

Setup Jenkins from Scratch

- Beginners guide

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

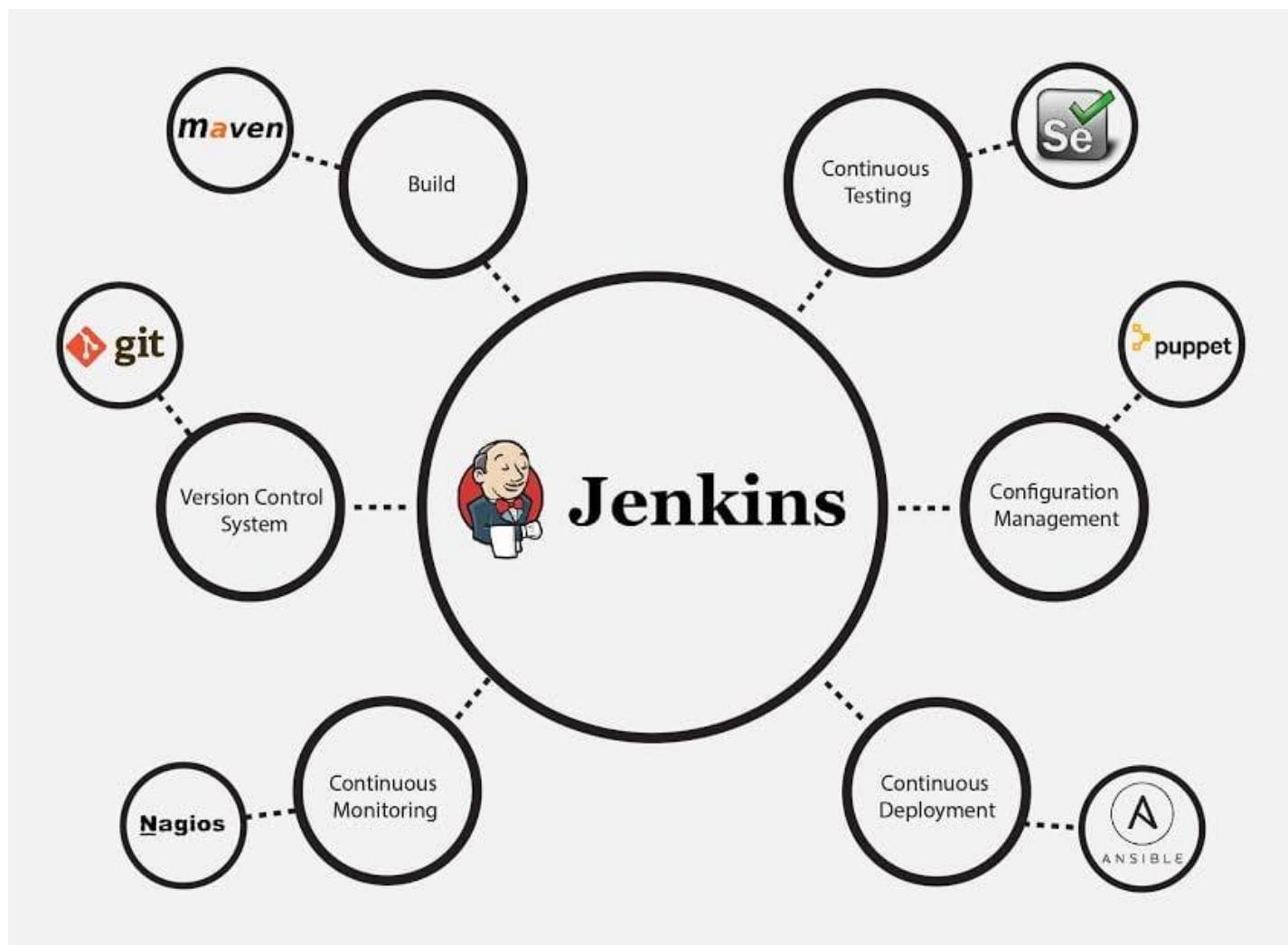
CONTENTS

CONTINUOUS INTEGRATION – CONTINUOUS DELIVERY (CI – CD)	3
LAB TESTING - INSTALLING JENKINS	4
LAB TESTING - INSTALLING PRODUCTION AND QA SERVER.....	9
LAB TESTING STAGE 1 - CONTINUOUS DOWNLOAD CONFIGURATION.....	10
LAB TESTING STAGE 2 - CONTINUOUS BUILD CONFIGURATION.....	13
LAB TESTING STAGE 3 - CONTINUOUS DEPLOY CONFIGURATION ON JENKINS.....	16
LAB TESTING STAGE 4 - CONTINUOUS TESTING CONFIGURATION ON JENKINS	20
LINKING DEVELOPMENT JOB WITH THE TESTING JOB	23
COPYING ARTIFACTS FROM DEVELOPMENT JOB TO TESTING JOB	25
LAB TESTING STAGE 5 - CONTINUOUS TESTING CONFIGURATION ON JENKINS	28
ALTERNATIVE (PERMANENT) METHODS OF JENKINS INSTALLATION.....	29
SETTING UP JENKINS AS A PERMANENT INSTALLATION	29
SETUP OF JENKINS USING TOMCAT.....	30
SCHEDULING THE JENKINS JOB TO EXECUTE AT A SPECIFIC TIME	31
SCHEDULING THE JENKINS JOB TO TRIGGER WHEN DEVELOPERS MAKE ANY NEW COMMITS TO GIT	32
JENKINS ARCHITECTURE.....	33
JENKINS DISTRIBUTED ARCHITECTURE	33
JENKINS MASTER.....	33
JENKINS SLAVE	33
LAB SETTING UP PASSWORDLESS SSH IN MASTER SLAVE (OR BETWEEN ANY TWO LINUX VMS)	34
LAB SETUP OF MASTER –SLAVE JENKINS VM	35
JENKINS PIPELINE	40
SAMPLE SYNTAX OF SCRIPTED PIPELINE.....	40
SAMPLE - DECLARATIVE PIPELINE	40
WHAT IS A JENKINSFILE?	41
LAB – SETTING UP SCRIPTED PIPELINE.....	41
LAB – errors Encountered	47
LAB – SETTING UP DECLARATIVE PIPELINE	48

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 GIT). 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

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP	IP
<input type="checkbox"/>	jenkins vm	i-0255b7c8aa2cf6a06	t2.micro	ap-southeast-1b	● running	✓ 2/2 checks ...	None	🟡	ec2-54-255-154-236.ap...	54.255.154.236
<input type="checkbox"/>	Prod vm	i-068c56bb97ff3df24	t2.micro	ap-southeast-1b	● running	✓ 2/2 checks ...	None	🟡	ec2-18-140-50-148.ap...	18.140.50.148
<input type="checkbox"/>	QA vm	i-0b9e664fab468a66a	t2.micro	ap-southeast-1b	● running	✓ 2/2 checks ...	None	🟡	ec2-13-229-122-234.ap...	13.229.122.234

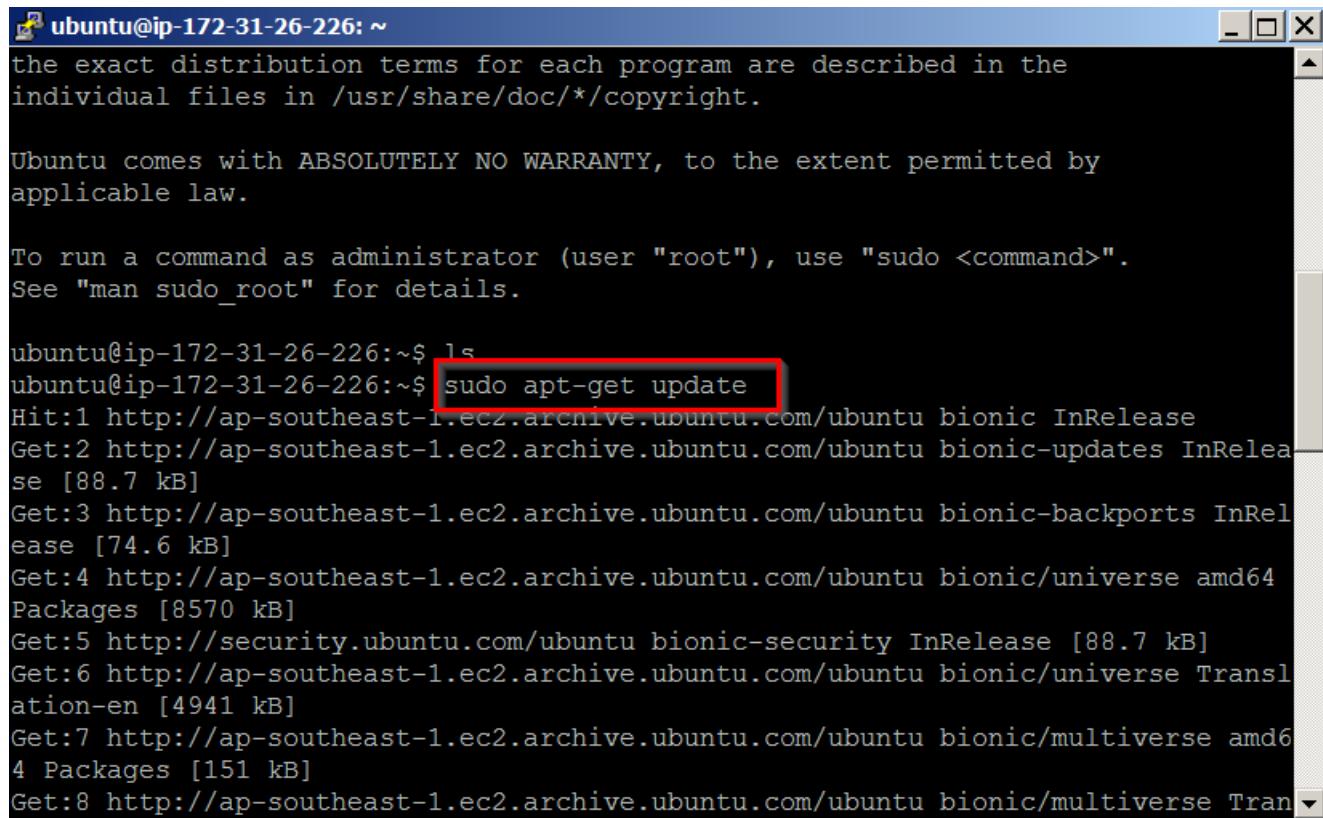
NOTE: this installation of Jenkins instance on the cloud is not ideal, as every time we start the jenkins server we'd need to start the Jenkins.war file manually by running Java –jar jenkins.war.

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

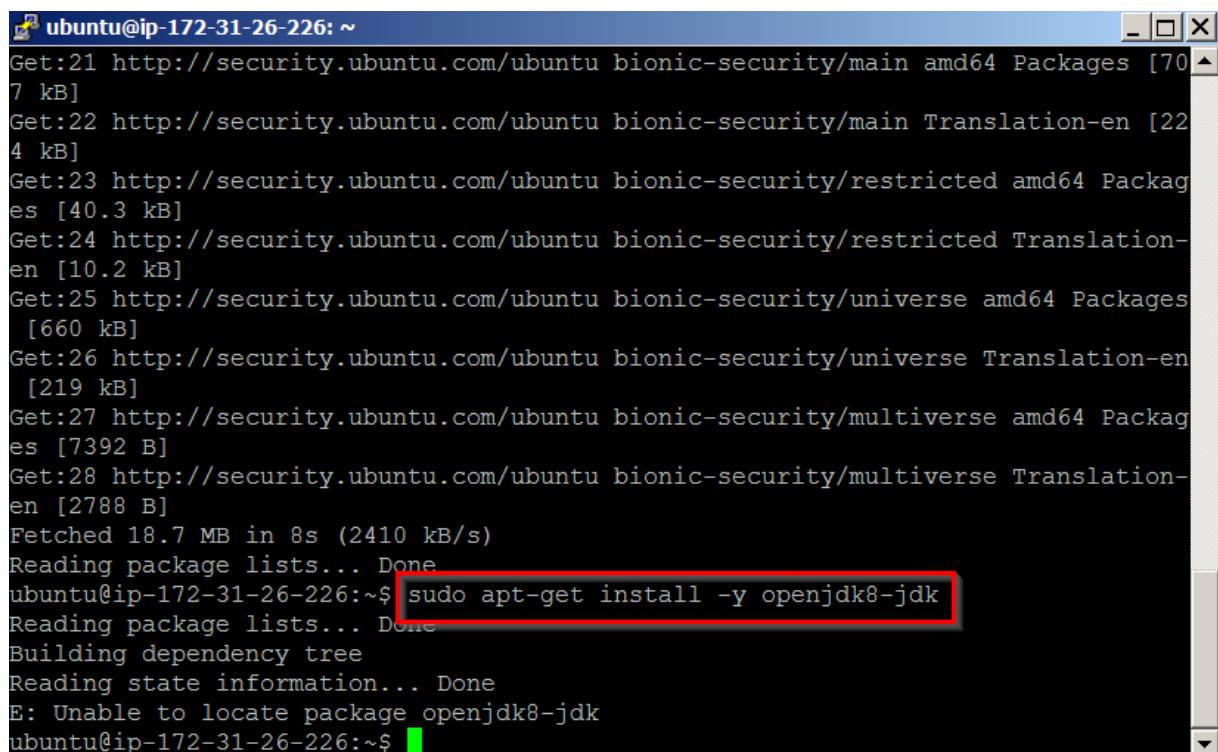
```
sudo apt-get update
```



```
ubuntu@ip-172-31-26-226:~$ sudo apt-get update
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [8570 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:6 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/universe Translation-en [4941 kB]
Get:7 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [151 kB]
Get:8 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/multiverse Translation-en [70 kB]
```

2. Install java

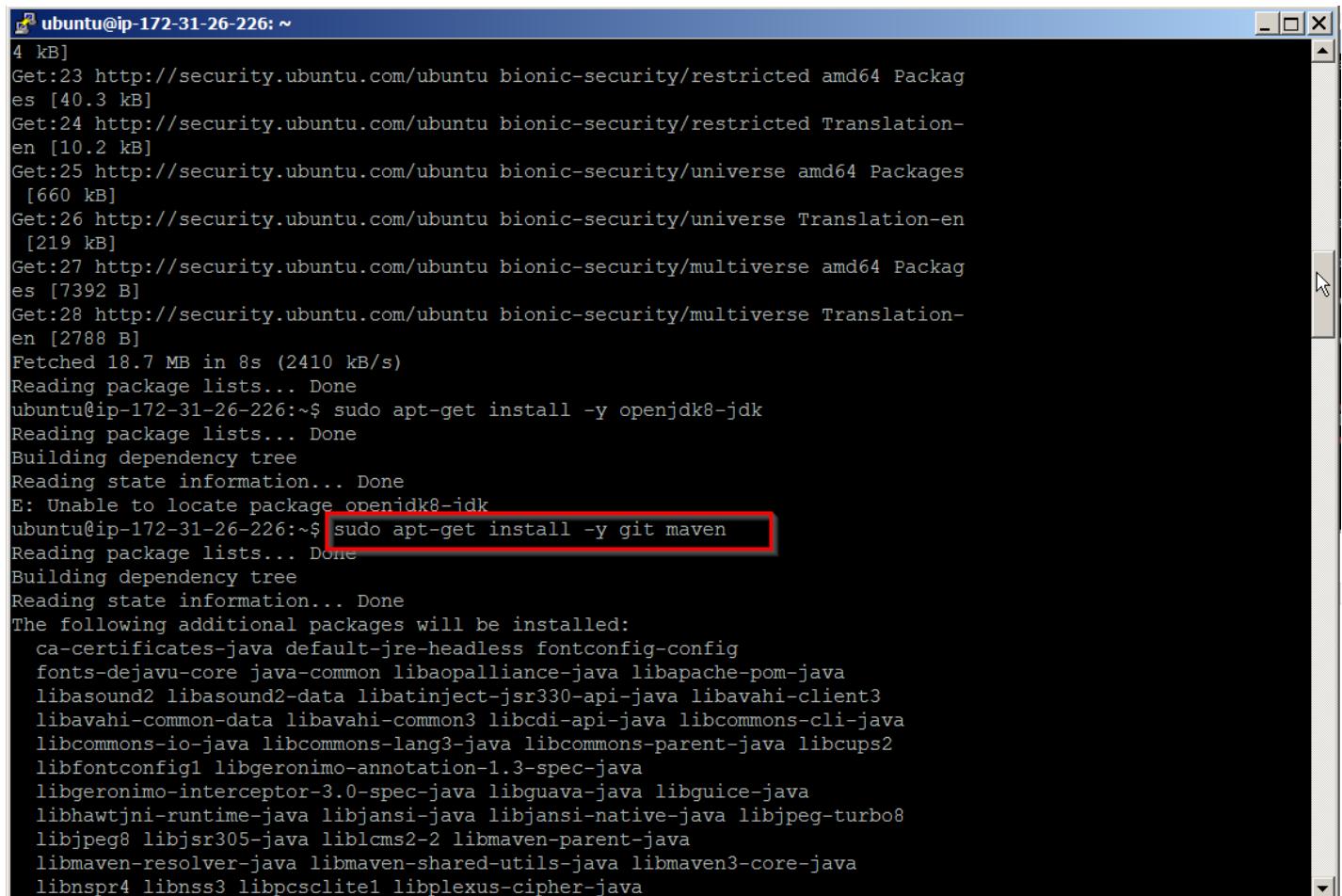
```
sudo apt-get install -y openjdk-8-jdk
```



```
ubuntu@ip-172-31-26-226:~$ sudo apt-get install -y openjdk-8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package openjdk-8-jdk
```

3. Install git and maven

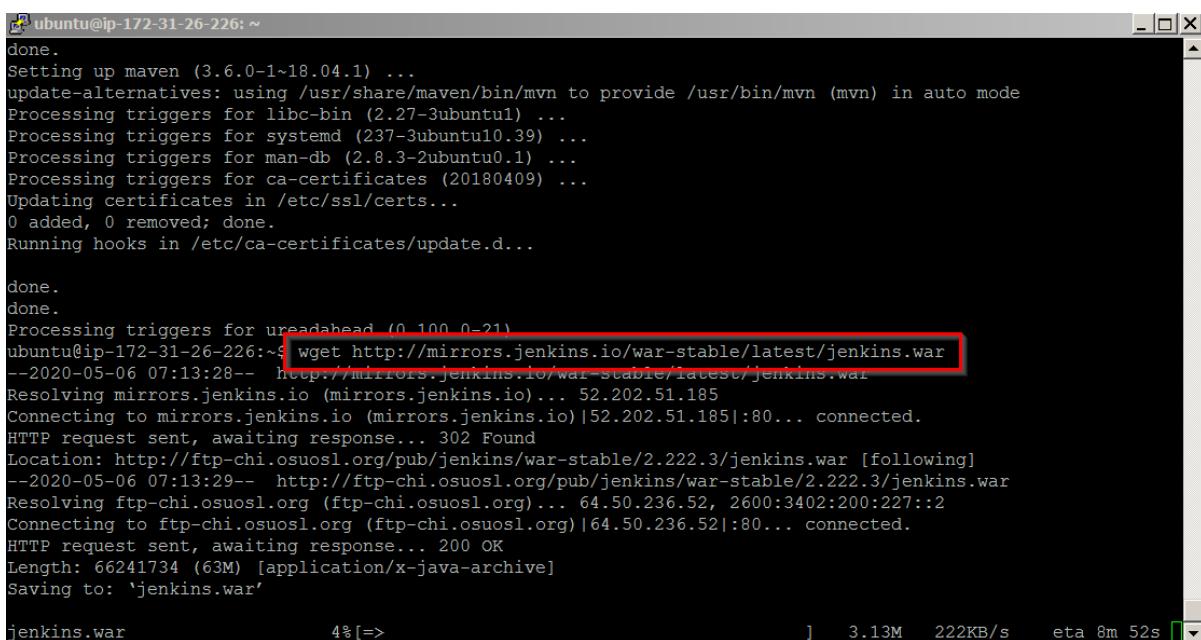
```
sudo apt-get install -y git maven
```



```
ubuntu@ip-172-31-26-226: ~
4 kB]
Get:23 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [40.3 kB]
Get:24 http://security.ubuntu.com/ubuntu bionic-security/restricted Translation-en [10.2 kB]
Get:25 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [660 kB]
Get:26 http://security.ubuntu.com/ubuntu bionic-security/universe Translation-en [219 kB]
Get:27 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [7392 B]
Get:28 http://security.ubuntu.com/ubuntu bionic-security/multiverse Translation-en [2788 B]
Fetched 18.7 MB in 8s (2410 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-26-226:~$ sudo apt-get install -y openjdk8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package openjdk8-jdk
ubuntu@ip-172-31-26-226:~$ sudo apt-get install -y git maven
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fontconfig-config
  fonts-dejavu-core java-common libaopalliance-java libapache-pom-java
  libasound2 libasound2-data libatinject-jsr30-api-java libavahi-client3
  libavahi-common-data libavahi-common3 libcdi-api-java libcommons-cli-java
  libcommons-io-java libcommons-lang3-java libcommons-parent-java libcurl2
  libfontconfig1 libgeronimo-annotation-1.3-spec-java
  libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
  libhawtjni-runtime-java libjansi-java libjansi-native-java libjpeg-turbo8
  libjpeg8 libjsr305-java liblcms2-2 libmaven-parent-java
  libmaven-resolver-java libmaven-shared-utils-java libmaven3-core-java
  libnss4 libnss3 libpcsc-lite1 libplexus-cipher-java
```

4. Download jenkins.war

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



```
ubuntu@ip-172-31-26-226: ~
done.
Setting up maven (3.6.0-1~18.04.1) ...
update-alternatives: using /usr/share/maven/bin/mvn to provide /usr/bin/mvn (mvn) in auto mode
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for systemd (237-3ubuntu0.39) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for ca-certificates (20180409) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
Processing triggers for ureadahead (0.100.0-21)
ubuntu@ip-172-31-26-226:~$ wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war
--2020-05-06 07:13:28-- http://mirrors.jenkins.io/war-stable/latest/jenkins.war
Resolving mirrors.jenkins.io (mirrors.jenkins.io)... 52.202.51.185
Connecting to mirrors.jenkins.io (mirrors.jenkins.io)|52.202.51.185|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://ftp-chi.osuosl.org/pub/jenkins/war-stable/2.222.3/jenkins.war [following]
--2020-05-06 07:13:29-- http://ftp-chi.osuosl.org/pub/jenkins/war-stable/2.222.3/jenkins.war
Resolving ftp-chi.osuosl.org (ftp-chi.osuosl.org)... 64.50.236.52, 2600:3402:200:227::2
Connecting to ftp-chi.osuosl.org (ftp-chi.osuosl.org)|64.50.236.52|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 66241734 (63M) [application/x-java-archive]
Saving to: 'jenkins.war'

jenkins.war          4%[=>]   3.13M  222KB/s  eta 8m 52s
```

5. To start jenkins

Java -jar jenkins.war

```
ubuntu@ip-172-31-26-226: ~
--2020-05-06 07:13:29-- http://ftp-chi.osuosl.org/pub/jenkins/war-stable/2.222.3/jenkins.war
Resolving ftp-chi.osuosl.org (ftp-chi.osuosl.org)... 64.50.236.52, 2600:3402:200:227::2
Connecting to ftp-chi.osuosl.org (ftp-chi.osuosl.org)|64.50.236.52|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 66241734 (63M) [application/x-java-archive]
Saving to: 'jenkins.war'

jenkins.war          100%[=====] 63.17M 97.0KB/s   in 6m 7s

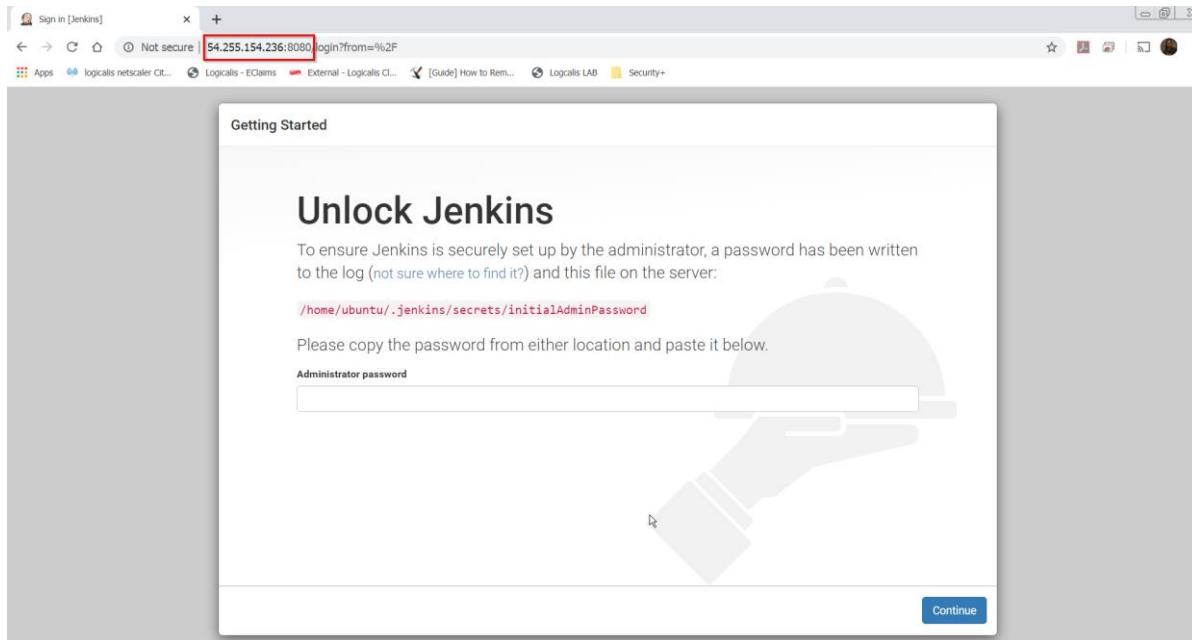
2020-05-06 07:19:37 (176 KB/s) - 'jenkins.war' saved [66241734/66241734]

ubuntu@ip-172-31-26-226:~$ ls
jenkins.war
ubuntu@ip-172-31-26-226:~$ java-jar jenkins.war
java-jar: command not found
ubuntu@ip-172-31-26-226:~$ java -jar jenkins.war
Running from: /home/ubuntu/jenkins.war
webroot: $user.home/.jenkins
2020-05-06 07:22:45.078+0000 [id=1]      INFO  org.eclipse.jetty.util.log.Log#initialized: Logging initialized @1406ms to org.eclipse.jetty.util.log.JavaUtilLog
2020-05-06 07:22:45.240+0000 [id=1]      INFO  winstone.Logger#logInternal: Beginning extraction from war fi
```

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

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 2.222.3

The screenshot shows the Jenkins 2.222.3 'Getting Started' page. At the top, there's a navigation bar with icons for back, forward, search, and other functions. Below it is a toolbar with links like 'Most Visited', 'Getting Started', 'Future bank Logo', 'Shopping & Dining - SL...', 'Suggested Sites', 'Web Slice Gallery', and two 'Login' buttons. The main content area has a title 'Getting Started' and a sub-section 'Getting Started'. Below that is a grid of plugin icons and names:

Folders	OWASP Markup Formatter	Build Timeout	Credentials Binding
Timestamper	Workspace Cleanup	Ant	Gradle
Pipeline	Github Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View
Git	Subversion	SSH Build Agents	Matrix Authorization Strategy
PAM Authentication	LDAP	Email Extension	Mailer

On the right side of the grid, there's a note: '** Trilead API' with a small icon. At the bottom of the grid, it says '** - required dependency'. The footer of the page shows 'Jenkins 2.222.3' and the URL '18.139.161.76:8080/#'.

Create first admin user--->Continue

The screenshot shows the 'Create First Admin User' form. It has fields for 'Username' (kedar.pavaskar), 'Password' (redacted), 'Confirm password' (redacted), 'Full name' (kedar pavaskar), and 'E-mail address' (kedar.pavaskar@hotmail.com). There are two buttons at the bottom: 'Continue as admin' and 'Save and 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
```

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

The screenshot shows the Jenkins interface for creating a new item. A red box highlights the 'Item name' input field containing 'Development-job'. Another red box highlights the 'Freestyle project' option under the 'Project types' section.

The screenshot shows the GitHub repository page for 'intelligittrainings / maven'. A red box highlights the 'Clone with HTTPS' button, which displays the URL 'https://github.com/intelligittrainings/maven.git'.

Example: <https://github.com/intelligittrainings/maven.git>

The screenshot shows the Jenkins job configuration for 'Development-job'. Under 'Source Code Management', the 'Git' option is selected. Other sections shown include 'Build Triggers' and 'Build Environment'.

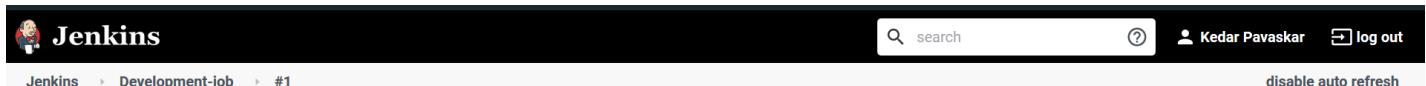
Mentioned the path of the Remote repository that the developer supposedly uploads the code.

“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.



Jenkins > Development-job > #1

search ? Kedar Pavaskar log out

Back to Project Status Changes Console Output Edit Build Info Console Output Delete build '#1' Git Build Data No Tags

Build #1 (May 9, 2020 2:13:42 PM)

No changes.

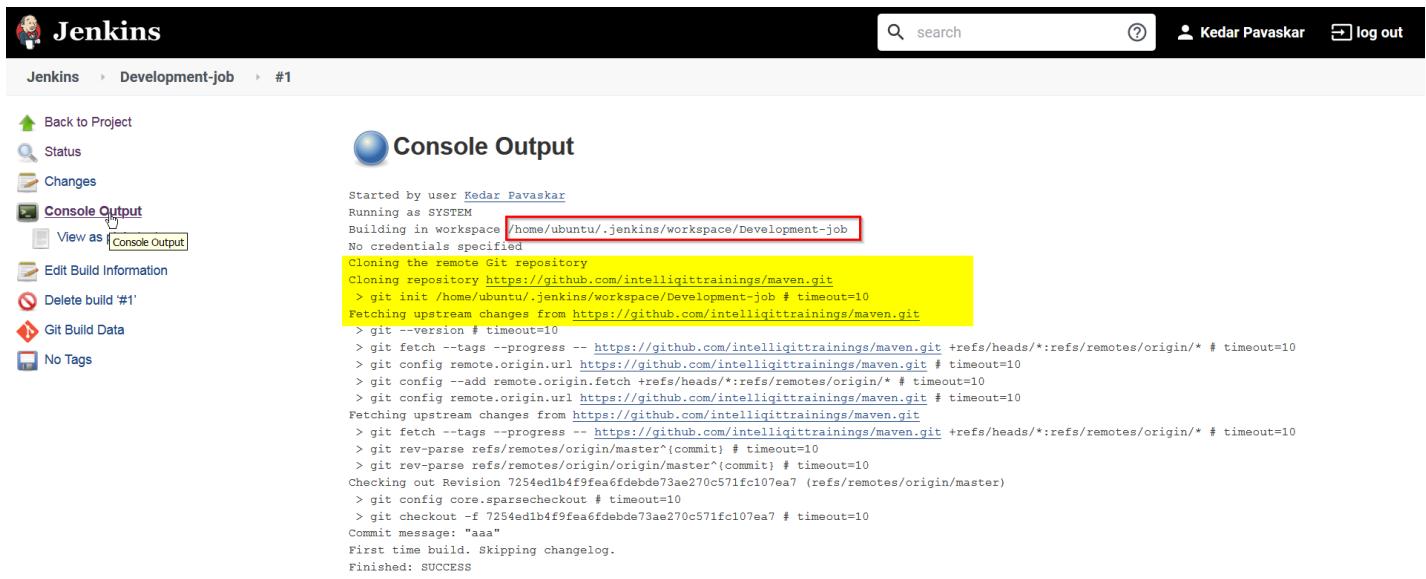
Started by user [Kedar Pavaskar](#)

Revision: 7254ed1b4f9fea6fdebde73ae270c571fc107ea7

• refs/remotes/origin/master

Keep this build forever Started 6 min 6 sec ago Took 7.4 sec Add description

Click on console output as seen below.



Jenkins > Development-job > #1

Back to Project Status Changes **Console Output** View as Console Output Edit Build Information Delete build '#1' Git Build Data No Tags

Console Output

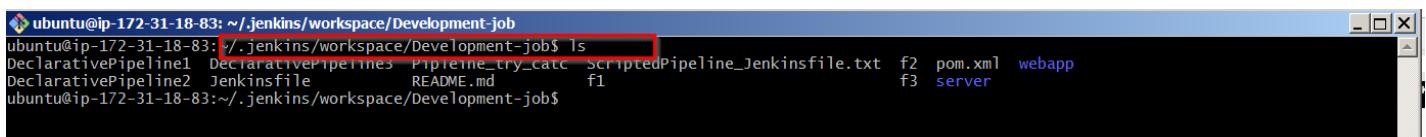
Started by user [Kedar Pavaskar](#)
Running as SYSTEM
Building in workspace [/home/ubuntu/.jenkins/workspace/Development-job](#)
No credentials specified

```
Cloning the remote Git repository https://github.com/intelliqittrainings/maven.git
> git init /home/ubuntu/.jenkins/workspace/Development-job # timeout=10
Fetching upstream changes from https://github.com/intelliqittrainings/maven.git
> git --version # timeout=10
> git fetch --tags --progress -- https://github.com/intelliqittrainings/maven.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/intelliqittrainings/maven.git # timeout=10
> git config --add remote.origin.fetch refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/intelliqittrainings/maven.git # timeout=10
Fetching upstream changes from https://github.com/intelliqittrainings/maven.git
> git fetch --tags --progress -- https://github.com/intelliqittrainings/maven.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^(commit) # timeout=10
> git rev-parse refs/remotes/origin/origin/master^(commit) # timeout=10
Checking out Revision 7254ed1b4f9fea6fdebde73ae270c571fc107ea7 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 7254ed1b4f9fea6fdebde73ae270c571fc107ea7 # timeout=10
Commit message: "aa"
First time build. Skipping changelog.
Finished: SUCCESS
```

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.



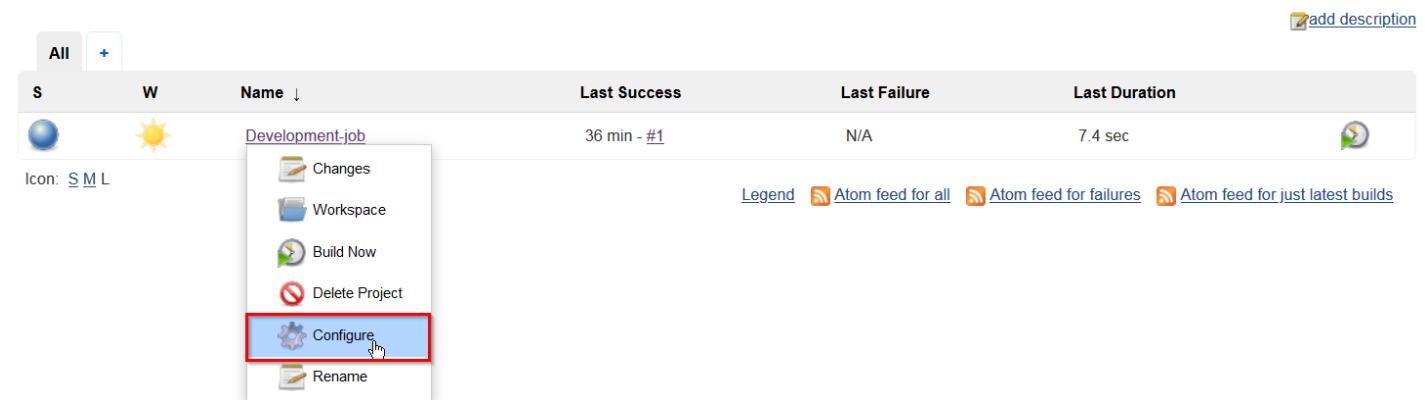
ubuntu@ip-172-31-18-83: ~/jenkins/workspace/Development-job\$ ls

DeclarativePipeline1 DeclarativePipeline Pipeline_ry_catt ScriptedPipeline_Jenkinsfile.txt f2 pom.xml webapp DeclarativePipeline2 Jenkinsfile README.md f1 f3 server

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.

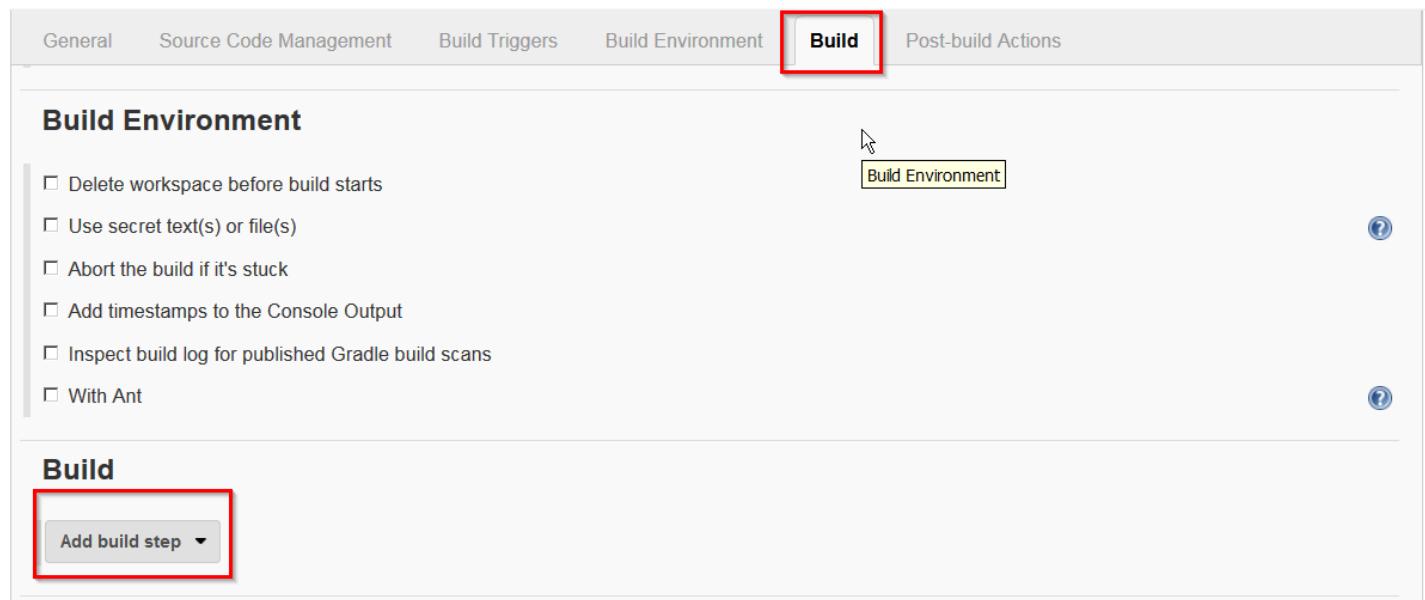
LAB testing Stage 2 - Continuous Build configuration

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



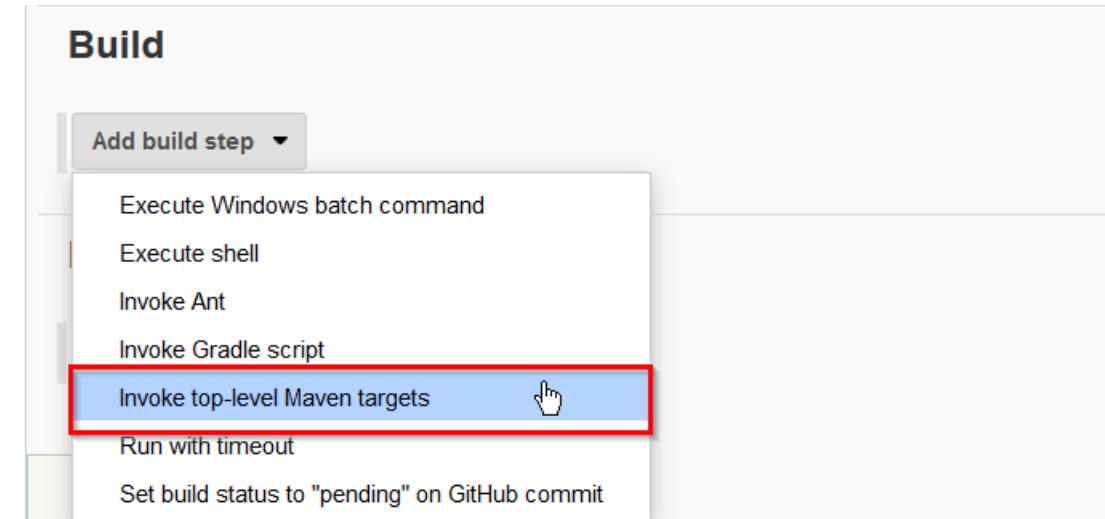
The screenshot shows the Jenkins dashboard with a single project listed: 'Development-job'. The project status is 'Last Success: 36 min - #1'. Below the project name is a context menu with several options: 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure' (which is highlighted with a red box), and 'Rename'. At the bottom of the menu is a link 'Icon: S M L'.

Select the build tab and click “add build step”



The screenshot shows the 'Development-job' configuration page. The 'Build' tab is selected and highlighted with a red box. Under the 'Build Environment' section, there are several checkboxes for build-related options. In the 'Build' section, there is a 'Add build step' button with a dropdown arrow, which is also highlighted with a red box.

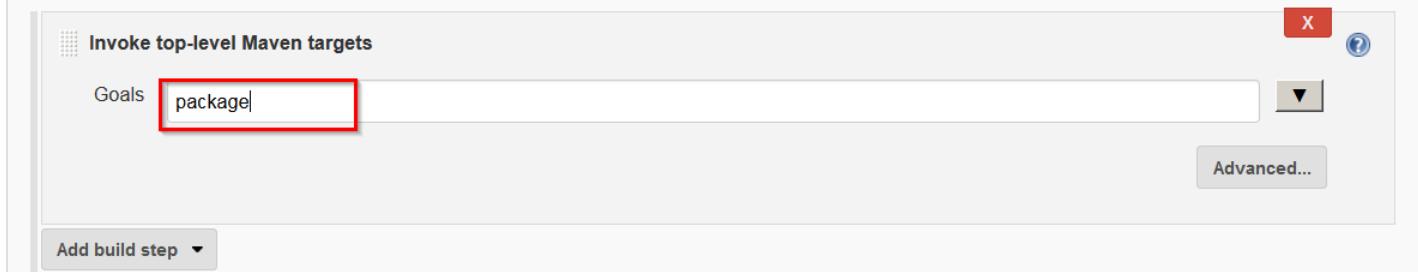
Click Invoke top-level Maven targets.



The screenshot shows the 'Build' configuration sub-page. The 'Add build step' dropdown is open, displaying several build step options. The 'Invoke top-level Maven targets' option is highlighted with a red box and has a cursor icon pointing at it.

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

Build

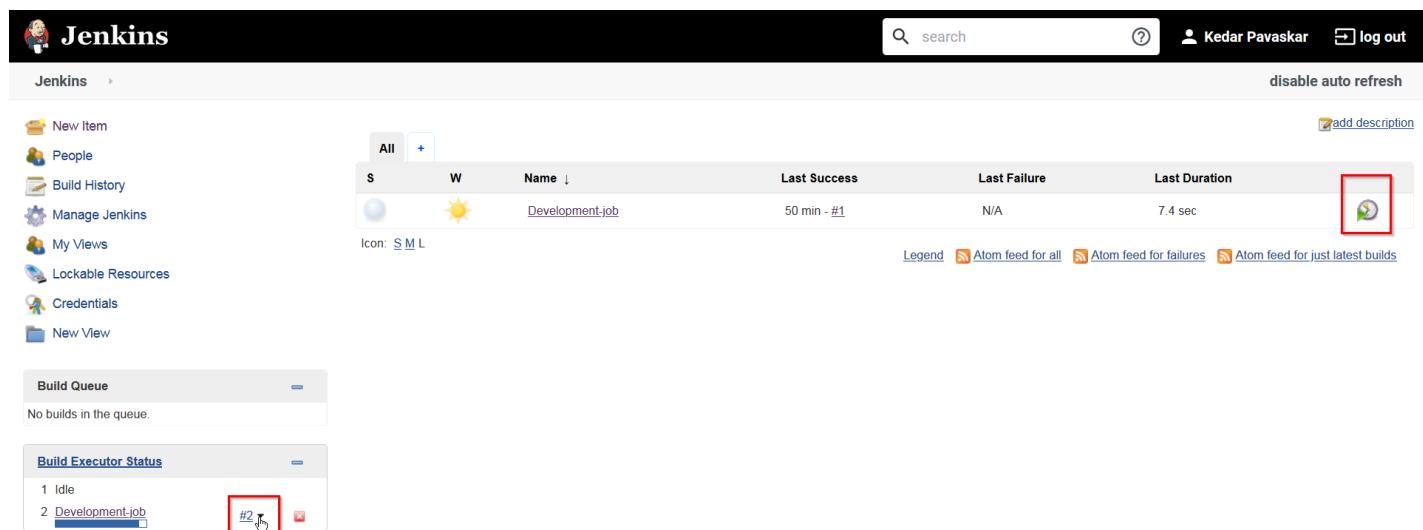


The screenshot shows the Jenkins build configuration interface. At the top, there's a section titled "Invoke top-level Maven targets". Below it, a "Goals" input field contains the text "package", which is highlighted with a red rectangle. To the right of the input field are buttons for "Advanced..." and a dropdown menu. At the bottom left, there's a "Add build step" button.

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.*

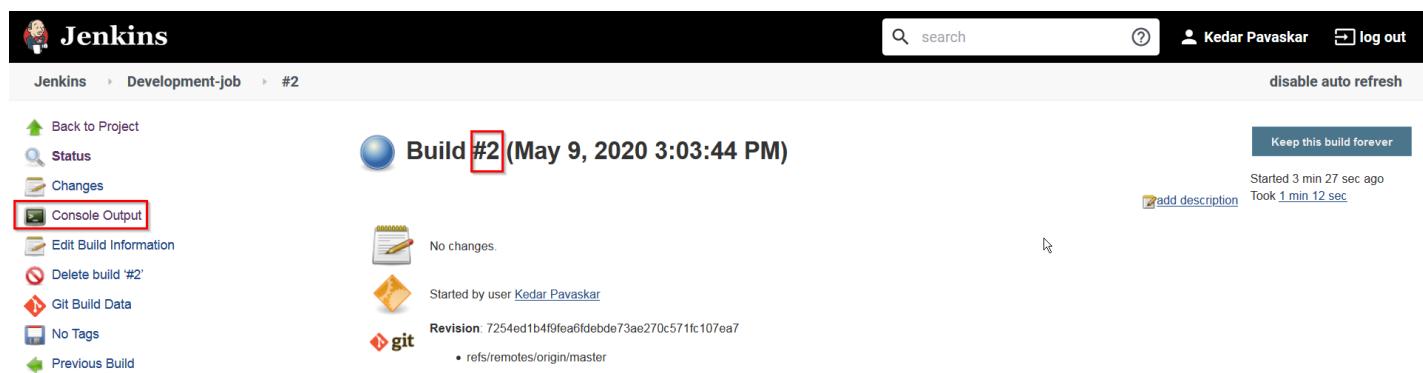
Package is a stage in the Maven Lifecycle, where the code developed by the developer, will converted into an artifact or setup file.



The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like "New Item", "People", "Build History", etc. The main area shows a table of jobs. One job, "Development-job", is listed with an icon of a sun, last success at 50 min - #1, and a duration of 7.4 sec. Below the table, there's a "Build Queue" section stating "No builds in the queue." and a "Build Executor Status" section showing "1 Idle" and "2 Development-job" with a link "#2" highlighted with a red box.

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.



The screenshot shows the Jenkins build details for "Development-job" build #2. The top bar shows "Jenkins > Development-job > #2". The left sidebar has links like "Back to Project", "Status", "Changes", "Console Output" (which is highlighted with a red box), "Edit Build Information", "Delete build '#2'", "Git Build Data", "No Tags", and "Previous Build". The main content area shows the build summary: "Build #2 (May 9, 2020 3:03:44 PM)". It includes a "Keep this build forever" button, a "Started 3 min 27 sec ago" message, and a "Took 1 min 12 sec" message. Below this, there are sections for "No changes.", "Started by user Kedar Pavaskar", and "Revision: 7254ed1b4f9fea6fdebd73ae270c571fc07ea7".

Click on the console output.

```
Progress (3): 33 kB | 410/431 kB | 25 kB
Progress (3): 33 kB | 414/431 kB | 25 kB

Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-filtering/1.0-beta-2/maven-filtering-1.0-beta-2.jar (33 kB at 93 kB/s)
Progress (2): 418/431 kB | 25 kB
Progress (2): 422/431 kB | 25 kB
Progress (2): 426/431 kB | 25 kB
Progress (2): 430/431 kB | 25 kB
Progress (2): 431 kB | 25 kB

Downloaded from central: https://repo.maven.apache.org/maven2/xpp3/xpp3\_min/1.1.4c/xpp3\_min-1.1.4c.jar (25 kB at 69 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/com/thoughtworks/xstream/xstream/1.3.1/xstream-1.3.1.jar (431 kB at 806 kB/s)
[[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 [32 mssecs]
[[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 [ 01:01 min]
[[1:34mINFO[m] Webapp ..... [1;32mSUCCESS[m [ 7.404 s]
[[1:34mINFO[m] [1m-----[m
[[1:34mINFO[m] [1;32mBUILD SUCCESS[m
[[1:34mINFO[m] [1m-----[m
[[1:34mINFO[m] Total time: 01:09 min
[[1:34mINFO[m] Finished at: 2020-05-09T15:04:57Z
[[1:34mINFO[m] [1m-----[m
[[1:34mINFO[m] Finished: SUCCESS
```

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

Deploy to container

This plugin allows you to deploy a war to a container after a successful build.
Glassfish 3.x remote deployment

Verifying Jenkins Plugins

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Deploy to container

Installing 

Loading plugin extensions

Pending 

→ [Go back to the top page](#)

(you can start using the installed plugins right away)

→ Restart Jenkins when installation is complete and no jobs are running 

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Development-job	39 min - #2	N/A	1 min 12 sec 

Icon: S M L

[Changes](#) [Workspace](#) [Build Now](#) [Delete Project](#) [Configure](#) [ReName](#)

[add description](#)

[Legend](#) [Atom feed for all](#) [Atom feed for failures](#) [Atom feed for just latest builds](#)

General Source Code Management Build Triggers Build Environment Build **Post-build Actions**

Add timestamps to the Console Output
 Inspect build log for published Gradle build scans
 With Ant

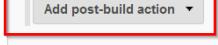
Build

Invoke top-level Maven targets

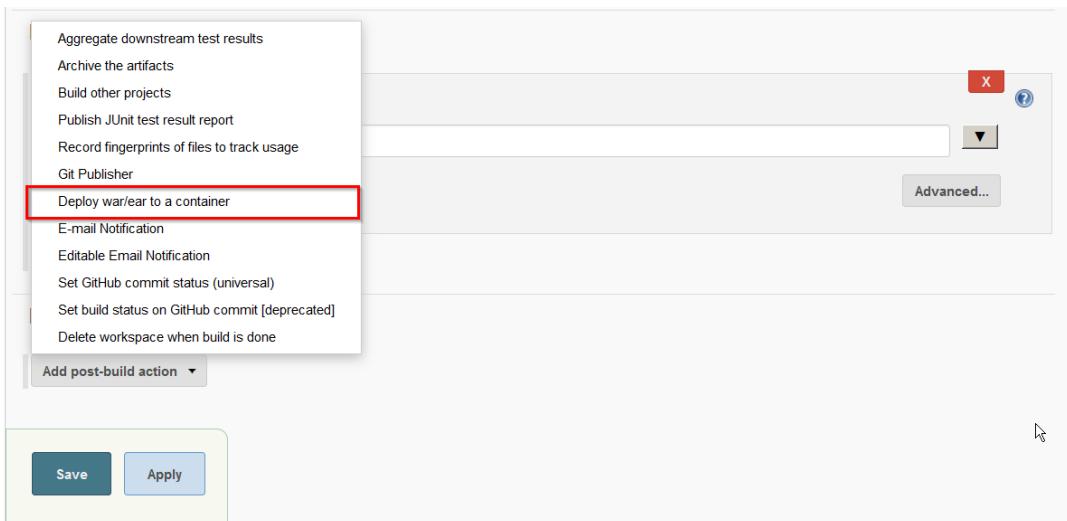
Goals: package 

Add build step ▾

Post-build Actions

Add post-build action ▾ 

Save **Apply**



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.

Post-build Actions

A screenshot of the Jenkins post-build actions configuration for the “Deploy war/ear to a container” action. The configuration fields are: WAR/EAR files (set to “***.war”), Context path (set to “kedarloginapp”), Containers (with an “Add Container” dropdown menu open, showing options like JBoss AS 3.x, JBoss AS 4.x, etc.), and Deploy on failure (unchecked). Below the configuration is an “Add post-build action” button.

“***.war” means in the current working directory it will look for any file with Extension .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.

Post-build Actions

A screenshot of the Jenkins post-build actions configuration for the “Deploy war/ear to a container” action. The configuration fields are: WAR/EAR files (set to “***.war”), Context path (set to “kedarloginapp”), Containers (with an “Add Container” dropdown menu open, showing options like JBoss AS 3.x, JBoss AS 4.x, etc.), and Deploy on failure (unchecked). Below the configuration is an “Add post-build action” button. At the bottom are “Save” and “Apply” buttons. A red box highlights the “Tomcat 8.x Remote” option in the “Add Container” dropdown menu.

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

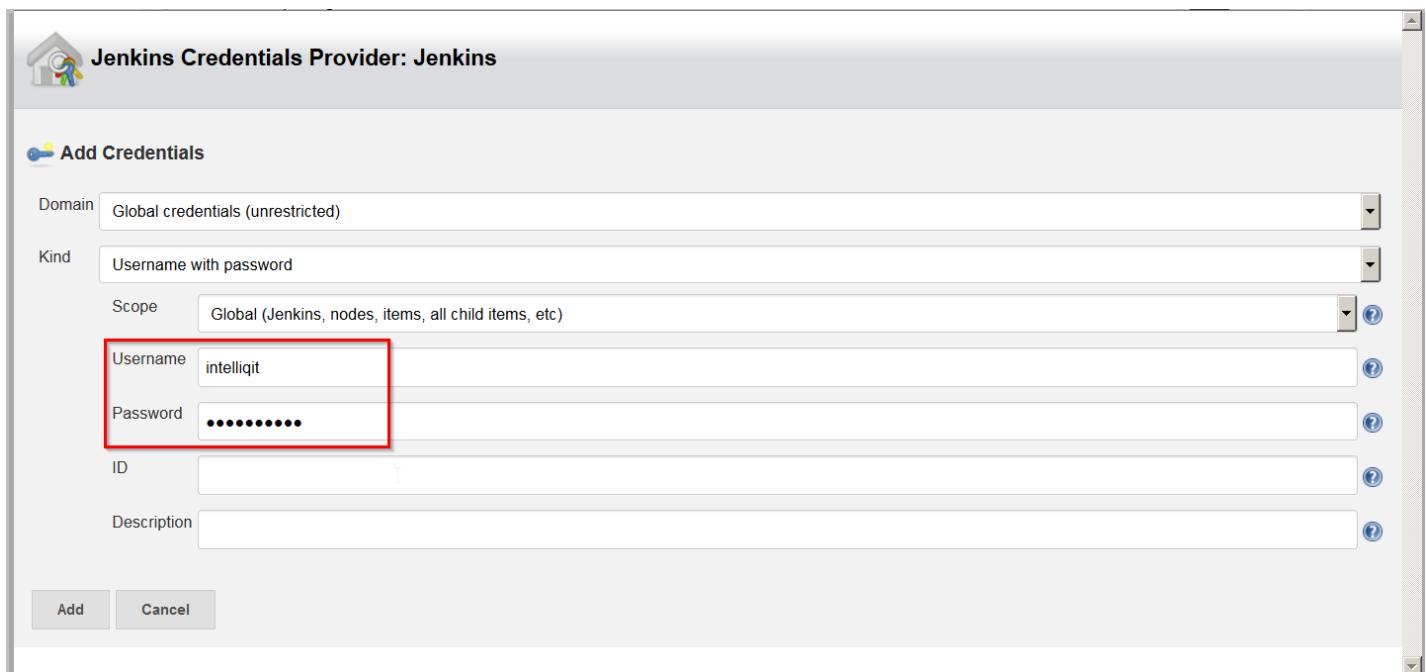
Username: intelliqit

Password: *****

ID:

Description:

Add Cancel



Deploy war/ear to a container

WAR/EAR files: ***.war

Context path: kedarloginapp

Containers

Tomcat 8.x Remote

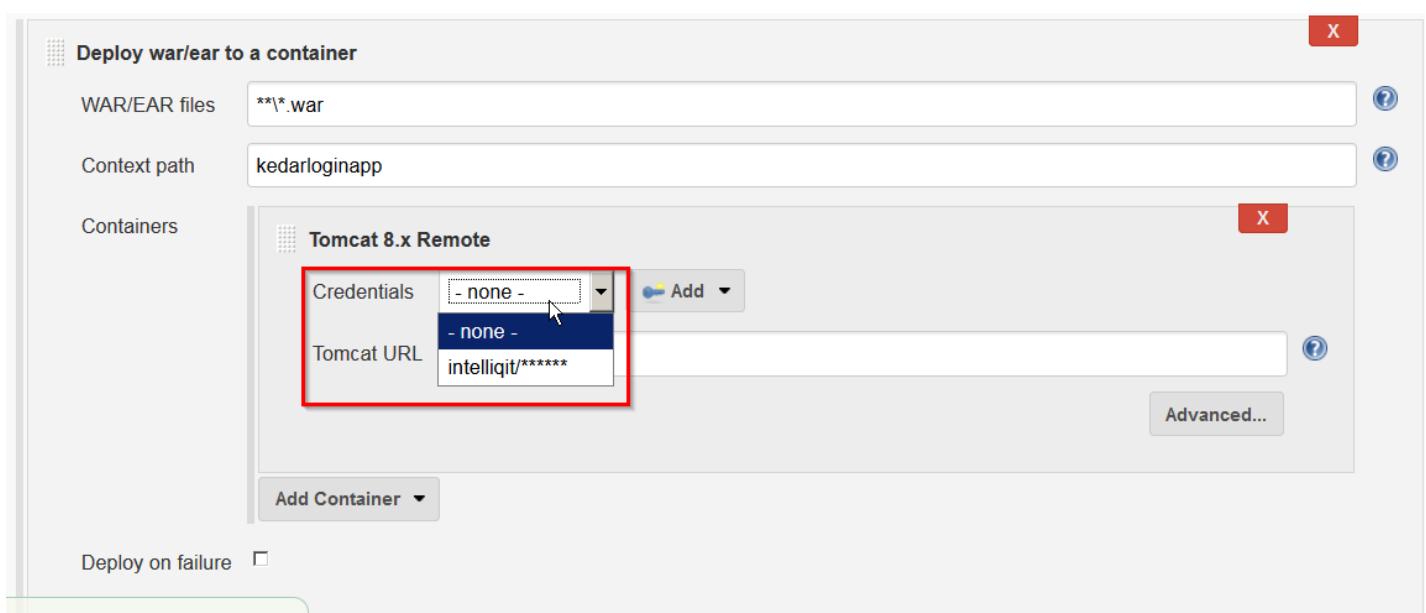
Credentials: - none -

Tomcat URL: intelliqit/*****

Advanced...

Add Container

Deploy on failure



Post-build Actions

Deploy war/ear to a container

WAR/EAR files: ***.war

Context path: kedarloginapp

Containers

Tomcat 8.x Remote

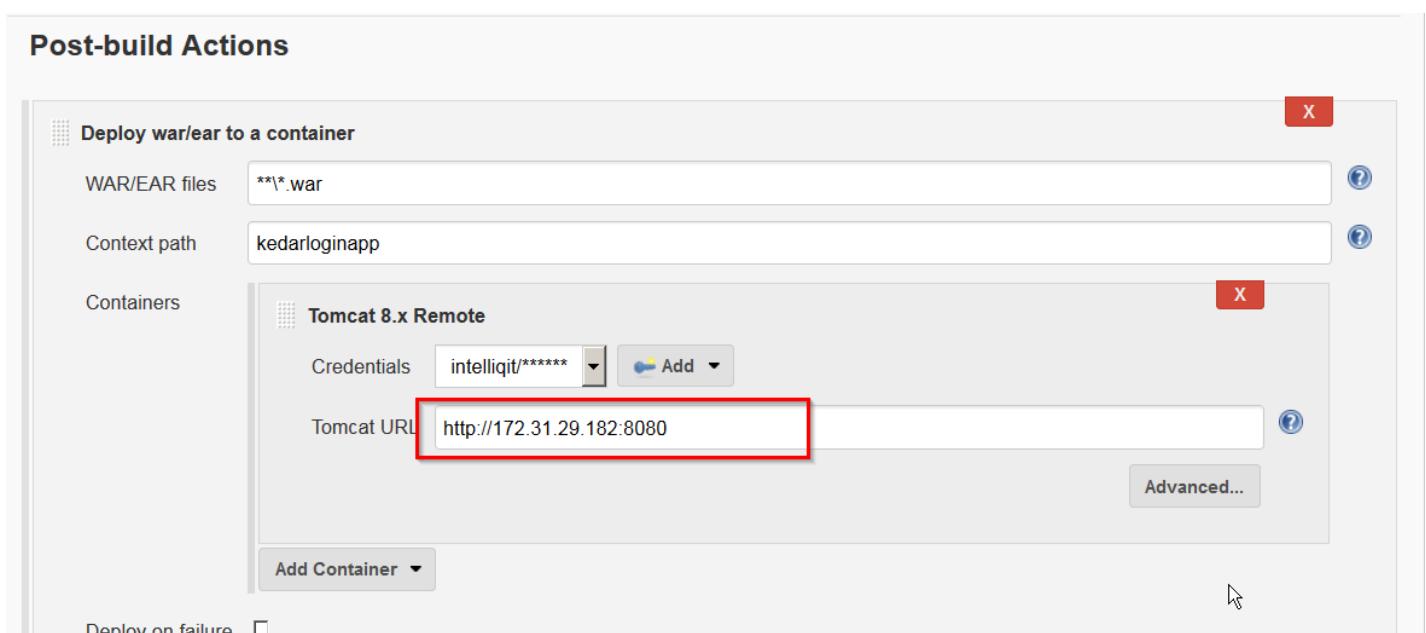
Credentials: intelliqit/*****

Tomcat URL: http://172.31.29.182:8080

Advanced...

Add Container

Deploy on failure



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

```

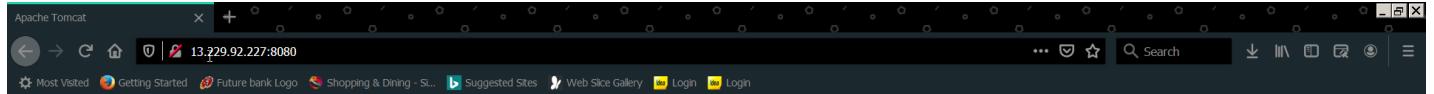
Results :

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

[[1;34mINFO[m] [[1;34mINFO[m] [lm--- [0;32mmaven-war-plugin:2.2:war[m [lm(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] [lm-----[m
[[1;34mINFO[m] [lmReactor 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] [lm-----[m
[[1;34mINFO[m] [1;32mBUILD SUCCESS[m  ↵
[[1;34mINFO[m] [lm-----[m
[[1;34mINFO[m] Total time: 3.620 s
[[1;34mINFO[m] Finished at: 2020-05-09T16:21:19Z
[[1;34mINFO[m] [lm-----[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.



It works !

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

This is the default Tomcat home page. It can be found on the local filesystem at: /var/lib/tomcat8/webapps/ROOT/index.html

Tomcat8 veterans might be pleased to learn that this system instance of Tomcat is installed with CATALINA_HOME in /usr/share/tomcat8 and CATALINA_BASE in /var/lib/tomcat8, following the rules from /usr/share/doc/tomcat8-common/RUNNING.txt.gz.

You might consider installing the following packages, if you haven't already done so:

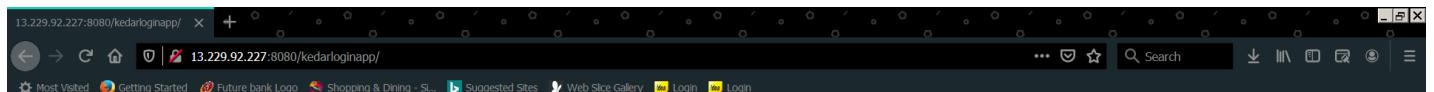
tomcat8-docs: This package installs a web application that allows to browse the Tomcat 8 documentation locally. Once installed, you can access it by clicking [here](#).

tomcat8-examples: This package installs a web application that allows to access the Tomcat 8 Servlet and JSP examples. Once installed, you can access it by clicking [here](#).

tomcat8-admin: This package installs two web applications that can help managing this Tomcat instance. Once installed, you can access the [manager webapp](#) and the [host-manager webapp](#).

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager-gui". The host-manager webapp is restricted to users with role "admin-gui". Users are defined in /etc/tomcat8/tomcat-users.xml.

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



Welcome To IntelliQ IT

Username:

Password:

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

The screenshot shows the Jenkins 'Enter an item name' dialog. In the input field, 'Testing-Job' is typed. Below the input field, there is a list of project types:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- GitHub Organization**: Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multi-branch Pipeline**: A set of Pipeline projects according to detected branches in one SCM repository.

The screenshot shows a GitHub repository page for 'intelliqittrainings/FunctionalTesting'. The repository details are as follows:

- Branch: master
- 1 commit
- 1 branch
- 0 packages
- 0 releases
- 1 contributor

On the right side, there are clone options:

- Clone with HTTPS
- Use SSH
- Use Git or checkout with SVN using the web URL: <https://github.com/intelliqittrainings/Fun>
- Open in Desktop
- Download ZIP

The repository tree shows the following files:

- .settings
- bin
- src
- .classpath
- .project
- testing.jar

<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.

Jenkins > Testing-Job > Saved

Source Code Management

General Source Code Management Build Triggers Build Environment Build Post-build Actions

None
 Git

Repositories

Repository URL: **https://github.com/intelliqitrainings/FunctionalTesting.git**

Credentials: - none - Add

Branches to build

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

Add Branch

Repository browser (Auto)

Jenkins search Kedar Pavaskar

Jenkins >

New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Lockable Resources

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Development-job	1 day 19 hr - #3	N/A	8.8 sec
		Testing-Job	N/A	N/A	N/A

Icon: S M L [Legend](#) Atom feed for all Atom feed for failures Atom feed for just latest builds

Click build

Jenkins search Kedar Pavaskar

Jenkins > Testing-Job > #1

Back to Project

Status

Changes

Console Output

View as plain text

Edit Build Information

Delete build '#1'

Git Build Data

No Tags

Console Output

```

Started by user Kedar Pavaskar
Running as SYSTEM
Building in workspace /home/ubuntu/.jenkins/workspace/Testing-Job
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/intelliqitrainings/FunctionalTesting.git
> git init /home/ubuntu/.jenkins/workspace/Testing-Job # timeout=10
Fetching upstream changes from https://github.com/intelliqitrainings/FunctionalTesting.git
> git --version # timeout=10
> git fetch --tags --progress -- https://github.com/intelliqitrainings/FunctionalTesting.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/intelliqitrainings/FunctionalTesting.git # timeout=10
> git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
> git config remote.origin.url https://github.com/intelliqitrainings/FunctionalTesting.git # timeout=10
Fetching upstream changes from https://github.com/intelliqitrainings/FunctionalTesting.git
> git fetch --tags --progress -- https://github.com/intelliqitrainings/FunctionalTesting.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin^{commit} # timeout=10
Checking out Revision 38095c8ede9e5192215ccc56b573f30223029a6 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 38095c8ede9e5192215ccc56b573f30223029a6 # timeout=10
Commit message: "a"
First time build. Skipping changelog.
Finished: SUCCESS

```

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

Jenkins > Testing-Job

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

Build Environment

- Delete workspace before build starts
- Use secret text(s) or file(s)
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- With Ant

Build

Add build step ▾

- Execute Windows batch command
- Execute shell**
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit

Save Apply

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

Testing-Job

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

Build

Execute shell

Command `java -jar /home/ubuntu/.jenkins/workspace/Testing-Job/testing.jar`

See [the list of available environment variables](#)

Advanced...

Add build step ▾

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

All

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Development-job	1 day 20 hr - #3	N/A	8.8 sec
		Testing-Job	4.7 sec - #2	N/A	3.5 sec

Icon: [S](#) [M](#) [L](#)

[add description](#)

Legend Atom feed for all Atom feed for failures Atom feed for just latest builds

Jenkins

Jenkins > Testing-Job > #2

[Back to Project](#)

[Status](#)

[Changes](#)

[Console Output](#)

[View as plain text](#)

[Edit Build Information](#)

[Delete build #2'](#)

[Git Build Data](#)

[No Tags](#)

[Previous Build](#)

Console Output

```

Started by user Kedar Pavaskar
Running as SYSTEM
Building in workspace /home/ubuntu/.jenkins/workspace/Testing-Job
No credentials specified
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/intelligittrainings/FunctionalTesting.git # timeout=10
Fetching upstream changes from https://github.com/intelligittrainings/FunctionalTesting.git
> git --version # timeout=10
> git fetch --tags --progress -- https://github.com/intelligittrainings/FunctionalTesting.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^(commit) # timeout=10
> git rev-parse refs/remotes/origin/master^(commit) # timeout=10
Checking out Revision 38095c8edea9e5192215ccc56b573f30223029a6 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 38095c8edea9e5192215ccc56b573f30223029a6 # timeout=10
Commit message: "a"
> git rev-list --no-walk 38095c8edea9e5192215ccc56b573f30223029a6 # timeout=10
[Testing-Job] $ /bin/sh -xe /tmp/jenkins3998770488875012379.sh
+ java -jar /home/ubuntu/.jenkins/workspace/Testing-Job/testing.jar
about:blank
Expected Message : Hello, World!
Actual Message :
Testing has Failed
Finished: SUCCESS

```

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.

The screenshot shows the Jenkins Job configuration interface for a 'Development-job'. The 'Post-build Actions' tab is selected. A specific action, 'Build other projects', is highlighted with a red box. Other actions listed include Deploy war/ear to a container, Aggregate downstream test results, Archive the artifacts, Publish JUnit test result report, 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], and Delete workspace when build is done. At the bottom, there are 'Save' and 'Apply' buttons.

Post-build Actions

Build other projects

Projects to build: Testing-Job Testing-Job

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Deploy war/ear to a container

WAR/EAR files: ***.war

Context path: kedarloginapp

Containers

Tomcat 8.x Remote

Credentials: intelliqit/***** intelliqit/*****

Tomcat URL: http://172.31.29.182:8080

Deploy on failure

Save **Apply**

Mentioned “testing-job” to link the jobs.

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

All	W	Name	Last Success	Last Failure	Last Duration	
		Development-job	2 min 4 sec - #5	N/A	8.4 sec	
		Testing-Job	1 min 46 sec - #4	N/A	3.2 sec	

Icon: S M L

[add description](#)

[Legend](#) [Atom feed for all](#) [Atom feed for failures](#) [Atom feed for just latest builds](#)

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

The screenshot shows the Jenkins Plugin Manager interface. At the top, there are tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. A search bar and user information are also at the top. The main area lists plugins under the 'Available' tab. One plugin, 'Copy Artifact' by Scriptler, is highlighted with a red box. Below this, a warning message is displayed in a pink box: 'Warning: This plugin version may not be safe to use. Please review the following security notices.' It lists a single item: '• Password stored in plain text'.

Install ↓	Name	Version
<input type="checkbox"/> Scriptler	Scriptler allows you to store/edit/execute Groovy scripts on any of the slaves/nodes... no need to copy paste Groovy code anymore. Beside administer your scripts, Scriptler also provides a way to share scripts between users via hosted script catalogs on the internet.	3.1
<input checked="" type="checkbox"/> Copy Artifact	Adds a build step to copy artifacts from another project.	1.44
<input type="checkbox"/> Jobcopy Builder	Copy a job in build steps.	1.4.0
<input type="checkbox"/> Copy data to workspace	This plugin copies data to workspace directory before each build. After build data will be deleted.	1.0
<input type="checkbox"/> Pipeline_Phoenix AutoTest	This plugin provide a set of DevOps pipeline step, i.e CopyStep let you copy a file from master to current node. We also have TimeStep, FtpStep, JdbcStep, FtpStep, DiskStep, WithSCMStep.	1.3
<input type="checkbox"/> CX Copy Data Management	This plugin integrates with Catalogic copy data management platform and facilitates the deployment of an virtual machine, application or storage volume as an optional step of a build or pipeline.	1.9
Warning: This plugin version may not be safe to use. Please review the following security notices:		
• Password stored in plain text		

Buttons at the bottom include 'Install without restart', 'Download now and install after restart', 'Update information obtained: 4 hr 44 min ago', and 'Check now'.

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

The screenshot shows the Jenkins dashboard with a table of projects. The 'Development-job' project is selected, and a context menu is open over its name. The menu items are: Changes, Workspace, Build Now, Delete Project, Configure (which is highlighted with a blue background), and Rename.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		Development-job	4 hr 24 min - #5	N/A	8.4 sec
			4 hr 24 min - #4	N/A	3.2 sec

At the bottom, there are links for 'Legend', 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'.

Jenkins > **Development-job** >

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Deploy war/ear to a container
WAR/EAR files `***.war`

Context path `kedarloginapp`

Aggregate downstream test results

Archive the artifacts

Build other projects
Publish JUnit test result report
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]
Delete workspace when build is done

Add post-build action ▾

Save Apply

The screenshot shows the Jenkins job configuration for 'Development-job'. The 'Post-build Actions' tab is selected. A dropdown menu is open over the 'Archive the artifacts' option, listing various actions like 'Deploy war/ear to a container' and 'Archive the artifacts'. The 'Archive the artifacts' option is highlighted with a red box.

Post-build Actions

Archive the artifacts
Files to archive `***.war`

Build other projects
Projects to build Testing-Job
 Trigger only if build is stable
 Trigger even if the build is unstable
 Trigger even if the build fails

Deploy war/ear to a container
WAR/EAR files `***.war`
Context path `kedarloginapp`

Containers
Tomcat 8.x Remote
Credentials intelliqit***** Add

Save Apply

This screenshot shows the expanded 'Archive the artifacts' section. It includes a field for 'Files to archive' containing the pattern `***.war`. Below it, the 'Build other projects' section is shown with 'Testing-Job' selected and three trigger options: 'Trigger only if build is stable' (which is checked), 'Trigger even if the build is unstable', and 'Trigger even if the build fails'. The 'Deploy war/ear to a container' section is also visible at the bottom.

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.

Jenkins > Testing-Job

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

Build

Execute shell

Command: `java -jar /home/ubuntu/.jenkins/workspace/Testing-Job/testing.jar`

See [the list of available environment variables](#)

Advanced...

Add build step ▾

Copy artifacts from another project

Execute Windows batch command

Execute shell

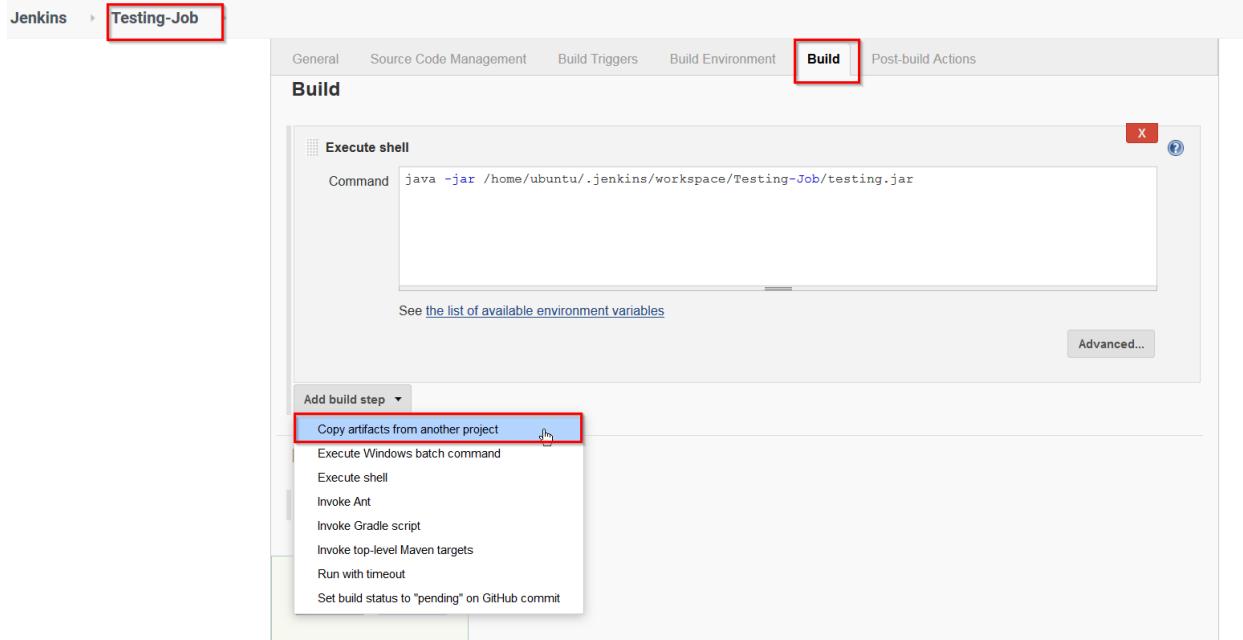
Invoke Ant

Invoke Gradle script

Invoke top-level Maven targets

Run with timeout

Set build status to "pending" on GitHub commit



Testing-Job

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

Build

Execute shell

Command: `java -jar /home/ubuntu/.jenkins/workspace/Testing-Job/testing.jar`

See [the list of available environment variables](#)

Advanced...

Copy artifacts from another project

Project name: De

Development.job

Which build: Latest successful build

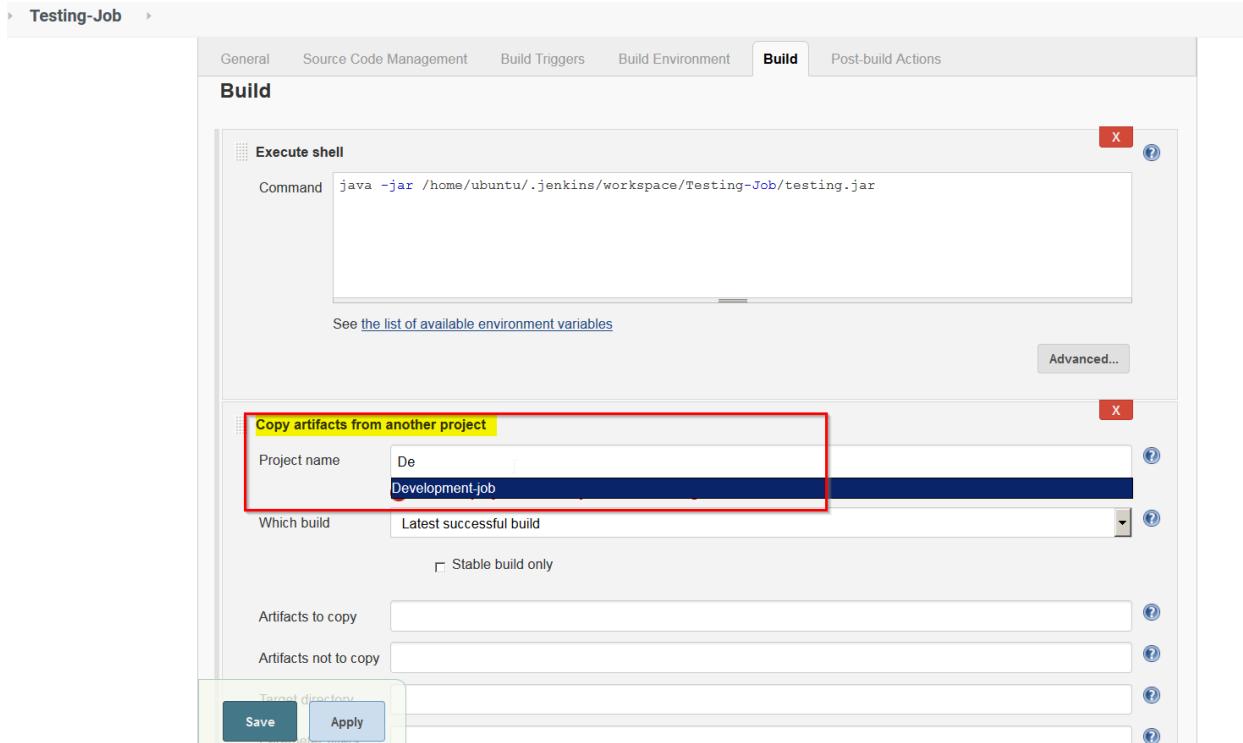
Stable build only

Artifacts to copy:

Artifacts not to copy:

Target directory:

Save Apply



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

The screenshot shows the Jenkins dashboard with two projects listed: 'Development-job' and 'Testing-Job'. A context menu is open over the 'Testing-Job' project, with the 'Configure' option highlighted and surrounded by a red box.

The screenshot shows the 'Testing-Job' configuration page with the 'Post-build Actions' tab selected. A large red box highlights the 'Deploy war/ear to a container' section, which includes fields for 'WAR/EAR files' (set to '**/*'), 'Context path' (set to 'productionURL'), and 'Containers' (selected 'Tomcat 8.x Remote' with 'Credentials' set to 'intelligit*****'). Below this, the 'Add post-build action' button is also highlighted with a red box.

Save and goto the dashboard of Jenkins.

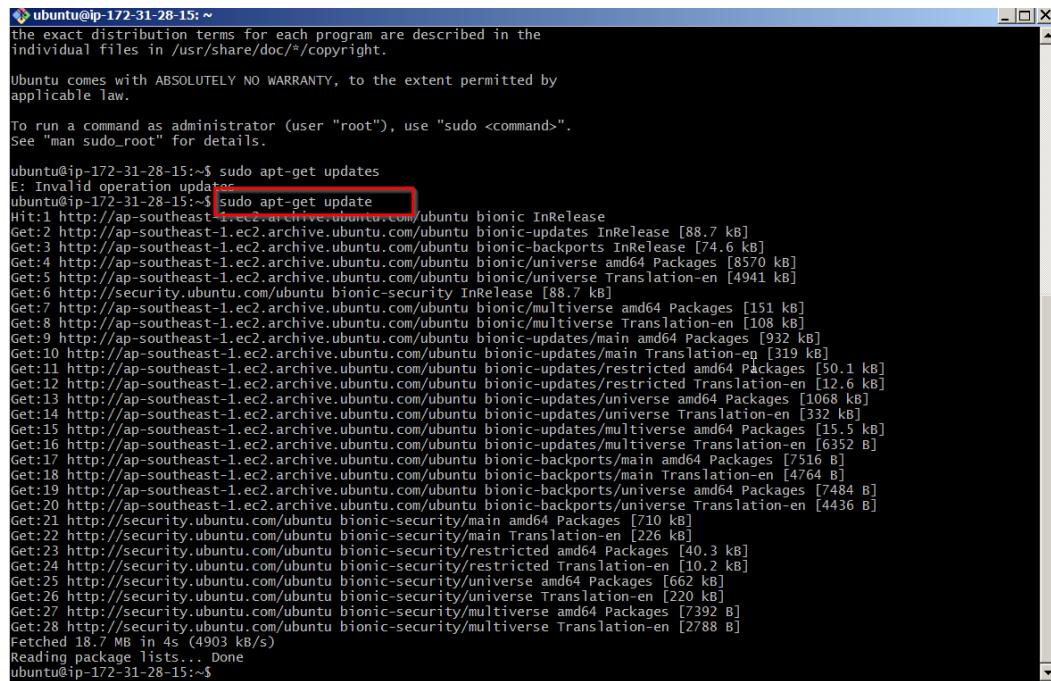
Now Running the Development Job Will Perform the First three stages of the CI/CD and the job will continue to call the Testing Job and perform the remaining stages of CI/CD.

Alternative (permanent) Methods of Jenkins Installation.

Setting up Jenkins as a permanent installation

1. Update the apt repository

`sudo apt-get update`



```
ubuntu@ip-172-31-28-15:~$ sudo apt-get update
[sudo] password for ubuntu:
E: Invalid operation update
ubuntu@ip-172-31-28-15:~$ sudo apt-get update
Hit:1 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [8570 kB]
Get:5 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/universe Translation-en [4941 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:7 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages [151 kB]
Get:8 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic/multiverse Translation-en [108 kB]
Get:9 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages [932 kB]
Get:10 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/main Translation-en [319 kB]
Get:11 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages [50.1 kB]
Get:12 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/restricted Translation-en [12.6 kB]
Get:13 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages [1068 kB]
Get:14 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/universe Translation-en [332 kB]
Get:15 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages [15.5 kB]
Get:16 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/multiverse Translation-en [6352 B]
Get:17 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages [7516 B]
Get:18 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports/main Translation-en [4764 B]
Get:19 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages [7484 B]
Get:20 http://ap-southeast-1.ec2.archive.ubuntu.com/ubuntu bionic-backports/universe Translation-en [4436 B]
Get:21 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [710 kB]
Get:22 http://security.ubuntu.com/ubuntu bionic-security/main Translation-en [226 kB]
Get:23 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [40.3 kB]
Get:24 http://security.ubuntu.com/ubuntu bionic-security/restricted Translation-en [10.2 kB]
Get:25 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages [662 kB]
Get:26 http://security.ubuntu.com/ubuntu bionic-security/universe Translation-en [220 kB]
Get:27 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [7392 B]
Get:28 http://security.ubuntu.com/ubuntu bionic-security/multiverse Translation-en [2788 B]
Fetched 18.7 MB in 4s (4903 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-28-15:~$
```

2. Install java

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

3. Add the repository key to the system

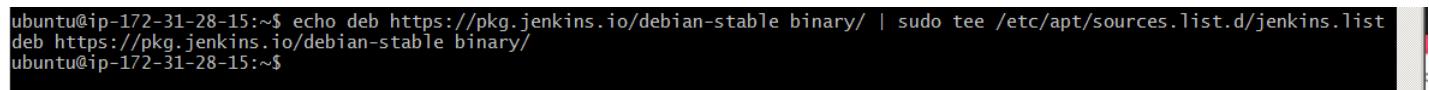
`wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -`



```
ubuntu@ip-172-31-28-15:~$ wget -q -O - https://pkg.jenkins.io/debian/jenkins-ci.org.key | sudo apt-key add -
OK
```

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`



```
ubuntu@ip-172-31-28-15:~$ echo deb https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list
deb https://pkg.jenkins.io/debian-stable binary/
ubuntu@ip-172-31-28-15:~$
```

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

General Source Code Management **Build Triggers** Build Environment Build Post-build Actions

Build Triggers

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

Build after other projects are built (?)

Build periodically (?)

Schedule

No schedules so will never run

This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:
MINUTE HOUR DOM MONTH DOW
MINUTE Minutes within the hour (0–59)
HOUR The hour of the day (0–23)
DOM The day of the month (1–31)
MONTH The month (1–12)
DOW The day of the week (0–7) where 0 and 7 are Sunday.

General Source Code Management **Build Triggers** Build Environment Build Post-build Actions

Build Triggers

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

Build after other projects are built (?)

Build periodically (?)

Schedule **48 8 16 5 6** (cursor)

Spread load evenly by using 'H 8 16 5 6' rather than '48 8 16 5 6'
This schedule will match dates only rarely (e.g. February 29) or never (e.g. June 31), so this job may be triggered very rarely, if at all.

This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:
MINUTE HOUR DOM MONTH DOW
MINUTE Minutes within the hour (0–59)
HOUR The hour of the day (0–23)
DOM The day of the month (1–31)
MONTH The month (1–12)
DOW The day of the week (0–7) where 0 and 7 are Sunday.

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.

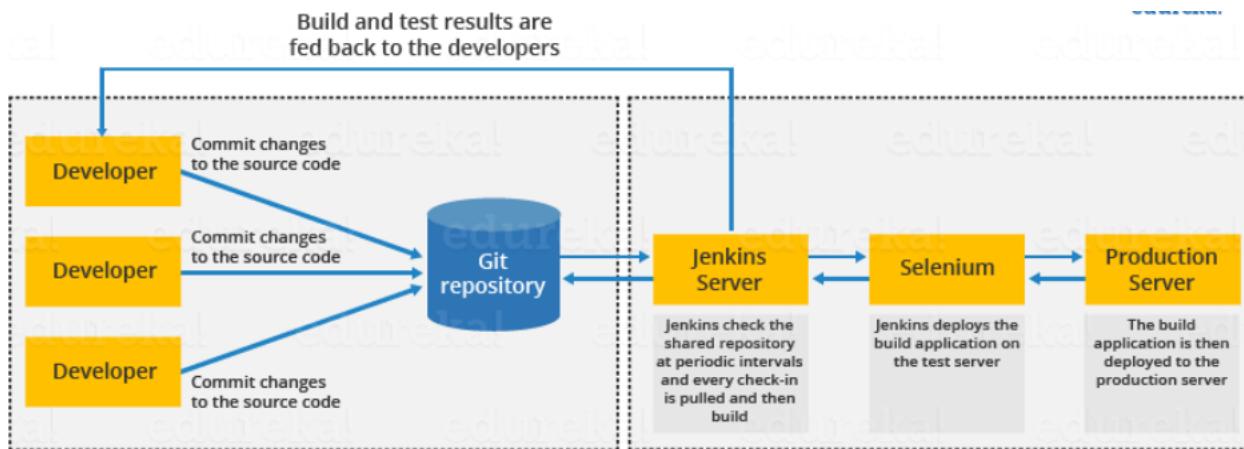
Scheduling the Jenkins job to trigger when developers make any new commits to git

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 screenshot shows the Jenkins configuration interface for a job named "Development-job". The "Build Triggers" tab is selected. Under the "Poll SCM" section, the "Schedule" field contains the cron expression "* * * * *". A red box highlights this field. A tooltip below the field provides information about cron syntax, stating: "This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace: MINUTE HOUR DOM MONTH DOW". It also lists the meanings of these fields: MINUTE (Minutes within the hour 0–59), HOUR (The hour of the day 0–23), DOM (The day of the month 1–31), MONTH (The month 1–12), and DOW (The day of the week 0–7 where 0 and 7 are Sunday). Below the tooltip, instructions for specifying multiple values are provided: "* specifies all valid values" and "M–N specifies a range of values".

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

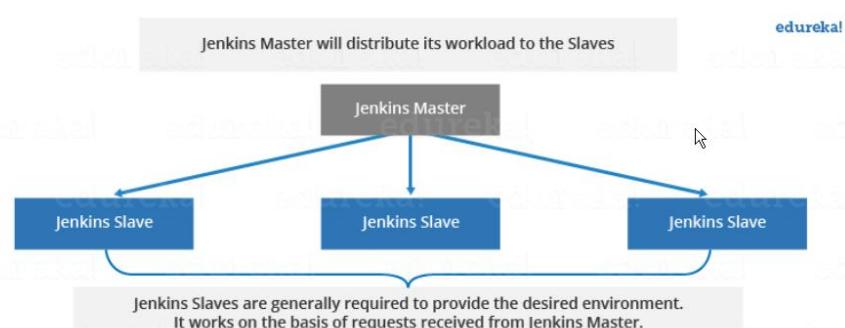
Your main Jenkins server is the Master. The Master's job is to handle:

- Schedule Build Jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves, 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

```
ubuntu@ip-172-31-16-113: ~
e /usr/bin/wsgen (wsgen) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jcmd to p
/usr/bin/jcmd (jcmd) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jarsigner
provide /usr/bin/jarsigner (jarsigner) in auto mode
Setting up openjdk-8-jre:amd64 (8u252-b09-1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/polic
to provide /usr/bin/policytool (policytool) in auto mode
Setting up openjdk-8-jdk:amd64 (8u252-b09-1~18.04) ...
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/appletvie
provide /usr/bin/appletviewer (appletviewer) in auto mode
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jconsole
vide /usr/bin/jconsole (jconsole) in auto mode
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.36.11-2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
ubuntu@ip-172-31-16-113: ~$
```

- 2) Setup the password for the default user

```
sudo passwd ubuntu
```

```
ubuntu@ip-172-31-16-113: ~$ sudo passwd ubuntu
Enter new UNIX password: [REDACTED]
Retype new UNIX password: [REDACTED]
passwd: password updated successfully
ubuntu@ip-172-31-16-113: ~$
```

- 3) Edit the sshd_config file

```
sudo vim /etc/ssh/sshd_config
```

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

```
ubuntu@ip-172-31-16-113: ~
GNU nano 2.9.3                               /etc/ssh/sshd_config                         Mc

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile      .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no
```

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:
SHA256:0Ajvx3v4xrrwFGjyZEsE+vcf91EMzr2a9Y4BmLPhJc ubuntu@ip-172-31-18-83
The key's randomart image is:
+---[RSA 2048]----+
| . . . + |
| + 0. . + |
| . =+. + . |
| o.+*o....o |
| .BS.=.E .. |
| .o*.* . + |
| 0= + ==. |
| .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

The screenshot shows the AWS EC2 console interface. In the left sidebar, there are filters for 'Instances', 'Launch Type', 'Status', and 'Tags'. The main area displays a table of instances. One instance, 'Jenkins Slave 1', is highlighted with a red box. The table columns include Name, Instance ID, Instance Type, Availability Zone, Instance State, Status Checks, Alarm Status, Public DNS (IPv4), IPv4 Public IP, and IP. The 'Jenkins Slave 1' instance is listed as t2.micro, running in ap-southeast-1b, with a green status check and no alarm status. Its public DNS is ec2-18-141-177-229.ap-southeast-1.compute.amazonaws.com and its IPv4 public IP is 18.141.177.229.

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`

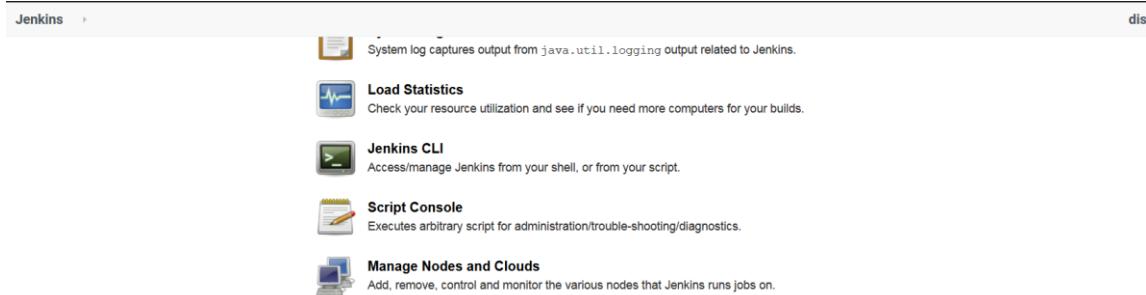
```
ubuntu@ip-172-31-16-113:~$ ls -l
total 1488
-rw-rw-r-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$ chmod u+x slave.jar
ubuntu@ip-172-31-16-113:~$
```

Before updating the permissions

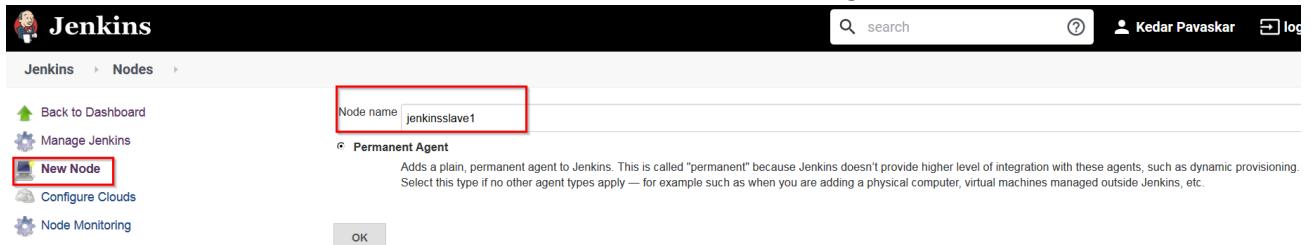
```
-rw-rw-r-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$ chmod u+x slave.jar
ubuntu@ip-172-31-16-113:~$ ls -l
total 1488
-rwxrwxr-- 1 ubuntu ubuntu 1522914 May  9 11:48 slave.jar
ubuntu@ip-172-31-16-113:~$
```

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.

Name	jenkinsslave1
Description	
# of executors	1

10) Launch Method: Select Launch agent via execution of command on master

Launch command: ssh ubuntu@private_ip_of_slave java -jar slave.jar

Name: jenkinsslave1

Description:

of executors: 1

Remote root directory: /home/ubuntu/kedarsworkspace

Labels: myslave

Usage: Use this node as much as possible

Launch method: Launch agent via execution of command on the master

Launch command: ssh ubuntu@172.31.16.113 java -jar slave.jar

Availability: Keep this agent online as much as possible

Node Properties

- Disable deferred wipeout on this node
- Environment variables
- Tool Locations

Save

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	jenkinsslave1	Linux (amd64)	In sync	6.02 GB	- 0 B	6.02 GB	67ms
	master	Linux (amd64)	In sync	5.16 GB	- 0 B	5.16 GB	0ms
	Data obtained	2 min 30 sec	2 min 30 sec	2 min 29 sec	2 min 29 sec	2 min 29 sec	2 min 30 sec

Refresh status

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

Jenkins > Development-job >

General Source Code Management Build Triggers Build Environment Build Post-build Actions

I. Permission to Copy Artifact

- This build requires lockable resources
- This project is parameterized
- Throttle builds
- Disable this project
- Execute concurrent builds if necessary
- Restrict where this project can be run

Label Expression: myslave

Label myslave is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

Advanced...

13) Enter slave label: myslave

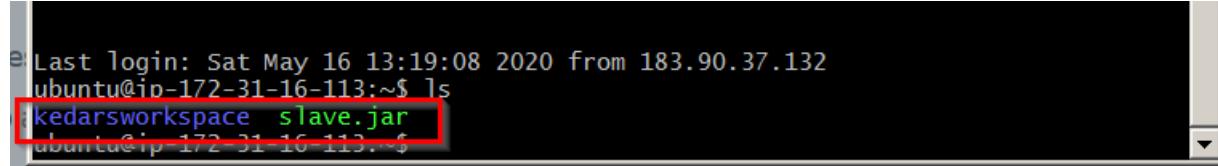
14) 18 Apply--->Save

15) Now run the Job you just edited to be run on slave. You will notice the below.

Build Executor Status

	master	1 Idle	2 Idle
	jenkinsslave1	1 Development.job	#13

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



```
Last login: Sat May 16 13:19:08 2020 from 183.90.37.132
ubuntu@ip-172-31-16-113:~$ ls
kedarsworkspace  slave.jar
ubuntu@ip-172-31-16-113:~$
```

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 pattern 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  
    }  
}
```

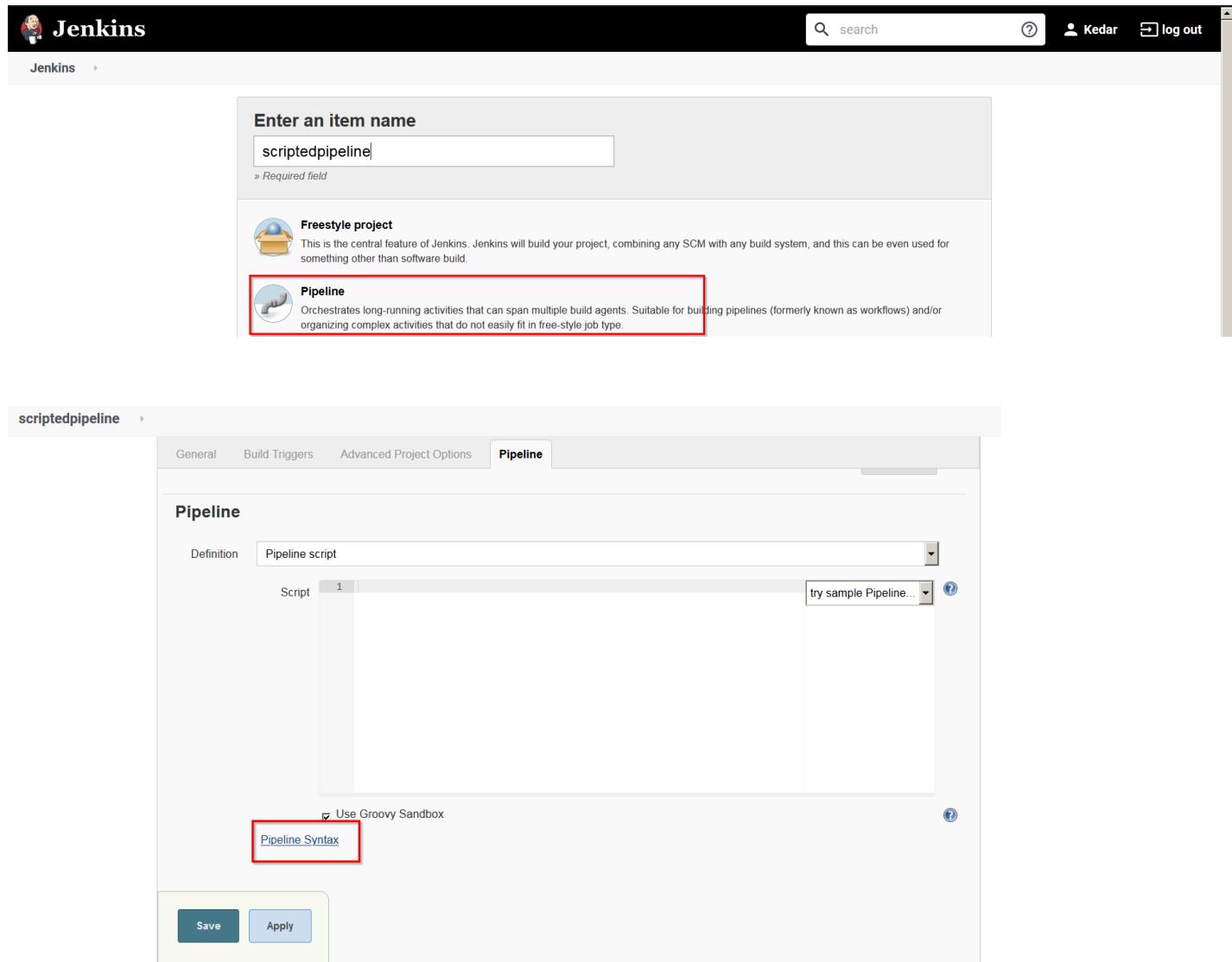
Sample - Declarative Pipeline

```
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.



The screenshot shows the Jenkins interface for creating a new pipeline project named "scriptedpipeline". The "Pipeline" tab is selected. In the "Definition" section, "Pipeline script" is chosen. Below it, there is a large text area for the Groovy script, which is currently empty. A red box highlights the "Pipeline Syntax" link located just above the script editor. At the bottom, there are "Save" and "Apply" buttons.

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

Jenkins > scriptedpipeline > Pipeline Syntax

[Back](#)

[Snippet Generator](#)

[Declarative Directive Generator](#)

[Declarative Online Documentation](#)

[Steps Reference](#)

[Global Variables Reference](#)

[Online Documentation](#)

[Examples Reference](#)

[IntelliJ IDEA GDSL](#)

Overview

This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step node: Allocate node

Label master

Label master is serviced by 1 node. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

Generate Pipeline Script

```
node('master') {
    // some block
}
```

Global Variables

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.

Steps

Sample Step stage: Stage

Stage Name Continuous Download

Generate Pipeline Script

```
stage('Continuous Download') {
    // some block
}
```

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

General Build Triggers Advanced Project Options **Pipeline**

Pipeline

Definition Pipeline script

Script

```
1 node('master')
2 {
3     stage('ContinuousDownload')
4     {
5         git 'https://github.com/selenium-saikrishna/maven.git'
6     }
}
```

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

```
1 node('master')
2 {
3     stage('ContinuousDownload')
4     {
5         git 'https://github.com/selenium-saikrishna/maven.git'
6     }
7     stage('ContinuousBuild')
8     {
9         sh label: '', script: 'mvn package'
10    }
11 }
```

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.

Definition Pipeline script

```
1 node('master')
2 {
3     stage('ContinuousDownload')
4     {
5         git 'https://github.com/selenium-saikrishna/maven.git'
6     }
7     stage('ContinuousBuild')
8     {
9         sh label: '', script: 'mvn package'
10    }
11 stage('ContinuousDeployment')
12 {
13     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.23.16'
14 }
15 }
```

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.

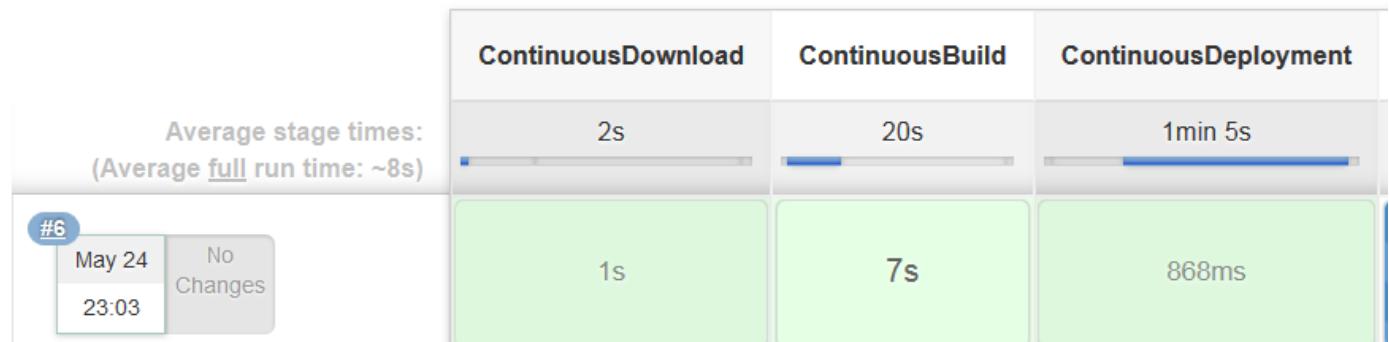
Run the job, and you would see the pipeline steps being executed successfully as below.

Pipeline scriptedpipeline



[Recent Changes](#)

Stage View



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/intelijittrainings/FunctionalTesting.git>

generate the groovy code for the same.

```
4 * {  
5     git 'https://github.com/selenium-saikrishna/maven.git'  
6 }  
7 stage('ContinuousBuild')  
8 * {  
9     sh label: '', script: 'mvn package'  
10 }  
11 stage('ContinuousDeployment')  
12 * {  
13     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.23.14'  
14 }  
15 stage('ContinuousTesting')  
16 * {  
17     git 'https://github.com/intelijittrainings/FunctionalTesting.git'  
18 }  
19 }
```

Use Groovy Sandbox

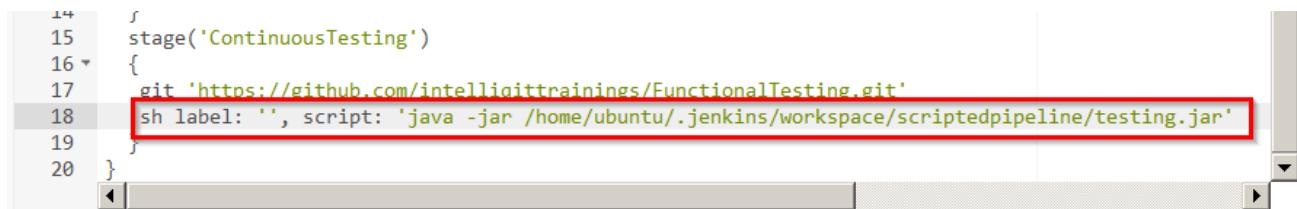
[Pipeline Syntax](#)

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.

```
ubuntu@ip-172-31-23-80: ~/jenkins/workspace/scriptedpipeline  
43 packages can be updated.  
33 updates are security updates.  
  
New release '18.04.4 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Sun May 24 14:50:26 2020 from 39.109.179.124  
ubuntu@ip-172-31-23-80:~$ cpwd  
No command 'cpwd' found, did you mean:  
Command 'hpwd' from package 'hfsutils' (main)  
Command 'pwd' from package 'coreutils' (main)  
cpwd: command not found  
ubuntu@ip-172-31-23-80:~$ pwd  
/home/ubuntu  
ubuntu@ip-172-31-23-80:~$ cd /.jenkins  
-bash: cd: /.jenkins: No such file or directory  
ubuntu@ip-172-31-23-80:~$ cd /.jenkins/  
-bash: cd: /.jenkins/: No such file or directory  
ubuntu@ip-172-31-23-80:~$ cd /home/ubuntu/.jenkins/workspace/scriptedpipeline  
ubuntu@ip-172-31-23-80:~/jenkins/workspace/scriptedpipeline$ ls  
bin server src testing.jar webapp  
ubuntu@ip-172-31-23-80:~/jenkins/workspace/scriptedpipeline$ |
```

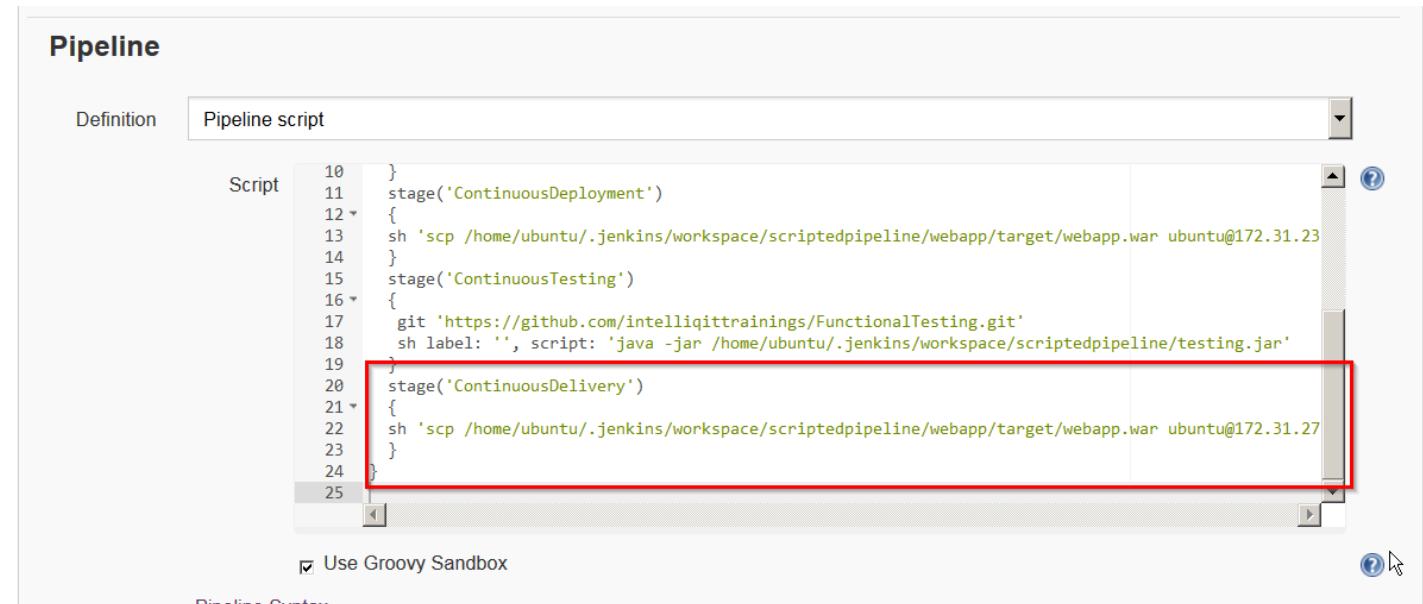
Furthermore, we need to generate a groovy code for executing the testing script.



```
14
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.



Pipeline

Definition Pipeline script

Script

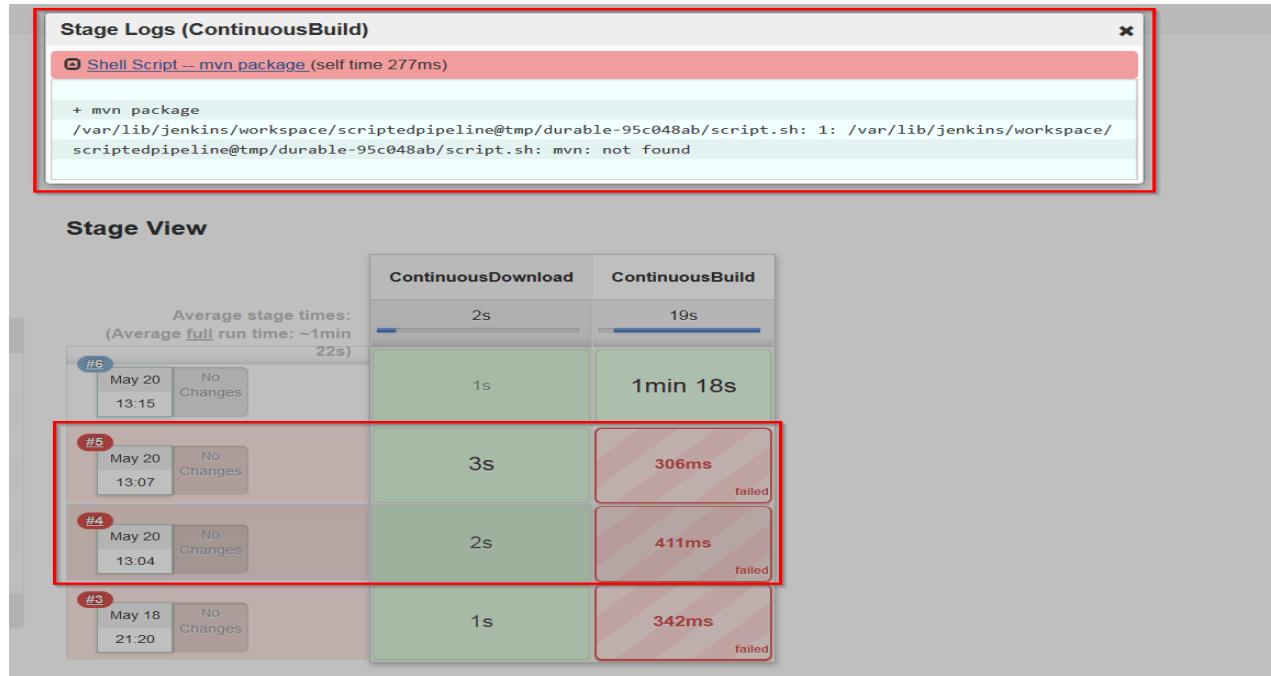
```
10 }
11 stage('ContinuousDeployment')
12 {
13     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.23'
14 }
15 stage('ContinuousTesting')
16 {
17     git 'https://github.com/intelliqittrainings/FunctionalTesting.git'
18     sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar'
19 }
20 stage('ContinuousDelivery')
21 {
22     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.27'
23 }
24 }
25 }
```

Use Groovy Sandbox

Pipeline Syntax

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.

Enter an item name

Declarative Pipeline
» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations such as testing on multiple environments, platform-specific builds, etc.

Pipeline

Definition Pipeline script

Script

```
1 pipeline
2 {
3     agent any
4 }
```

try sample 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 master or slave.

Stage1 – continuous Download

Pipeline

Definition Pipeline script

Script

```
1 pipeline
2 {
3     agent any
4     Stages
5     {
6         Stage('ContinuousDownload')
7         {
8             Steps
9             {
10                git 'https://github.com/intelliqittrainings/maven.git'
11            }
12        }
13    }
14 }
15 }
```

try sample Pipeline...

Use Groovy Sandbox

[Pipeline Syntax](#)

Notice the syntax is drastically different from the scripted pipeline.

Stage2 – continuous Build

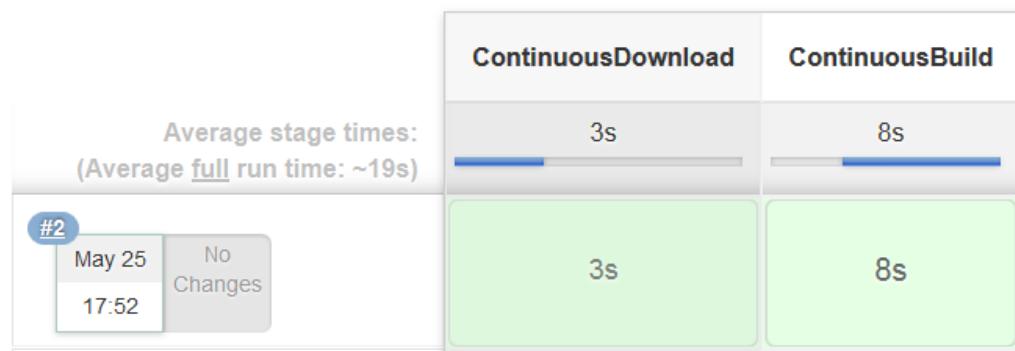
Pipeline

Definition Pipeline script

```
Script
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

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

Stage View



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.

Pipeline

Definition Pipeline script

```
14 *
15
16 *
17
18
19
20
21 *
22
23 *
24
25
26
27
28
29 }
```

Script

```
steps
{
    sh label: '', script: 'mvn package'
}
stage('ContinuousDeployment')
{
    steps
{
    sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@192.168.1.100:/var/www/html'
}
```

Use Groovy Sandbox

[Pipeline Syntax](#)

Stage3 – Continuous Deployment

Definition Pipeline script

```
14 *
15
16 *
17
18
19
20
21 *
22
23 *
24
25
26
27
28
29 }
```

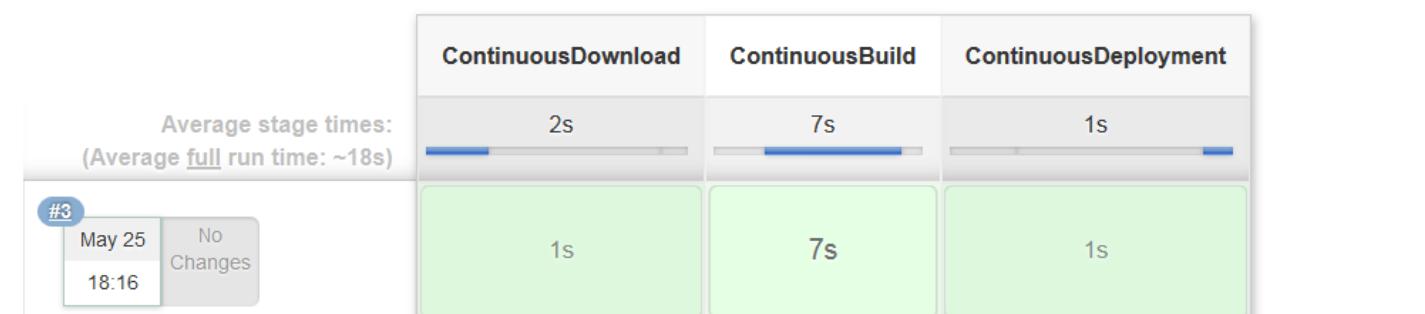
Script

```
steps
{
    sh label: '', script: 'mvn package'
}
stage('ContinuousDeployment')
{
    steps
{
    sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@192.168.1.100:/var/www/html'
}
```

Use Groovy Sandbox

[Pipeline Syntax](#)

Stage View



Stage4 – Continuous Testing

Pipeline

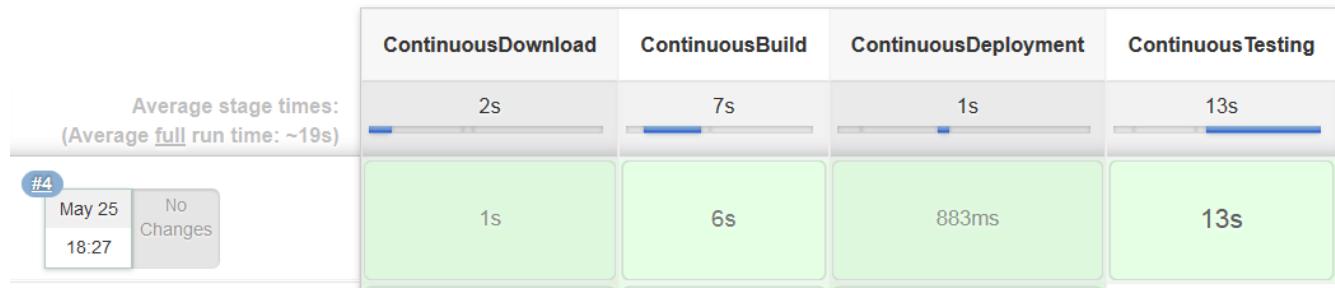
Definition Pipeline script

```
Script
22 steps
23 {
24     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.1.11:~/webapp'
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 }
```

Use Groovy Sandbox 

[Pipeline Syntax](#) 

Stage View



Stage5 – Continuous Delivery

Pipeline

Definition Pipeline script

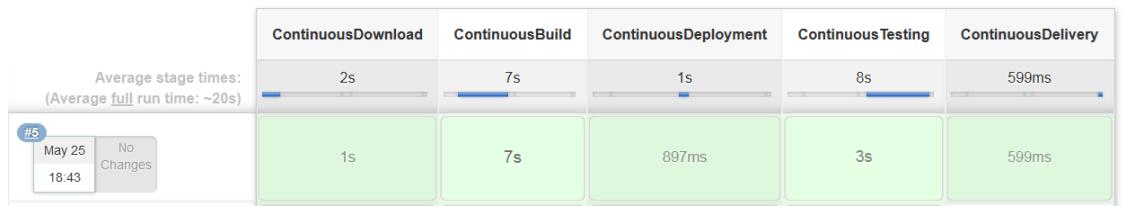
```
Script
29 steps
30 {
31     git 'https://github.com/intelliqittrainings/FunctionalTesting.git'
32     sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar'
33 }
34 }
35 stage('ContinuousDelivery')
36 {
37     steps
38 {
39     sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.1.11:~/webapp'
40 }
41 }
42 }
43 }
44 }
```

Use Groovy Sandbox 

[Pipeline Syntax](#) 



Stage View



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

Overview

This Snippet Generator will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step input: Wait for interactive input

Message Waiting Approval for Final production Deployment from Delivery Manager

Custom ID

OK Button Caption

Allowed Submitter kumar

Parameter to store the approving submitter

Parameters Add

Pipeline

Definition Pipeline script

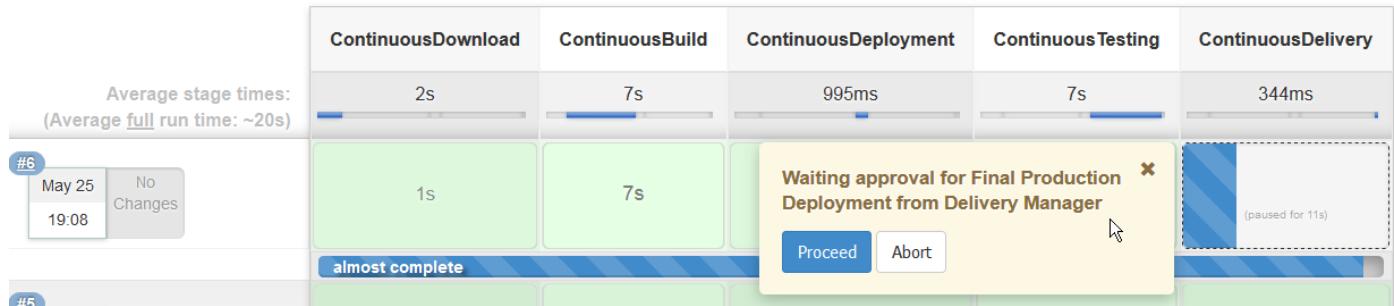
Script

```
27
28    stage('ContinuousTesting')
29    {
30        steps
31        {
32            git 'https://github.com/inteliquitrainings/FunctionalTesting.git'
33            sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/scriptedpipeline/testing.jar'
34        }
35    }
36
37
38
39    stage('ContinuousDelivery')
40    {
41        steps
42        {
43            input message: 'Waiting Approval for Final production Deployment from Delivery Manager', submitter: 'kumar'
44            sh 'scp /home/ubuntu/.jenkins/workspace/scriptedpipeline/webapp/target/webapp.war ubuntu@172.31.27.212:/var/lib/tomcat8/webapps/dpprodapp.war'
45        }
46    }
```

Use Groovy Sandbox

[Pipeline Syntax](#)

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.