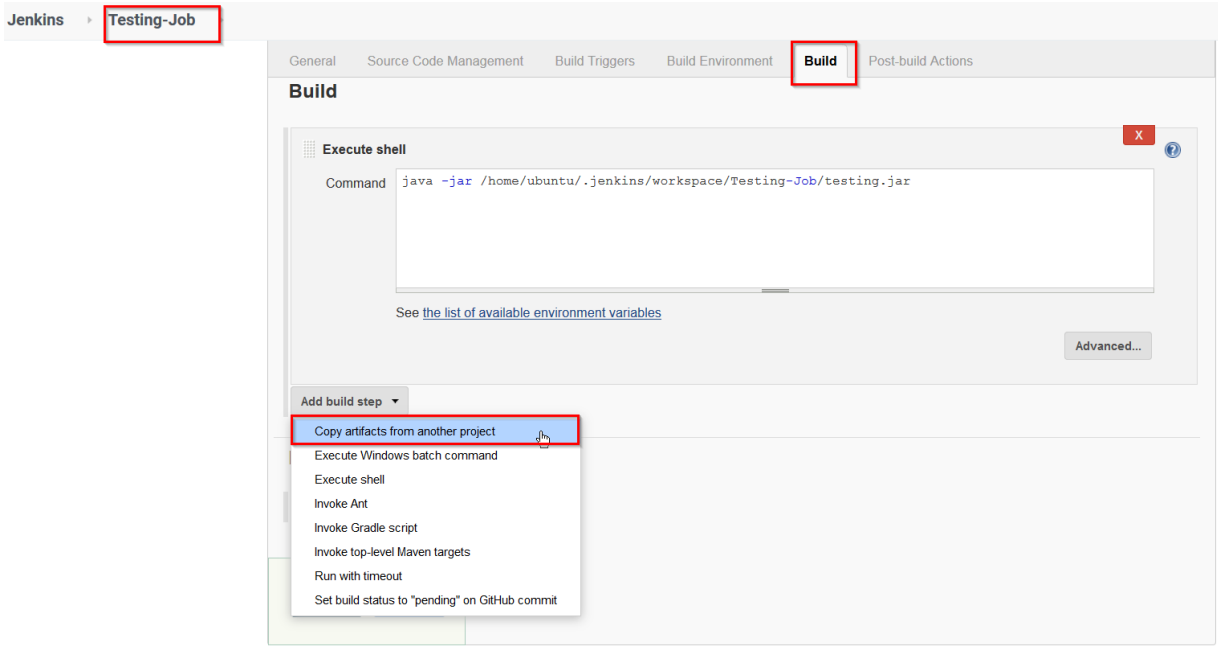


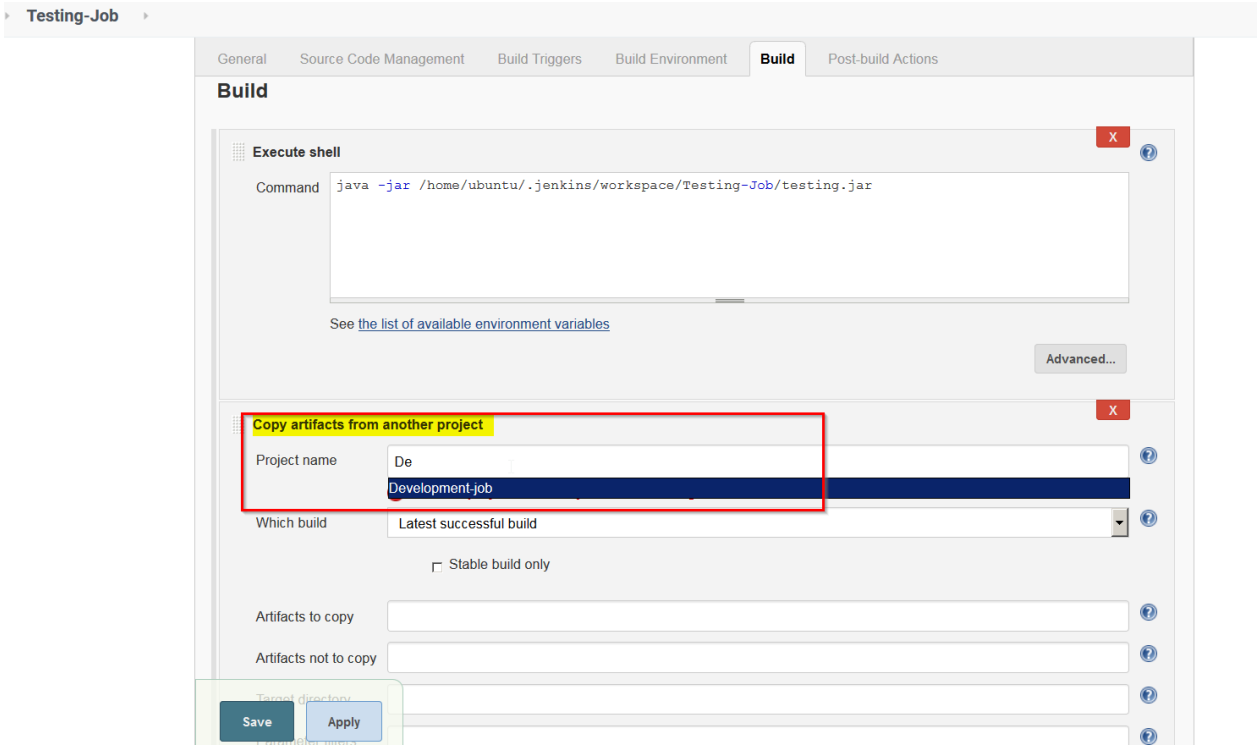




Above steps are performed on the development-job and we are instructing jenkins to archive (bundle) all the war files in the development workspace.

In the next step move to the Testing Job.



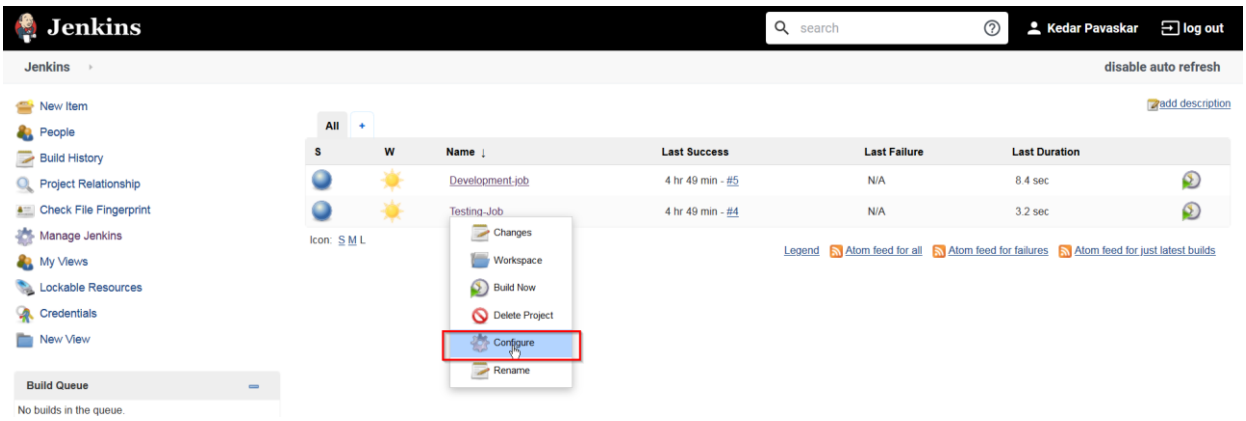


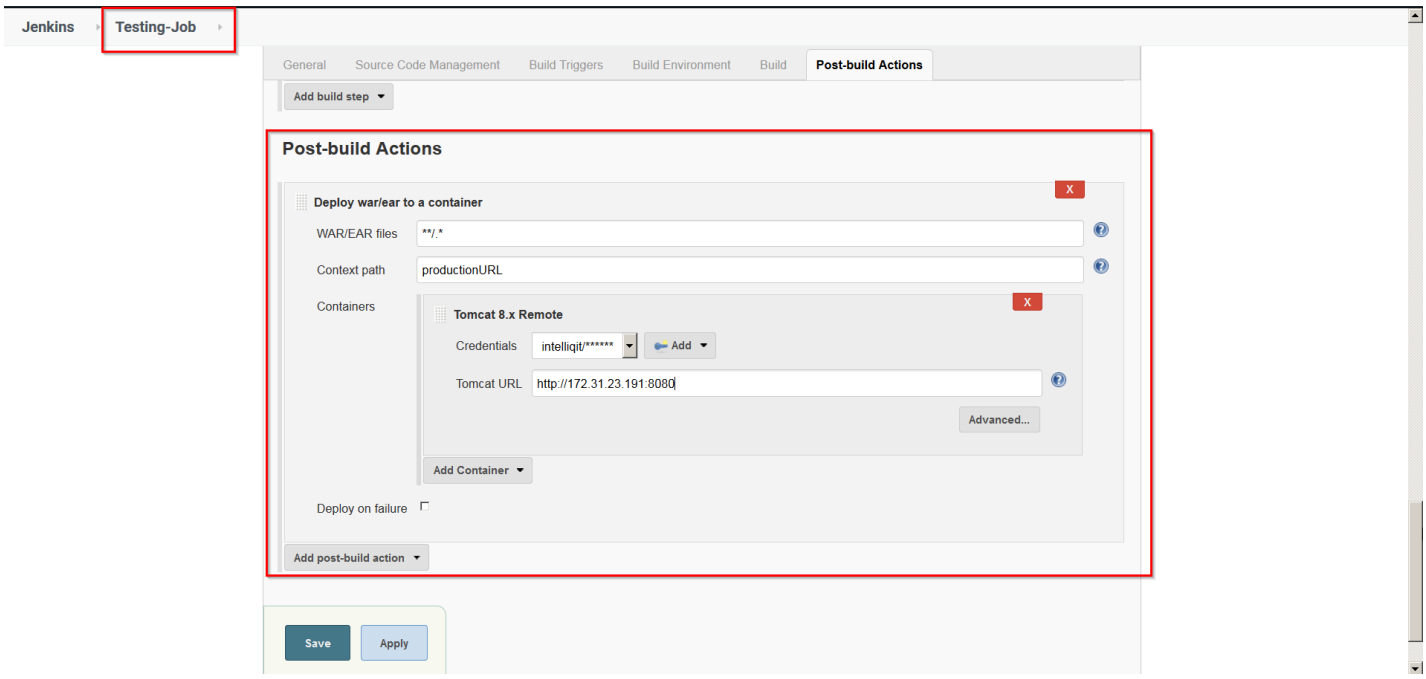
Copy the artifact from the development job, soon after you type development job name the dropdown lists the name.

Once the copy artifacts from one job to another has been completed, we'd need to setup config to deploy the artifact to the Production server (similar to Deploying the artifact into QA servers in Stage 3.

LAB testing Stage 5 - Continuous Testing configuration on jenkins

- Open the dashboard of Jenkins
- Go to the Testing job--->Click on Configure
- Go to Post build actions--->Click on add post build action
- Click on Deploy war/ear to container
- Enter war/ear files: **/*.war
    - Context path: productionURL
    - Click on Add container
    - Select tomcat
    - Enter tomcat username and password
    - Tomcat url: privateip_of_prodserver:8080
- Apply--->Save





Save and goto the dashboard of Jenkins.

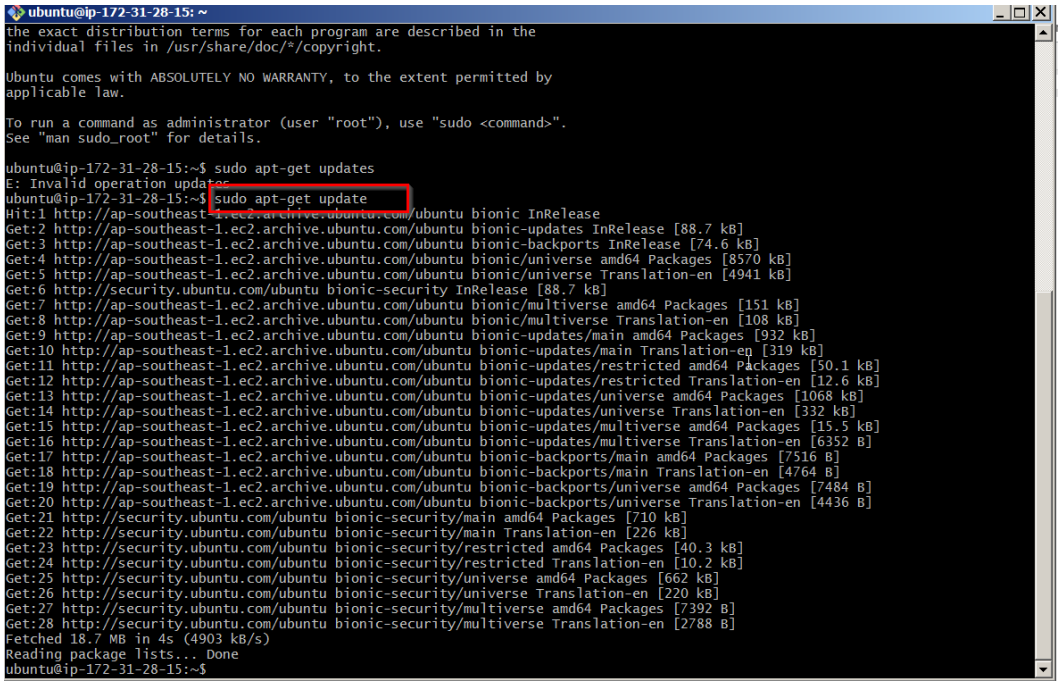
Now Running the Development Job Will Perform the First three stages of the CI/CD and the job will continue to call the Testing Job and perform the remaining stages of CI/CD.

Alternative (permanent) Methods of Jenkins Installation.

Setting up Jenkins as a permanent installation

1. Update the apt repository

sudo apt-get update



2. Install java

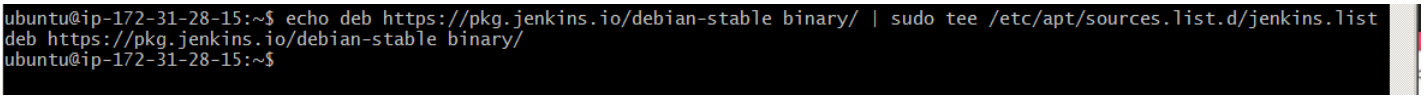
sudo apt-get install -y openjdk-8-jdk

3. Add the repository key to the system

wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -

4. Add the debain package address to sources list

echo deb https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list



5. Update the apt repository

sudo apt-get update

6. Install jenkins
   sudo apt-get install -y jenkins

7. Install git and maven
   sudo apt-get install -y git maven

8. To access jenkins

   Launch any browser

   public_ip_of_jenkinsserver:8080

Setup of Jenkins using Tomcat

1. Update the apt repository

sudo apt-get update

2. Install tomcat

sudo apt-get install -y tomcat8

3. Install tomcat8-admin

sudo apt-get install -y tomcat8-admin

4. Download jenkins.war

wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war

5. Give execute permissions on tomcat8 for others

sudo chmod o+w -R /var/lib/tomcat8

6. Copy jenkins.war into tomcat8

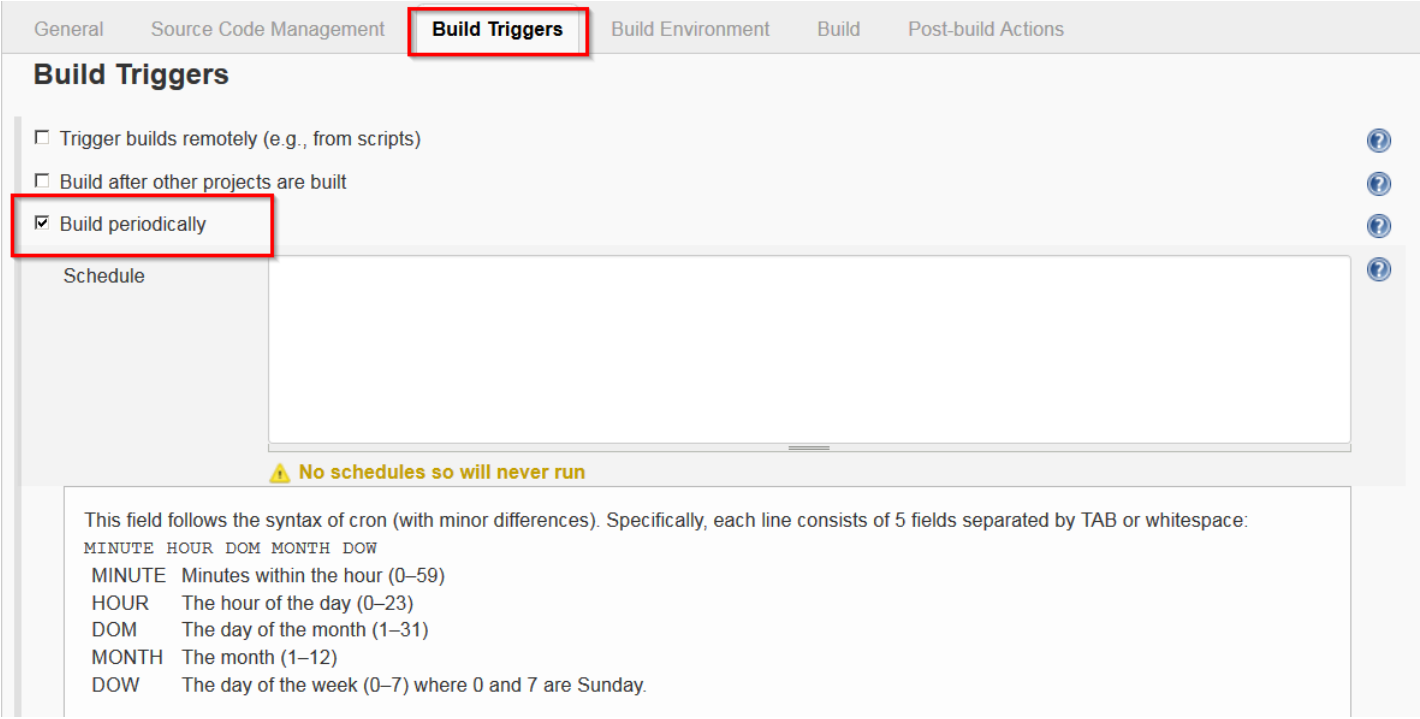
cp jenkins.war /var/lib/tomcat8/webapps

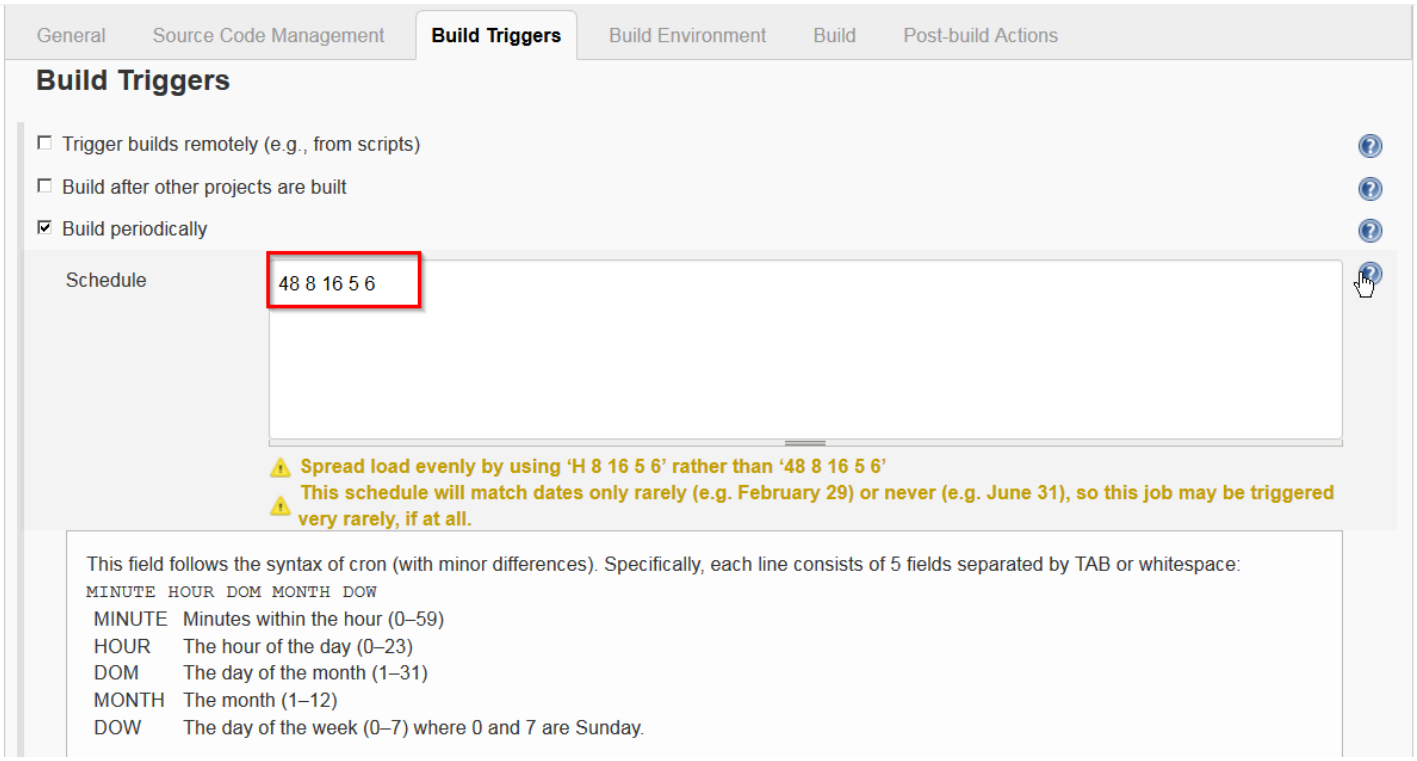
7. To access jenkins from browser

public_ip_of_tomcat:8080/jenkins

Scheduling the Jenkins job to execute at a specific time

1. Open the dashboard of Jenkins
2. Go to the job that we want to schedule
3. Click on Configure
4. Go to Build triggers--->Click on Build Periodically
5. Schedule the date and time

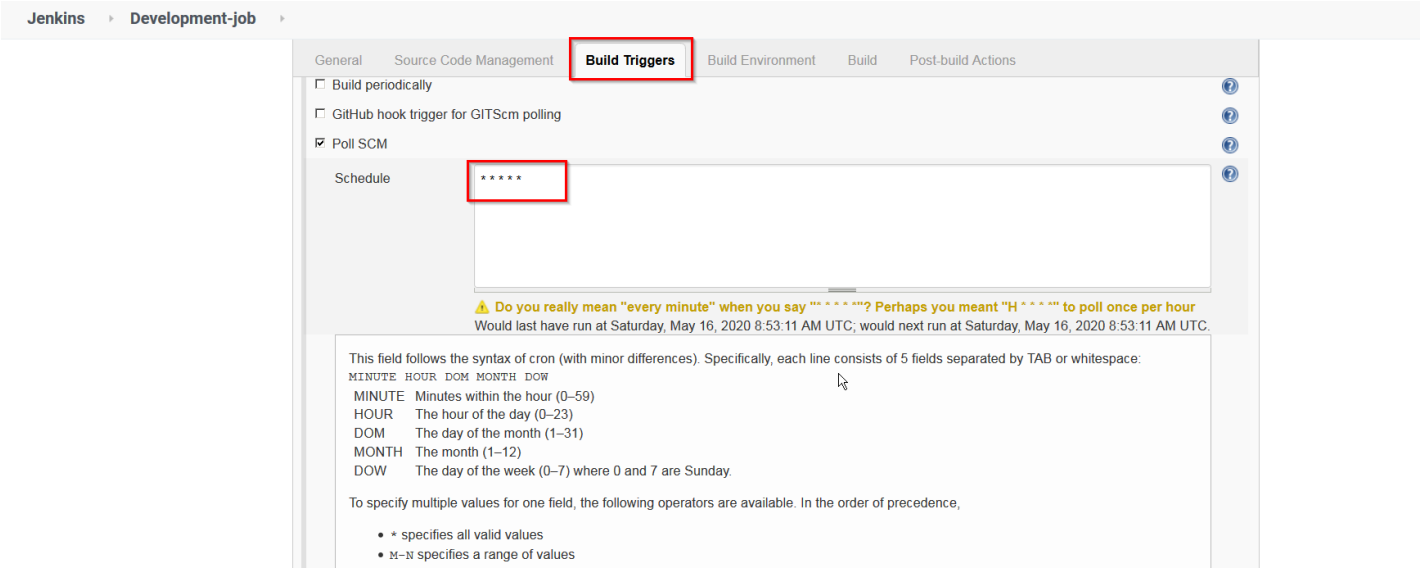




So here in the above example, the job will automatically run on 48th minute, 8th hour of the day, on 16th day of the 5th month, and 6th day of the week. Basically at 8:48am on 16th may Saturday.

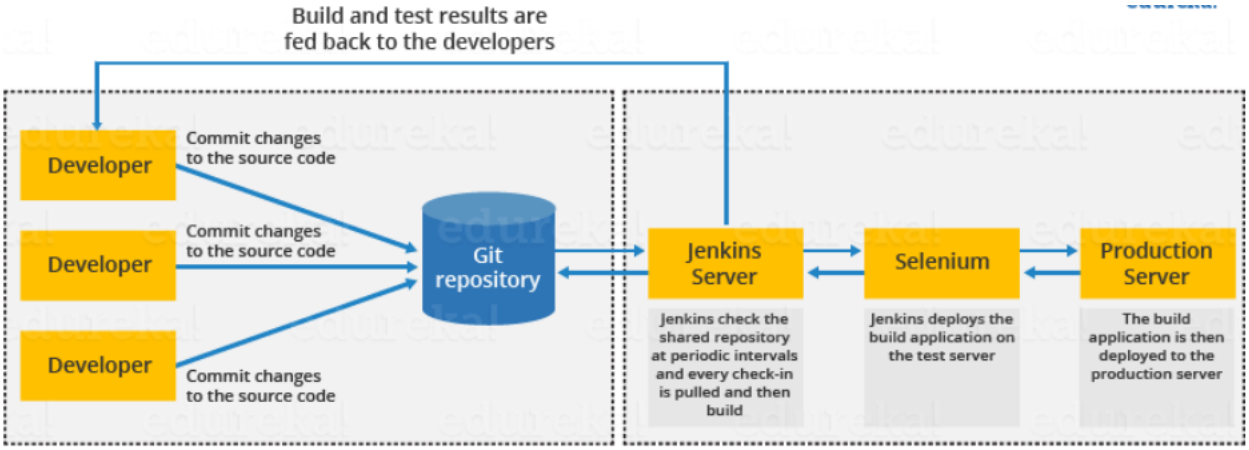
Scheduling the jenkins job to trigger when developers make any new commits to git

1. Open the dashboard of Jenkins
2. Go to the job that we want to schedule
3. Click on Configure
4. Go to Build triggers
5. Click on Poll SCM
6. Schedule: * * * * *
7. Apply→Save



The above generic expression * * * * * means that jenkins will constantly keep polling every minute, every hour, day month, the remote GIT repository to see if the developers have uploaded any new code.

Jenkins architecture



In the above scenario, sometimes a single jenkins server is not able to meet the Requirements like:

- When you have the need for several different environments to test your builds, this cannot be done by single jenkins server.
- If larger and heavier projects get built on the single jenkins server, it may not be able to handle it. As sometimes, running jobs simultaneously can hand the jenkins server vm

Jenkins Distributed Architecture

Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

Jenkins Master

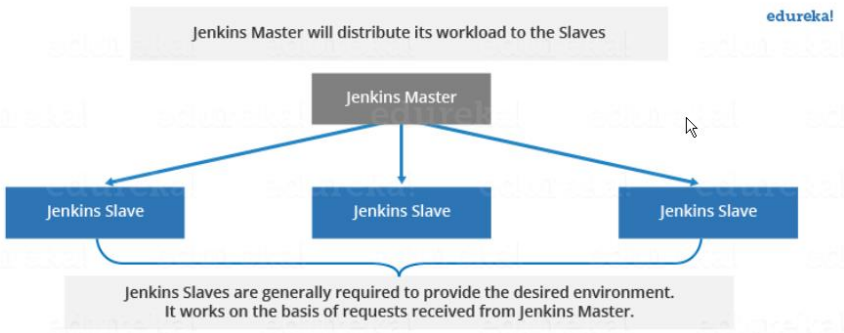
Your main Jenkins server is the Master. The Master's job is to handle:

- Schedule Build Jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the salves, possibly taking them Online and offline.
- Recording and presenting the build results.
- A master instance of jenkins can also execute build jobs directly.

Jenkins Slave

Jenkins slave is a java executable (Jar file) that runs on a remote machine. Its characteristics are:

- It here's requests from the jenkins master instance.
- Slaves can be run on a variety of OS's
- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- You can configure a Project to always run on a particular Slave machine, or simple let jenkins choose the slave machine.

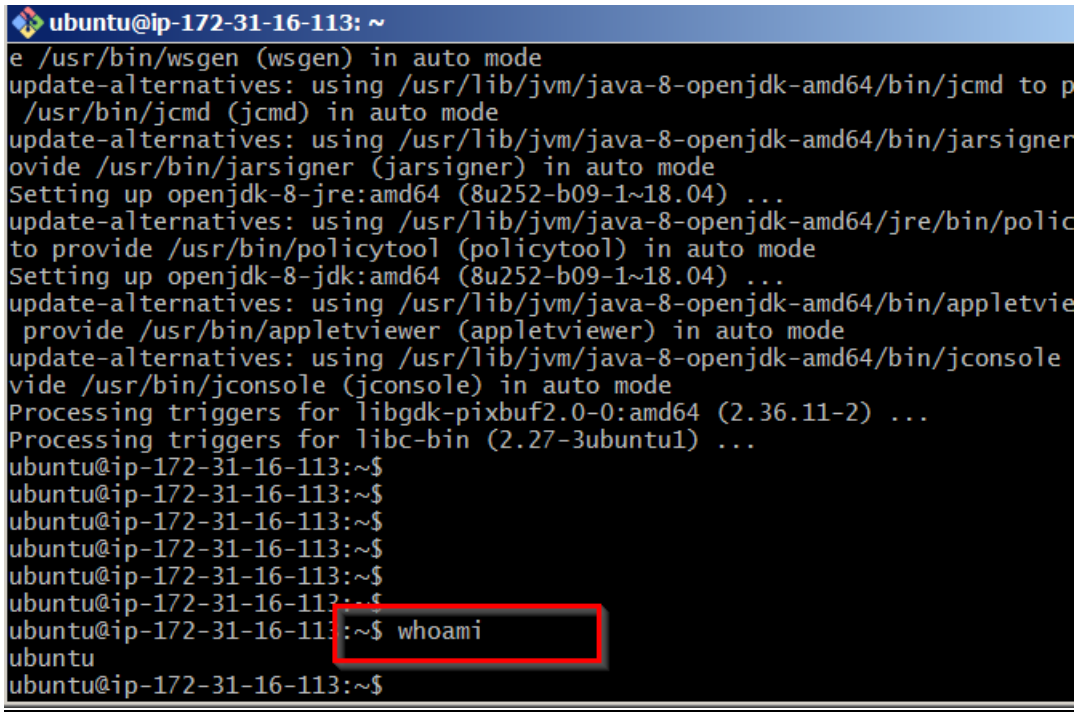


Pre-requisites for Master-Slave

- Same java version needs to be running on both the VM's
- Slave will only work when Master is available, if master Vm is not available, slave will not function.

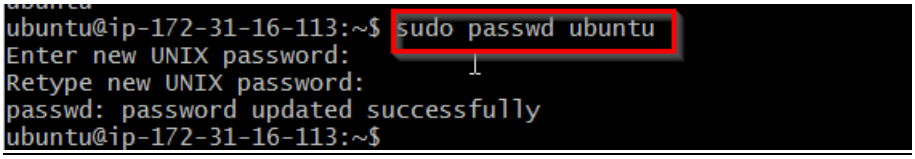
LAB Setting up passwordless SSH in Master slave (or between any two Linux Vms)

1) After setting the Up the slave VM, login and check what is the user name as mentioned below with the ***whoami*** command



2) Setup the password for the default user

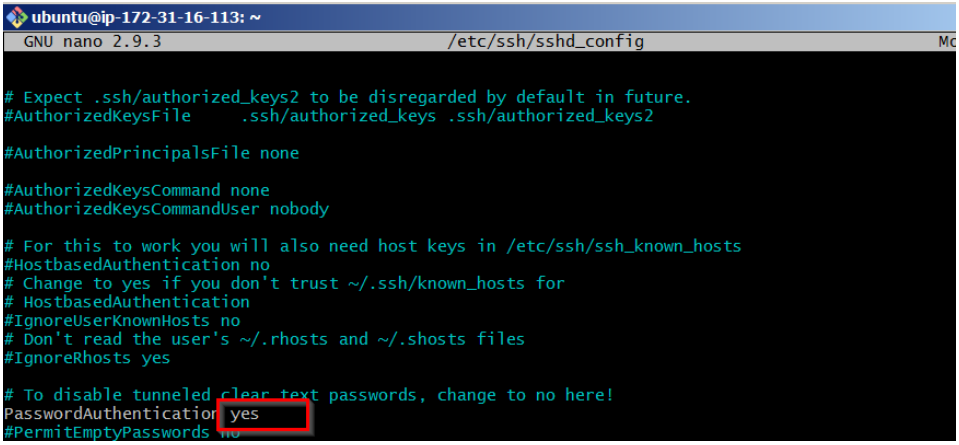
sudo passwd ubuntu



3) Edit the sshd_config file

sudo vim /etc/ssh/sshd_config

Search for "PasswordAuthentication" and change it to no to yes



4) Restart ssh Service

sudo service ssh restart

5) Connect to master VM with Gitbash/putty and generate the sshkeys

```
ubuntu@ip-172-31-18-83:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa): kedarsshkeys
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in kedarsshkeys.
Your public key has been saved in kedarsshkeys.pub.
The key fingerprint is:
SHA256:OAJvx3v4xrrWFGjyZEsE+vcf91EVMzr2a9Y4BmLPhJc ubuntu@ip-172-31-18-83
The key's randomart image is:
+---[RSA 2048]----+
|     . ..       + |
|      + o.      . +|
|     . =.+.    +  .|
|      o.+*o....o .|
|       .BS.=.E  ..|
|        .o*.* . .+|
|          o= + ==.|
|         .o.. +oo.|
|        .o.   .   .|
+----[SHA256]-----+
ubuntu@ip-172-31-18-83:~/.ssh$ cd .ssh/
-bash: cd: .ssh/: No such file or directory
ubuntu@ip-172-31-18-83:~/.ssh$ ls
authorized_keys  id_rsa  id_rsa.pub  known_hosts
ubuntu@ip-172-31-18-83:~/.ssh$ ssh-copy-id ubuntu@172.31.16.113
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ubuntu/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ubuntu@172.31.16.113's password:
```

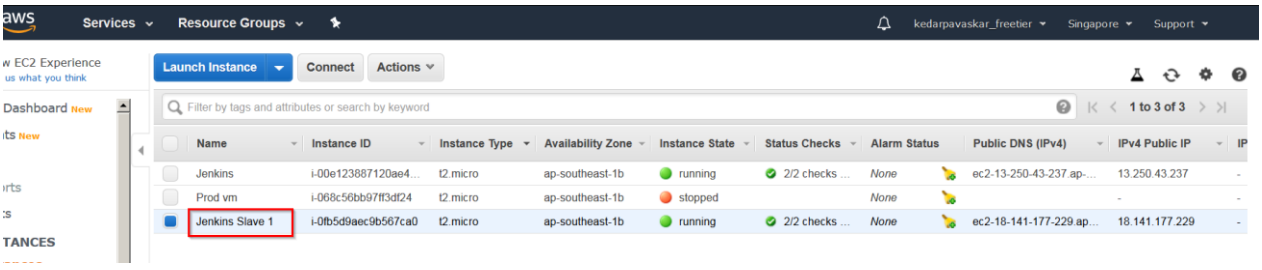
The newly generated keys will be listed under .ssh

```
ubuntu@ip-172-31-18-83:~/.ssh$ ssh-copy-id ubuntu@172.31.16.113
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ubuntu/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ubuntu@172.31.16.113's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ubuntu@172.31.16.113'"
and check to make sure that only the key(s) you wanted were added.
```

6) Now the password less SSH connection between two Linux hosts has been implemented.

LAB Setup of Master –Slave Jenkins VM

1) Create a new AWS ubuntu instance--->Name it JenkinsSlave1



2) Install the same version of java that is present on master

sudo apt-get update

sudo apt-get install -y openjdk-8-jdk

3) Download the slave.jar

wget http://private_ip_jenkinsserver:8080/jnlpJars/slave.jar

```
ubuntu@ip-172-31-16-113:~$ wget http://172.31.18.83:8080/jnlpJars/slave.jar
--2020-05-16 12:45:46--  http://172.31.18.83:8080/jnlpJars/slave.jar
Connecting to 172.31.18.83:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1522914 (1.5M) [application/java-archive]
Saving to: 'slave.jar'

slave.jar                 100%[=====================================>]   1.45M  --.-KB/s    in 0.02s

2020-05-16 12:45:46 (76.3 MB/s) - 'slave.jar' saved [1522914/1522914]

ubuntu@ip-172-31-16-113:~$
```

4) After downloading the slave.jar file on the slave Vm, you can see that the jar file only has read and write permissions for the owner, we need to change that. To do that, Give execute permissions on slave.jar
chmod u+x slave.jar

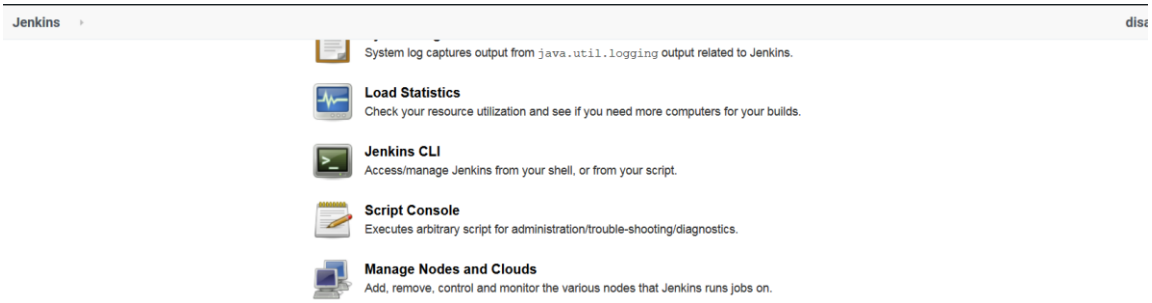
```
ubuntu@ip-172-31-16-113:~$ ls -l
total 1488
-rw-rw-r-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$ chmod u+x slave.jar
ubuntu@ip-172-31-16-113:~$
```

Before updating the permissions

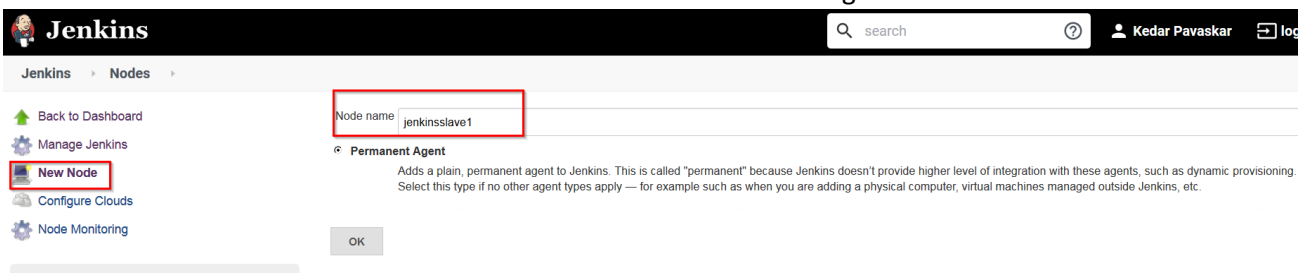
```
-rw-rw-r-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$ chmod u+x slave.jar
ubuntu@ip-172-31-16-113:~$ ls -l
total 1488
-rwxrw-r-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$
```

After updating the permissions

5) Create a folder that will act as a workspace
mkdir kedarsworkspace
6) Open the dashboard of Jenkins and Click on Manage Jenkins -> Click on Manage nodes and clouds



7) 9 Click on New node--->Enter some node name--->Select Permanent Agent

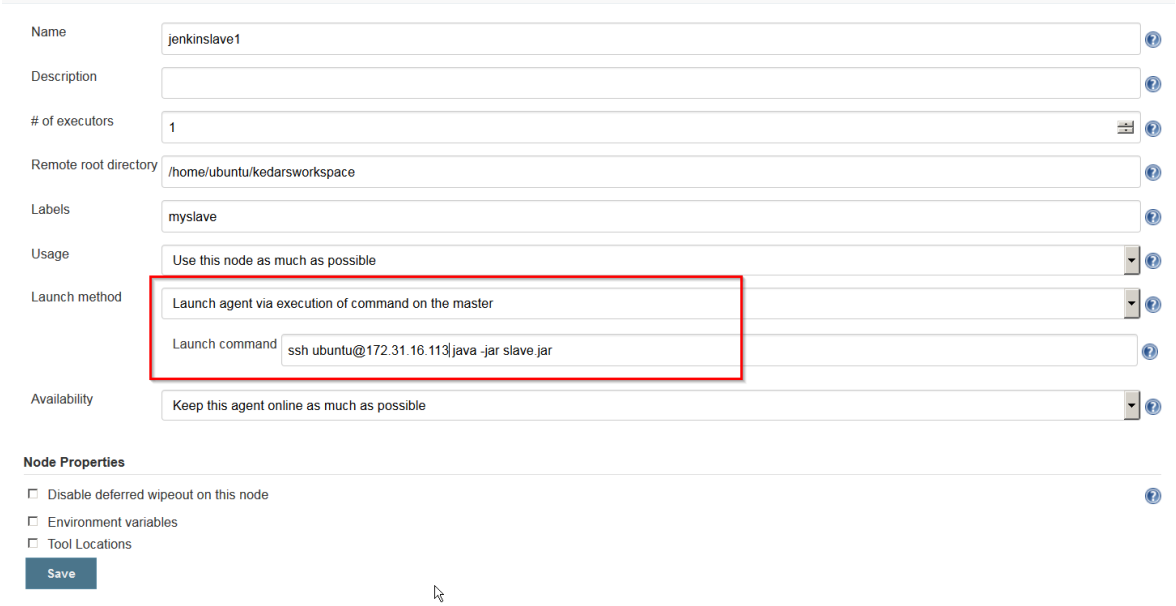


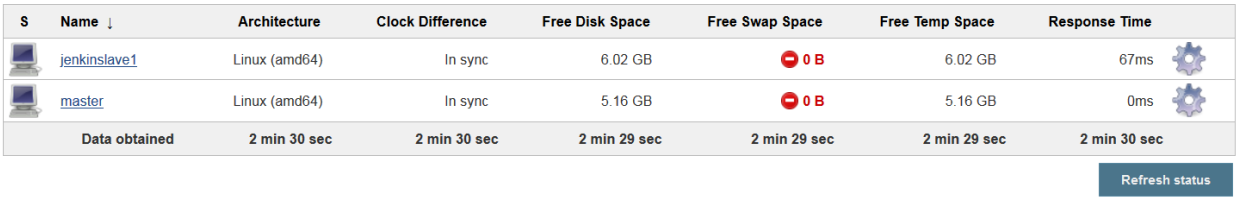
8) Remote root directory: /home/ubuntu/kedarsworkspace

9) Labels: myslave
(Acts as an alias)

Note: on the slave VM, note the #of Executors, this means that at any point of time jenkins will only run one job at a time.

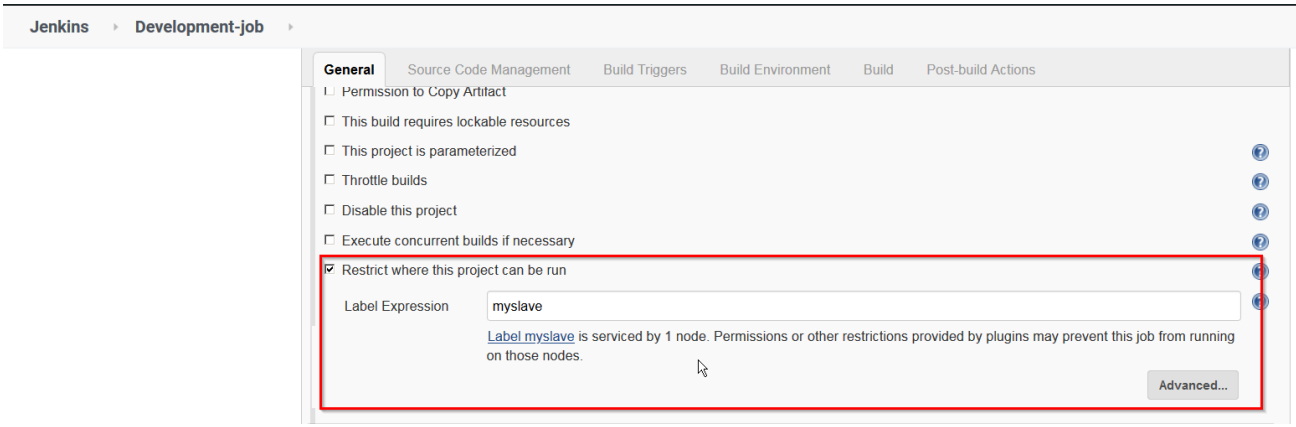
10) Launch Method: Select Launch agant via execution of command on master
Launch command: ssh ubuntu@private_ip_of_slave java -jar slave.jar





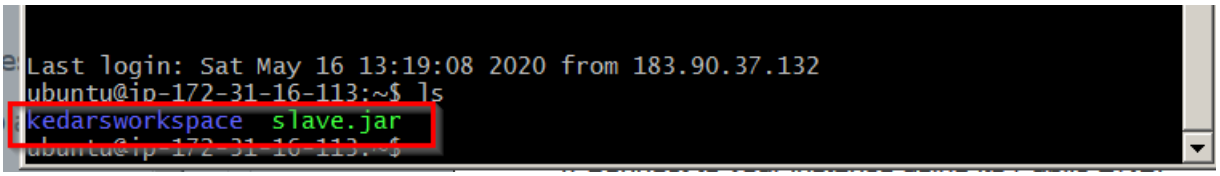
11) Go to the dashboard of Jenkins and Select the job that we want to run on slave--->Click on Configure

12) Go to General section, click on Restrict where this project can be run



13) Enter slave label: myslave
14) 18 Apply--->Save
15) Now run the Job you just edited to be run on slave. You will notice the below.

16) Navigate on the Gitbash/putty of the slave VM to verify if the job and the workspace is created.



Creating users in Jenkins

==========

- Open the dashboard of jenkins
- click on manage jenkins
- click on manage users
- click on create users
- enter user credentials

Creating roles and assigning

==========

- Open the dashboard of jenkins
- click on manage jenkins
- click on manage plugins
- click on role based authorization strategy plugin
- install it
- go to dashboard-->manage jenkins
- click on configure global security
- check enable security checkbox
- go to authorization section-->click on role based strategy radio button
- apply-->save
- go to dashboard of jenkins
- click on manage jenkins
- click on manage and assign roles
- click on manage roles
- go to global roles and create a role "employee"
- for this employee in overall give read access and in view section give all access
- go to project roles-->Give the role as developer and patter as Dev.* (i.e. developer role can access only those jobs whose name start with Dev)
- similarly create another role as tester and assign the pattern as "Test.*"
- give all permissions to developers and tester
- apply—save
- click on assign roles
- go to global roles and add user1 and user2
- check user1 and user2 as employees
- go to item roles
- add user1 and user2
- check user1 as developer and user2 as tester
- apply-->save

If we login into jenkins as user1 we can access only the development

related jobs and user2 can access only the testing related jobs

Jenkins Pipeline

All the stages of CI/CD can be triggered from the level of groovy script based file which is called as jenkins file.

this Jenkins file is generally uploaded along with the application code in to the remote version controlling system(Git)

and this jenkins file will perform all the stages of CI/CD from the remote version controlling system.

Advantages:

- Since all the stages of pipeline are controlled from the level of script, it allows the team members to review and edit the jenkins file and also parallelly maintain multiple versions of jenkins file in the remote version controlling system
- Jenkins file can withstand planned and unplanned restarts of the jenkins master server and continue the execution of CI/CD.
- Pipeline scripts can perform all the stages of CI/CD with minimum number of plugins due to this reason they are much faster.
- Pipelines can be used for implementing all the real life challenges like handling failure scenarios and implementing conditional statements and loops extra.

Pipelines can be implemented in two ways.

1. Scripted Pipeline
2. Declarative Pipeline

1 Scripted Pipeline

2 Declarative Pipeline

Sample Syntax of Scripted Pipeline

```
node('master/slave')
{
   stage('Stage name in CI-CD')
   {
      Groovy script to implement this stage
   }
}
```

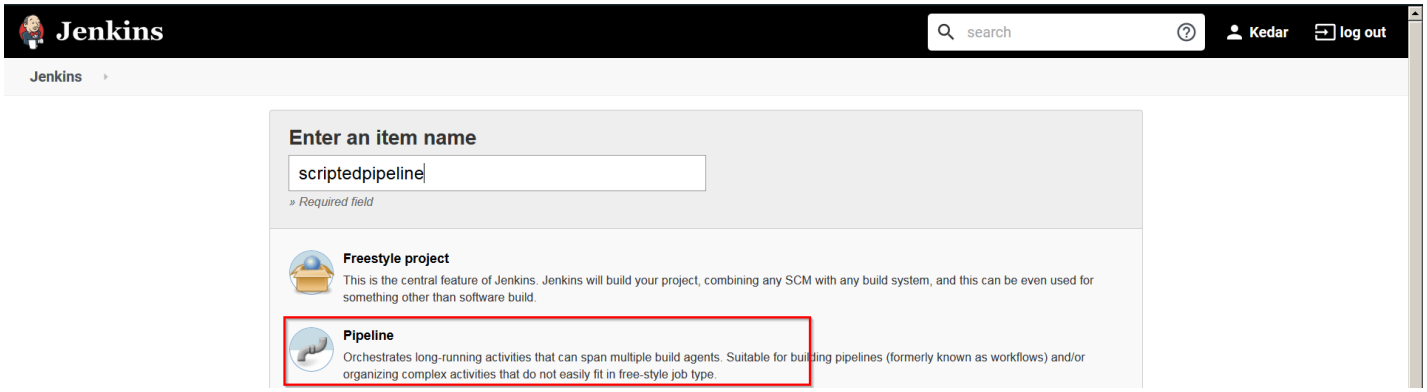
Sample - Declarative Pipeline

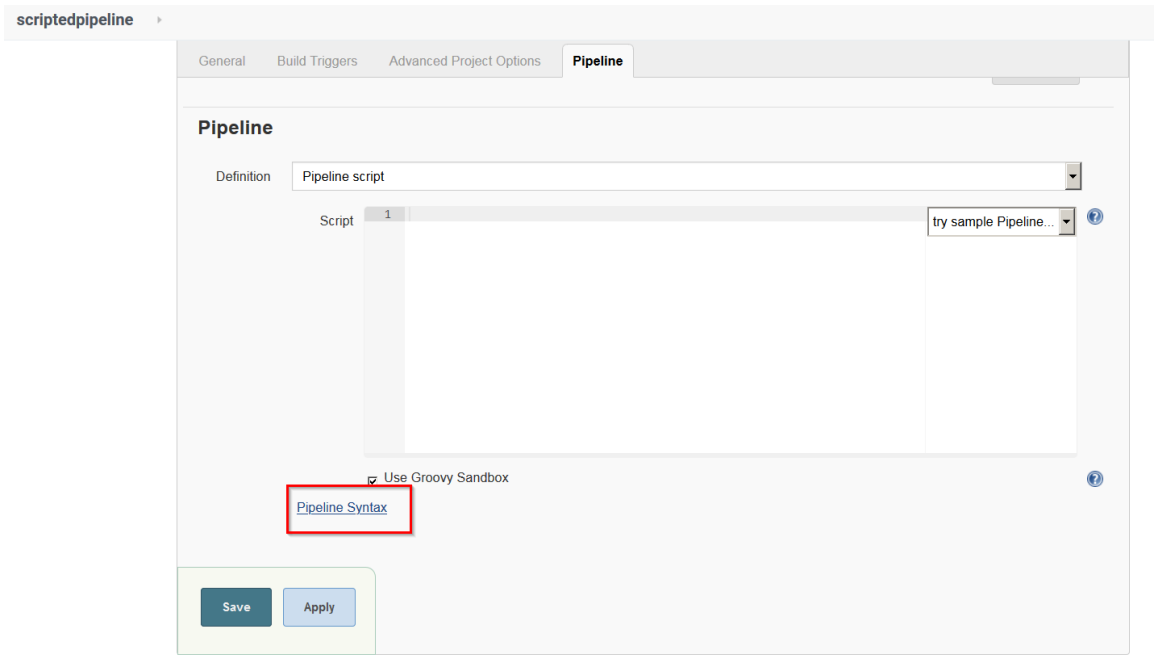
```
pipeline
{
 agent any
 stages
 {
   stage('Stage name in CI-Cd')
   {
     steps
     {
       Groovy script to implement this stage
     }
   }
 }
}
```

What is a Jenkinsfile?

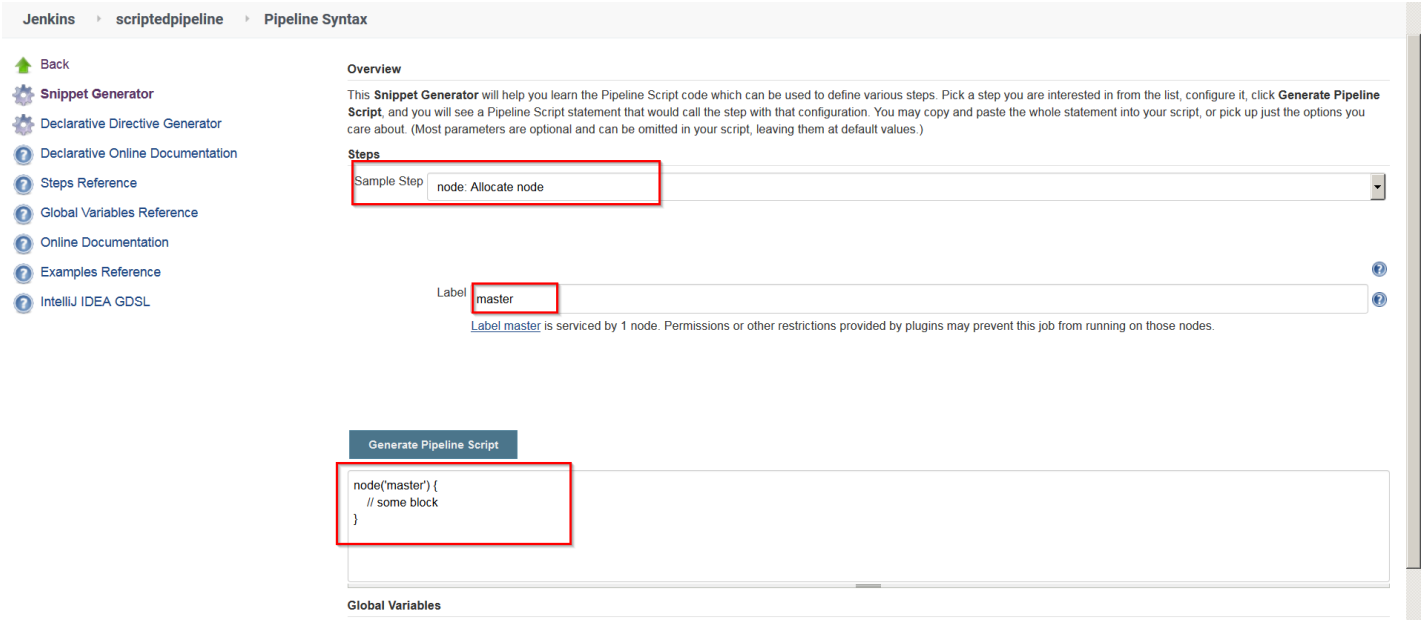
A Jenkins file is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system. How is this advantageous? This enables the developers to **access, edit and check the code at all times**.

LAB – setting up Scripted pipeline.





Pipeline syntax – As highlighted in the above Screenshot, is a built-in plugin to help us develop a groovy script for the pipeline.



Clicking on pipeline syntax, brings up the above screen wherein you can select the task you want to perform, and you will be provided the pipeline script for that predefined task, as shown above.

Similarly, select stage step as seen below and enter the stage name as “continuous Download” and the groovy script will be generated.

Eventually your first stage groovy script will look something like this.

Repeat the steps to build the second stage, however the continuous build stage code would be different. The code would look something like this.

Pipeline

Definition: Pipeline script

Script

```
node('master')
{
  stage('ContinuousDownload')
  {
    git 'https://github.com/selenium-saikrishna/maven.git'
  }
  stage('ContinuousBuild')
  {
    sh label: '', script: 'mvn package'
  }
}
```

Third stage for the jenkins CI/CD process would be Continuous Deployment. In our case we have a QA server to do the Test deployment. In the free style project, we manually configured jenkins to deploy the artifact to the QA server.

Here via the scripted pipeline, we are going to do the same via the groovy script.

Since we already have the artifact built via 2nd stage, we just need to copy the artifact to the QA server. To do this we just need to utilize the SCP (secure copy) command to copy the artifact from jenkins instance to QA server Instance.

- Initiate password-less authentication between Jenkins and QA server
- Copy the artifact via the SCP command (scp “source path” “ubuntu@privateIP: dest path”)
- Modify the permissions on the QA server Destination path, to be able to copy the artifact.

Pipeline

Definition: Pipeline script

Script

```
node('master')
{
  stage('ContinuousDownload')
  {
    git 'https://github.com/selenium-saikrishna/maven.git'
  }
  stage('ContinuousBuild')
  {
    sh label: '', script: 'mvn package'
  }
  stage('ContinuousDeployment')
  {
  sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.23.16
  }
}
```

Use Groovy Sandbox

Pipeline Syntax

```
Last login: Wed May 20 07:19:56 2020 from 172.31.28.246
ubuntu@ip-172-31-26-131:~$ sudo chmod o+w -R /var/lib/tomcat8/
ubuntu@ip-172-31-26-131:~$
```

The above permissions are to have write permissions on the path mentioned, recursively. This is required as the pipeline script would copy the artifact from the jenkins server to the Test Server.

Run the job, and you would see the pipeline steps being executed successfully as below.

Pipeline scriptedpipeline

Recent Changes

Stage View

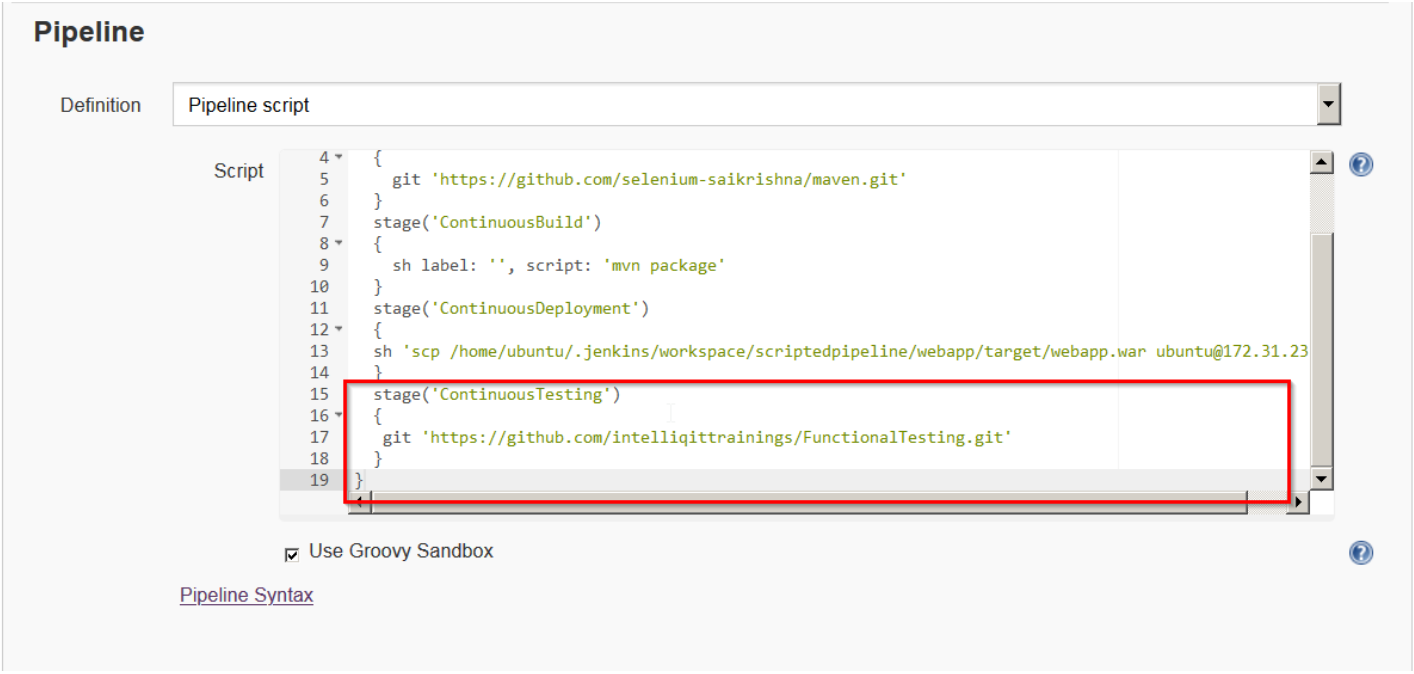
	ContinuousDownload	ContinuousBuild	ContinuousDeployment
Average stage times: (Average full run time: ~8s)	2s	20s	1min 5s
#6 May 24 23:03 No Changes	1s	7s	868ms

Continue the pipeline for the **continuousTesting** part.

In this example, the testing scripts are readily available in the below path. We just need to call the git repository path of the selenium script (testing script) as below.

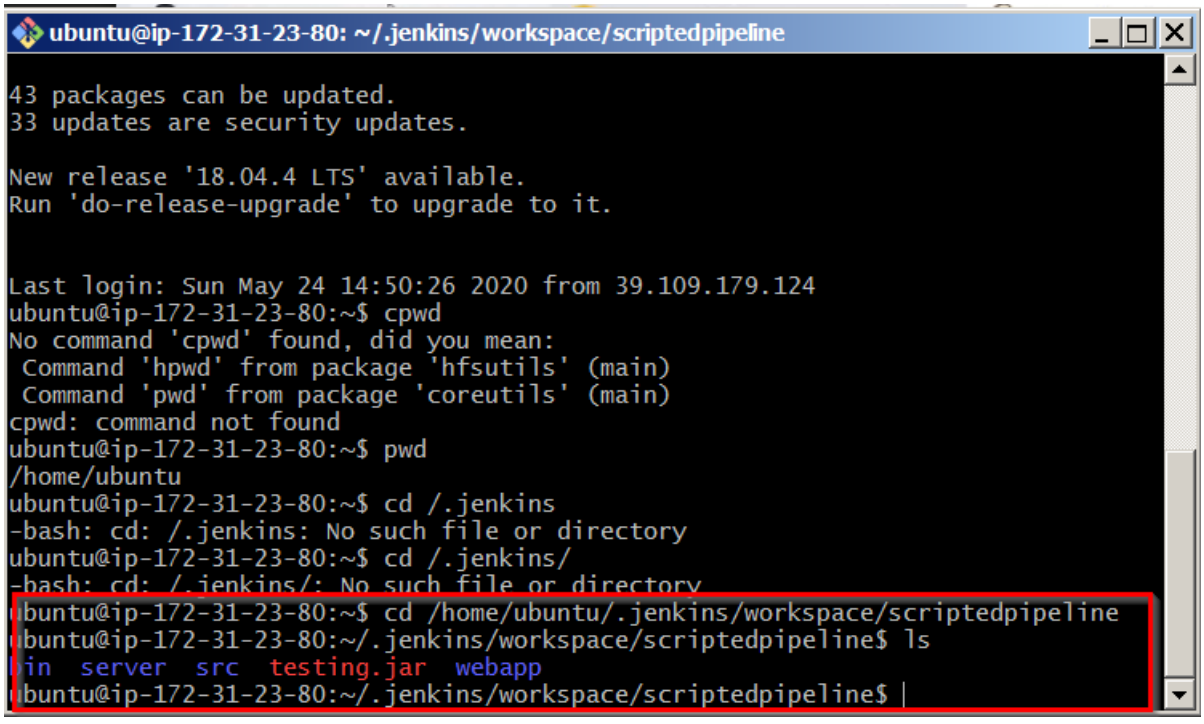
https://github.com/intelliqittrainings/FunctionalTesting.git

generate the groovy code for the same.



Note that every code is downloaded to the Workspace inside the jenkins server. The workspace here is as below.

Do notice the testing.war file inside the jenkins server.



Furthermore, we need to generate a groovy code for executing the testing script.

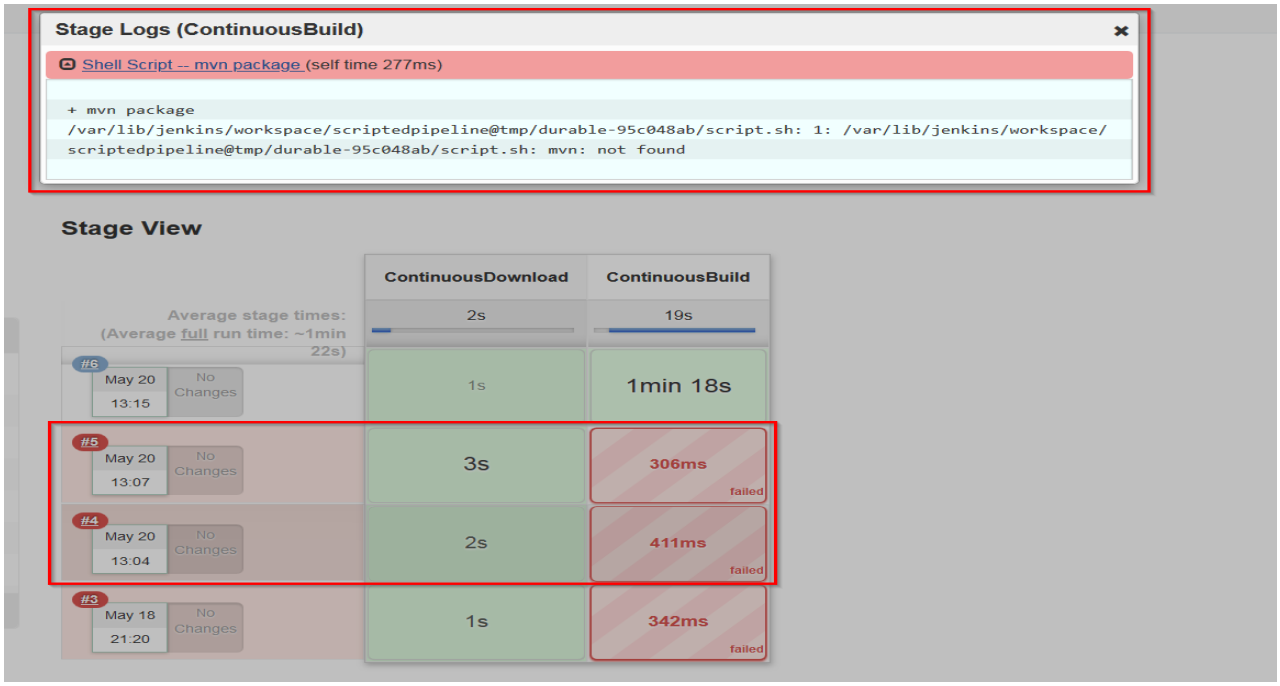
```
15  stage('ContinuousTesting')
16  {
17   git 'https://github.com/intelliqittrainings/FunctionalTesting.git'
18   sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar'
19  }
20 }
```

Similarly, continue with the last stage of the CI/CD process.

In the last stage, we just need to copy the artifact from jenkins server to production server exactly like we did in continuous deployment stage.

LAB – errors Encountered

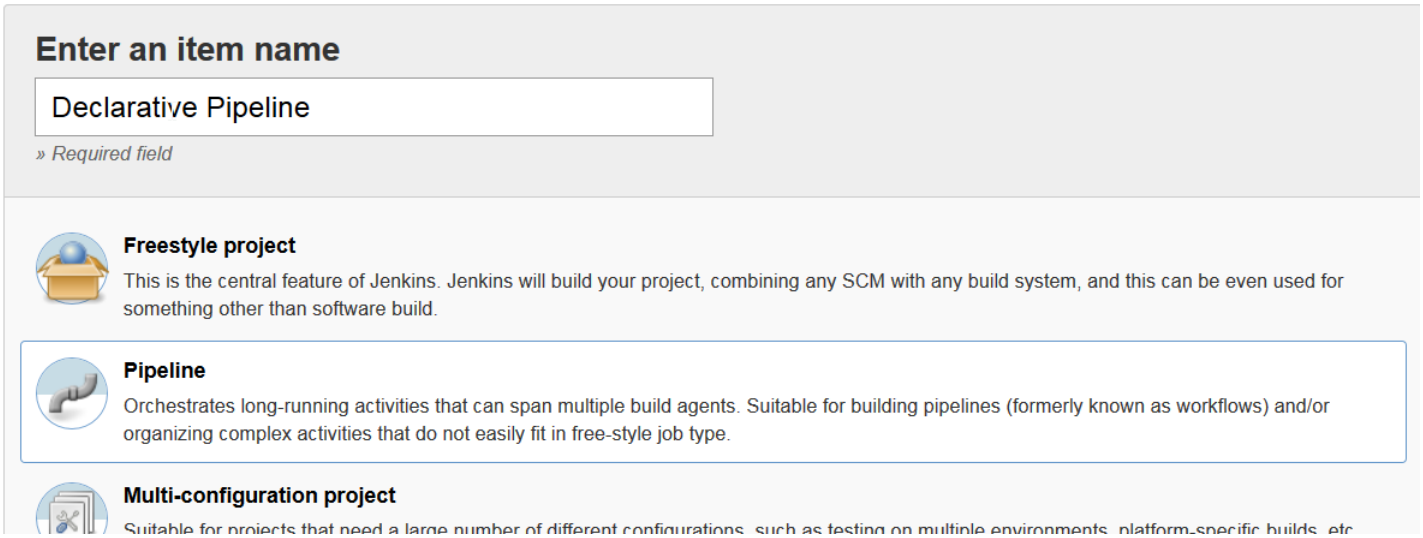
During the Scripted pipeline build, the **continuous Build** goory code wasn't getting executed. It threw the below error.

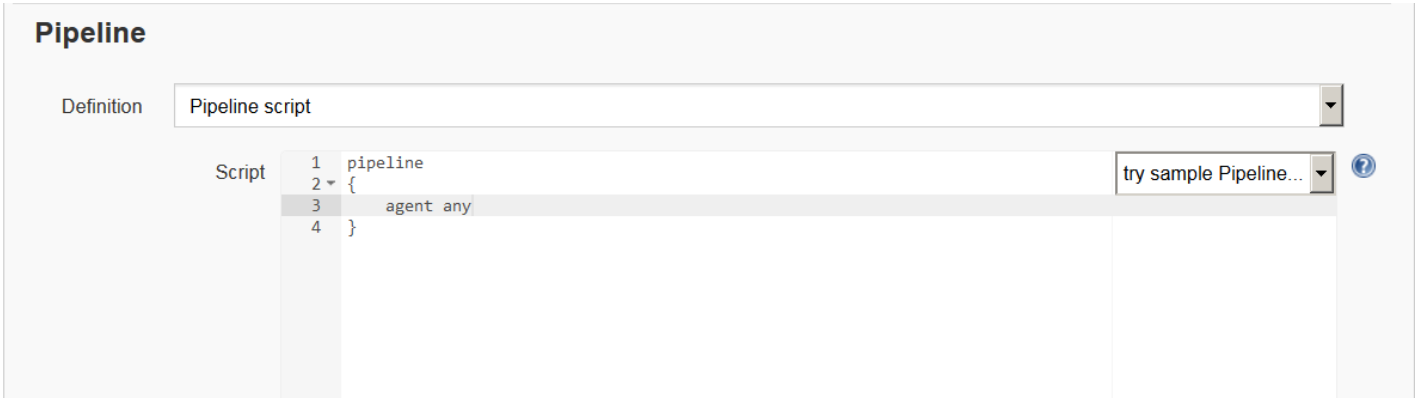


Solution: On investigation, found out that the permanent jenkins installation, didn't have the maven package Installed.

Installing the maven package on the Jenkins server fixed the issue.

LAB – setting up Declarative pipeline.





Declarative pipeline starts with a syntax as seen above, here "agent any" means that we are suggesting the Declarative pipeline to use any available jenkins server whether <u>master or slave.</u>

<u>Stage1 – continuous Download</u>

Notice the syntax is drastically different from the scripted pipeline.

Stage2 – continuous Build

Pipeline

Definition: Pipeline script

Script

```
{
    stage('ContinuousDownload')
    {
        steps
        {
            git 'https://github.com/intelliqittrainings/maven.git'
        }
    }
    stage('ContinuousBuild')
    {
        steps
        {
            sh label: '', script: 'mvn package'
        }
    }
```

Stage View

	ContinuousDownload	ContinuousBuild
Average stage times: (Average full run time: ~19s)	3s	8s
#2 May 25 17:52 No Changes	3s	8s

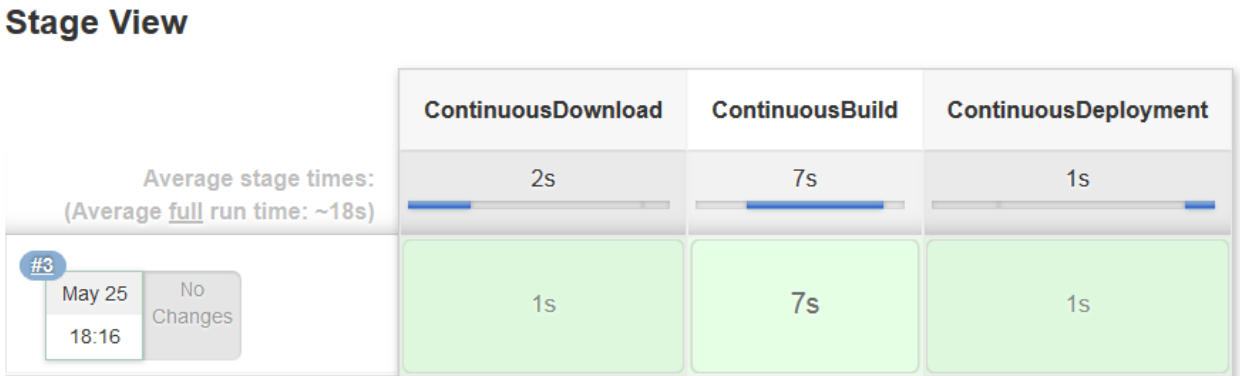
Stage Logs (ContinuousBuild)

Shell Script -- mvn package (self time 8s)

```
[INFO] Processing war project
[INFO] Copying webapp resources [/home/ubuntu/.jenkins/workspace/Declarative Pipeline/webapp/src/main/we
bapp]
[INFO] Webapp assembled in [40 msecs]
[INFO] Building war: /home/ubuntu/.jenkins/workspace/Declarative Pipeline/webapp/target/webapp.war
[INFO] WEB-INF/web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] Maven Project ..................................... SUCCESS [  0.004 s]
[INFO] Server ............................................ SUCCESS [  4.289 s]
[INFO] Webapp ............................................ SUCCESS [  0.868 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 5.454 s
[INFO] Finished at: 2020-05-25T09:52:20+00:00
[INFO] Final Memory: 17M/41M
[INFO] ------------------------------------------------------------------------
```

As seen above the build process has been successfully completed. And the artifact has been created on the jenkins server in the above location.

Stage3 – Continuous Deployment



Stage4 – Continuous Testing

Pipeline

Definition: Pipeline script

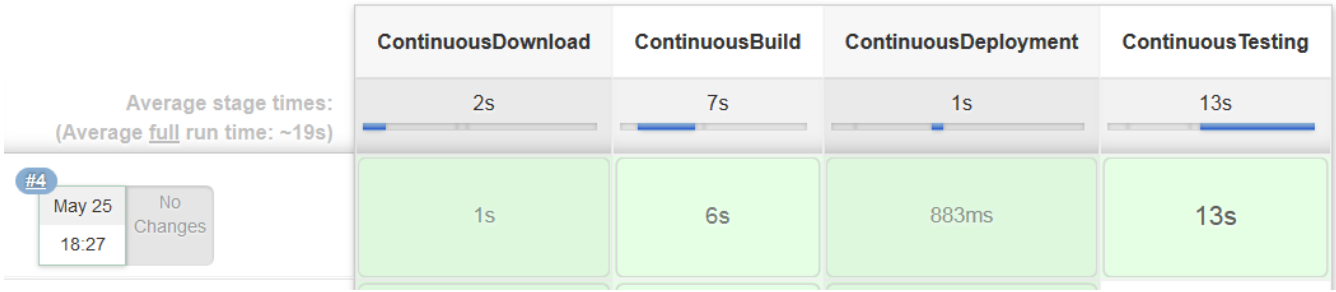
Script

```
22      steps
23      {
24      sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.
25      }
26  }
27  stage('ContinuousTesting')
28  {
29      steps
30      {
31      git 'https://github.com/intelliqittrainings/FunctionalTesting.git'
32      sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar
33      }
34  }
35
36
37
```

Use Groovy Sandbox

Pipeline Syntax

Stage View



Stage5 – Continuous Delivery

Pipeline

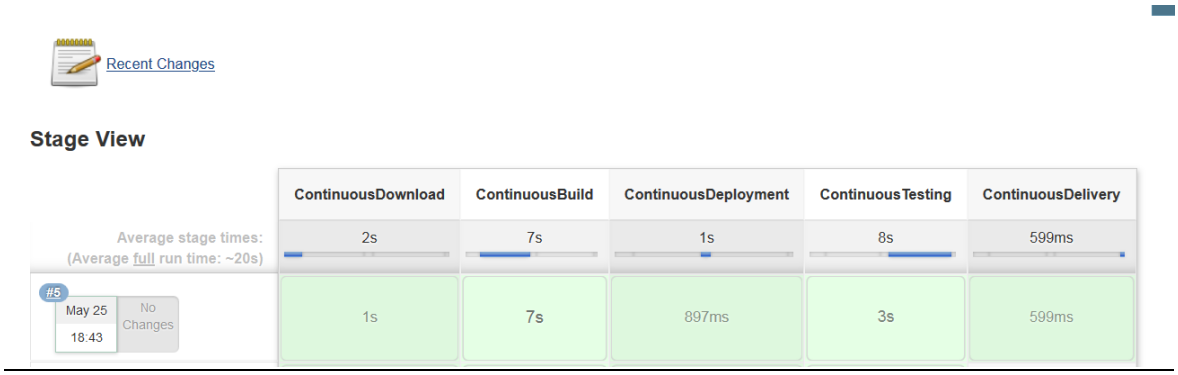
Definition: Pipeline script

Script

```
29      steps
30      {
31      git 'https://github.com/intelliqittrainings/FunctionalTesting.git'
32      sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar'
33      }
34  }
35  stage('ContinuousDelivery')
36  {
37      steps
38      {
39      sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31
40      }
41  }
42
43
44
```

Use Groovy Sandbox

Pipeline Syntax

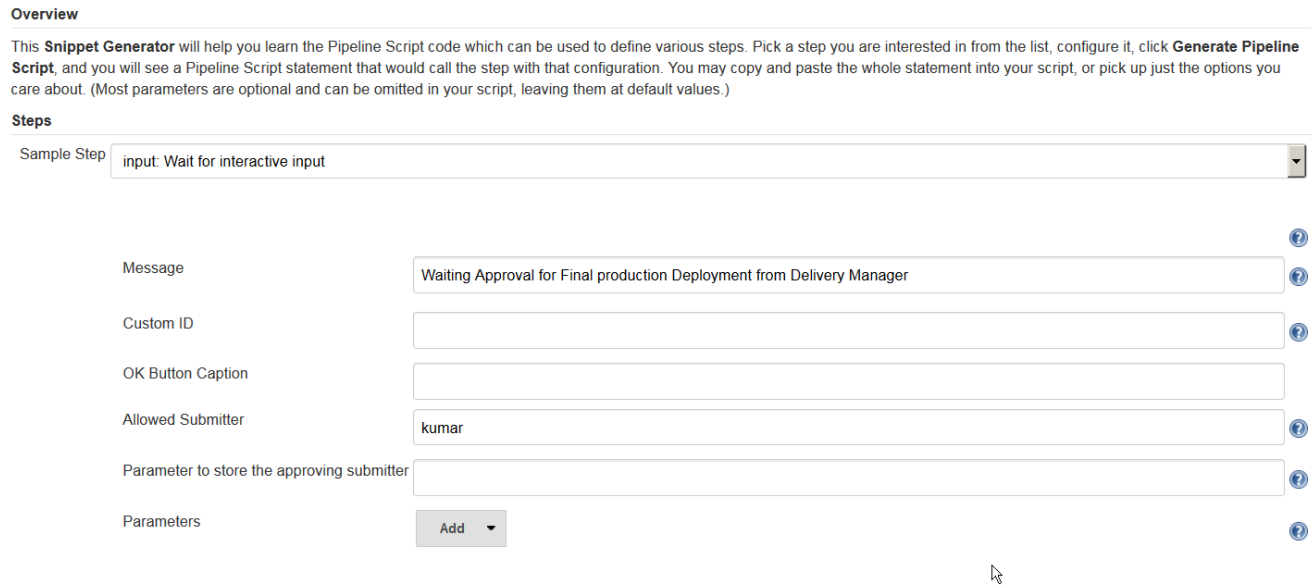


All 5 stages have been successfully completed by Declarative Pipeline

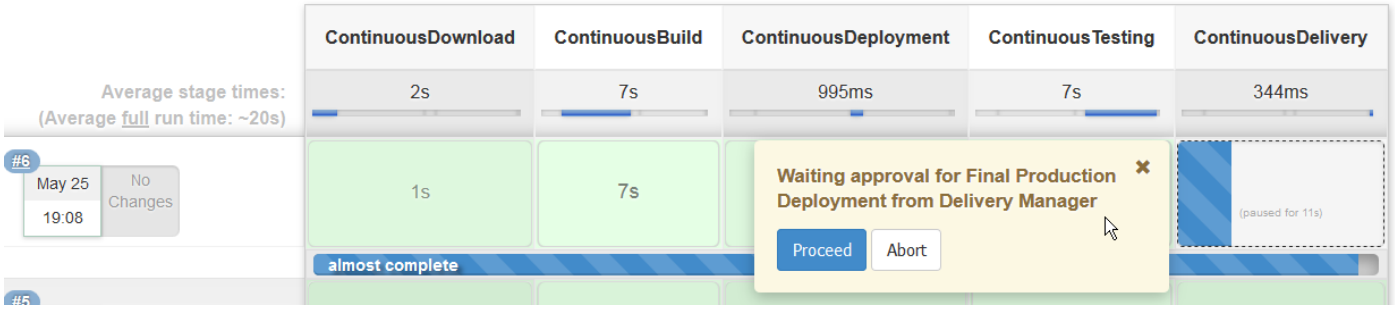
Scenario

In case the 5th stage would be required to be approved by the delivery manager, we can also input an Interactive input inside the groovy code so that during the final deployment of code into the production servers, the delivery manger has the final say.

Go to the pipeline syntax generator and select, “input: wait for interactive input”, and generate the pipeline script



The interactive input code can be put in place as seen above, and when run, jenkins will prompt for approval to proceed further. Wherein, the delivery manager would required to logon to jenkins and approve to proceed.



Above would be the message displayed when an interactive input is required.