Jenkins Lab Instructions

Pre-Requisites: -

1.) Jenkins installed on Server 1(Ubuntu 16) in AWS. This is Jenkins Master Server, Build Server. Jenkins URL: http://54.88.34.215:8080/

Note: Security Group Inbound Rules should be opened for the required ports and IPs from where it is being accessed.

2.) Tomcat installed on Server 2(Ubuntu 16) in AWS. This is our Deployment Server. Tomcat URL: http://34.192.236.63:9090/

Note 1: Security Group Inbound Rules should be opened for the required ports and IPs from where it is being accessed.

Note 2: - After installing tomcat, we have to do some modifications in some files in tomcat server level. The changes to be made mentioned in **Configuring Tomcat to support deployment through Jenkins**. Already done for this server.

3.) Git Repository to be deployed should be available. The git repo to be used in training: https://github.com/rajnikhattarrsinha/java-tomcat-maven-example

4.) Jenkins user to be created for each user.

Lab 1: Sample Free Style Job

1. Login using your credentials in Jenkins on browser using URL: http://<jenkins master server ip>:8080/
2. Click **New Item** link on left panel
3. **Enter an Item name** like <yourname>_firstjob
4. Select **Freestyle project**
5. Click **OK**

Job Details screen should be displayed.

6. Enter **Description** like “This is first free style project created by <Your name>”
7. Scroll down, under **Build** section, Select Add Build Step=”Execute Shell”
8. Enter **Command** as echo “Hello <Your Name>, Welcome to Jenkins world !!”
9. Click Save

Congratulations!! Your First Jenkins Project is created.

10. On Left Panel, click Build Now;
11. Click the link **“# 1”** under left panel section **Build History**
12. Click **Console Output** link
13. View the log.

It should display your hello message and Finished: SUCCESS

Lab 2: Repeatable Job

1. Click **New Item** link on left panel
2. **Enter an Item name** like <yourname>_repeatablejob

3. Select **Freestyle project**
4. Click **OK**

Job Details screen should be displayed.

5. Enter **Description** like “This is a Repeatable job created by <your name>”
6. Scroll Down, under **Build Triggers** section, Select **Build periodically** checkbox.
7. Enter **Schedule** as “H/02 * * * *”
8. Scroll down, under **Build** section, Select Add Build Step=”Execute Shell”
9. Enter **Command** as echo “Hello <Your Name>, This is repeatable job, will run in defined interval !!”
10. Click **Save.**
11. Note the current System time and then view the **Build History** section

The Build History section should display the build triggered in just 2 minutes from your current system time. The job will trigger in every 2 minutes .

14. Click the link **“# 1”** under left panel section **Build History**
15. Click **Console Output** link
16. View the log.

Lab 3: Parameterized Job

1. Click **New Item** link on left panel
2. **Enter an Item name** like <yourname>_parametrizedjob
3. Select **Freestyle project**
4. Click **OK**

Job Details screen should be displayed

5. Enter **Description** like “This is a parameterized job created by <your name>”
6. Select **This project is parameterized** checkbox
7. Click **Add Parameter** and select **String Parameter**
8. Under String Parameter screen, Enter Name as “name”
9. Enter Default Value as “test”
10. Select **Trim the string**
11. Scroll down, under **Build** section, Select Add Build Step=”Execute Shell”
12. Enter **Command** as echo “Hello <Your Name>, This is parameterized Job !! , Displaying the parameter as ${name}”
13. Click **Save**
14. Note that **Build Now** is now changed to **Build with parameters**
15. Click on **Build with parameters**
16. On right panel, it is showing the name of the parameter and the default value we provided in configuration.

17. Enter **name** as “training” as replace the default value ‘test’
18. Click **Build**
19. Click the link **“# 1”** under left panel section **Build History**
20. Click **Console Output** link
21. View the log.

It should display your configured message and Finished: SUCCESS

.

Lab 4: SMTP Notifications

Pre-requisites:

1. Jenkins should already be configured to support following features and tools:-
    a.) Email Notification (Steps used mentioned in **Configuring Jenkins for SMTP Notifications)**
2. Jenkins already have following plugins installed:-
    Email Extension : It allows to configure every aspect of email notifications: when an email is sent, who should receive it and what the email says

*Note: To know how to install plugin in Jenkins, follow steps mentioned in **Installing Plugins in Jenkins***

Steps:

1. Navigate to Jenkins home page;
2. Click your any job created in the Labs 1-3
3. Click **Configure** from Left Panel
4. Scroll down, Under **Post-build Actions** section
5. Click Add post-build action
6. Select **Editable Email Notification**

It should add **Editable Email Notification** section

7. In **Project Recipients List** field, add comma and <email id where you want to send notification>
8. Click **Save**
9. Build the job again;
10. Click on the new triggered build number under Build History in left panel
11. Click **Console Output**
12. View the log

It should display the information about the email id where the notification is sent *and Finished: SUCCESS*

13. Open your email account ;

An email should be present in your inbox.

Lab 5: Backup Plugin

Pre-requisites:

Jenkins already have following plugins installed:-

Backup plugin: Backup or restore your Hudson configuration files

Setting at Jenkins master server end. (Already done for this server)

1. Ssh to Jenkins Master machine
2. Type command: sudo su
3. Type command: sudo chmod 777 /var/lib

Steps:

1. Click on **Manage Jenkins**
2. Scroll down, Click on Backup manager
3. Click on Setup
4. Enter **Backup** directory as "/tmp"
5. Select **Format** as zip
6. Select **Configuration files (.xml) only** checkbox
7. Click Save
8. Click on **Backup Hudson configuration**

It should start taking backup and show that a backup file is created in zip format in /tmp folder.
Note: You can confirm if the zip file is created or not by doing ssh to the Jenkins master server and navigating to /tmp using commands sudo su, cd /tmp, ls. It should show a zip file.

9. Click on back button of browser and come to Backup manager default page
10. Click on **Restore Hudson configuration**
11. Select the zip file radio button;
12. Click **Launch Restore**
13. It should start restoring the Jenkins
14. You can navigate back to any page while is processing.

It should create a Jenkins_restore folder with all the jenkins configurations file at location /var/lib
Note: You can confirm if the jenkins_restore folder is created or not by doing ssh to the Jenkins master server and navigating to /var/lib using commands sudo su, cd /var/lib, ls. It should display the all jenkins configuration files under it.

Lab 6: Integration with GitHub using Git Plugin, pull in maven based repository & configure it to run various Maven Targets

Pre-requisites:

1. Jenkins should already be configured to support following features and tools: -
   b.) GitHub (Steps used mentioned in **Configuring Jenkins for GitHub)**
   c.) Maven (Steps used mentioned in **Configuring Jenkins for building Maven Projects)**
2. Jenkins already have following plugins installed: -

   Github Integration: This provides configurations fields required for Jenkins and GitHub Integration.

   Maven Integration Plugin: This plugin provides integration between Jenkins and Maven and help in automatic building projects.

3. Jenkins already configured to run maven project.(Steps used to **Configure Jenkins for building Maven projects)**

Steps:

15. Click on **New Item**
16. Enter **Name** such as <yourname>_gitmavenjob
17. Select **Maven project**
18. Click **OK**
19. Under Source Code Management section, Select **Git** radio button
    a. Enter Repository URL - <Git repo>
    (Note: Git Repository URL for Training is https://github.com/rajnikhattarrsinha/java-tomcat-maven-example)
20. Under Build section
    a. Enter **Root POM**-pom.xml
    b. Enter **Goals and options**-clean package
21. Under Post Steps,Run only if build succeeds
22. Under Post-build Actions, Select **Add post-build action**- Editable Email Notification;
    a. In **Project Recipients List** field, add comma and <email id where you want to send notification>
23. Click **Save**
24. Click **Build Now**
25. Follow the same steps to see the log as we did in previous labs.

The complete log of the steps involved in successful building of job are displayed.

Lab 7: Continuous Integration Demo – GitHub web-hook Integration for triggering automated Builds

Pre-requisites:

1. **Lab 6: Integration with GitHub using Git Plugin, pull in maven based repository & configure it to run various Maven Targets** should be completed
2. GitHub webhook should be configured.Steps used for configuring are mentioned in **Configuring GitHub Webhook for Jenkins**
3. Jenkins should be configured for GitHub webhook integration.Steps used for configuring are mentioned in **Configuring Jenkins for GitHub**

Steps:

1. Open job created in Lab 6: Integration with GitHub using Git Plugin, pull in maven based repository & configure it to run various Maven Targets
2. Under Build Triggers section, Select **GitHub hook trigger for GITScm polling** checkbox

3. Click **Save**
4. On github end, If you are the owner of the git repo, edit a file and commit the changes.
5. Open the jenkins and observe that the build is triggered automatically as soon as you committed the file in github in your repo.
6. Click on the latest triggered build number
7. View the Console Output

The complete log of the steps involved in successful building of job are displayed.

Lab 8: Continuous Deployment: Web-App deployment to Tomcat server using Deployment to Container plugin

Pre-requisites:

1. Jenkins should already be configured to support following features and tools:-
    a. GitHub (Steps used mentioned in **Configuring Jenkins for GitHub)**
    b. Maven (Steps used mentioned in **Configuring Jenkins for building Maven Projects)**
2. Jenkins already have following plugins installed:-
   Github Integration: This provides configurations fields required for Jenkins and GitHub Integration.
   Maven Integration Plugin: This plugin provides integration between Jenkins and Maven and help in automatic building projects.
3. Jenkins already configured to run maven project.(Steps used to **Configure Jenkins for building Maven projects)**

Step A: *Create a job for building the project*

1. Click on **New Item**
2. Enter **Name** such as <yourname>_buildjob
3. Select **Maven project**
4. Click **OK**
5. Under Source Code Management section, Select **Git** radio button
    a. Enter Repository URL - <Git repo>
    (Note: Git Repository URL for Training is https://github.com/rajnikhattarrsinha/java-tomcat-maven-example)
6. Under Build Triggers section, Select **GitHub hook trigger for GITScm polling** checkbox
7. Under Build section
    a. Enter **Root POM**-pom.xml
    b. Enter **Goals and options**-clean package
8. Under Post Steps,Run only if build succeeds
9. Under Post-build Actions, Select **Add post-build action**- Editable Email Notification;
    a. In **Project Recipients List** field, add comma and <email id where you want to send notification>

10. Click **Save**

Step B: *Create a job for deploying the application on Tomcat Server*

1. In Jenkins, Click on **New Item**
2. Enter **Name** such as <yourname_deployjob>
3. Select **Freestyle project**
4. Click **OK**
5. Under General, Select **This project is parameterized** checkbox
    a. Select **Add Parameter**-String Parameter
    b. Enter **Name-**DEPLOY_VERSION
    c. Enter **Default value**-0
    d. Enter **Description**-To deploy latest war file built in build job
    e. Select **Trim the string** checkbox
6. Under Build section, Select **Add Build Step**- Execute Shell and paste the following script:
   cp
   /var/lib/jenkins/jobs/<yourname_buildjob>/builds/$DEPLOY_VERSION/com.example\$java-tomcat-maven-example/archive/com.example/java-tomcat-maven-example/1.0-SNAPSHOT/java-tomcat-maven-example-1.0-SNAPSHOT.war
   /var/lib/jenkins/workspace/<yourname>_deployjob/

Note: In the above script, you have to replace job names with your build and deploy job names respectively.

7. Under Post-build Actions, Select **Add post-build action**- Deploy war/ear to a container.
    a. Enter WAR/EAR files: **/*.war
    b. Context path: java-tomcat-maven-example_<yourname>
    c. **Containers** field:
        i. Click **Add Containers** dropdown : Select Tomcat 8.x
        ii. **Credentials**: select tomcat user;
        Note: For Lab, this is already configured in jenkins and can be selected for use. New user can be added using Add button. New user if created in jenkins, has to be added in tomcat server also.
        iii. Enter **Tomcat URL**: http://<Deployment Server IP>:9090
8. Under Post-build Actions, Select **Add post-build action**- Editable Email Notification;
    a. In **Project Recipients List** field, add comma and <email id where you want to send notification>
9. Click **Save**

Step C: *Modify the build job*

1. Open the build job created in Step A.
2. Under Post-build Actions, Select **Add post-build action**- Trigger parameterized build on other projects

a. Enter **Build Triggers->Projects to build**-<Name of the Deploy job>
b. Select **Trigger when build is-**Stable
c. Select **Add Parameters**-Predefined parameters
    i. Enter **Parameters-**DEPLOY_VERSION=${BUILD_NUMBER}

3. Click **Save**

Step D: *Running the build job*

1. Open the Job created in Step A
2. Click **Build Now**
3. Follow the same steps to see the log as we did in previous labs.
4. Click on the job name(triggered after the successful completion of build job) present at the end of the page;
5. View the Console Output of the latest build executed in this deploy job

It should display the Finished status as Success

Step E: *Verifying the deployed application*

1. Open any browser
2. http://<Deployment Server IP>:9090/<Context path>

Note: the <Context path> was set in the deploy job, please take from there.

3. Hit Enter;

Lab 9: Demo: Jenkins Master-Slave Configuration

It is already configured using steps mentioned under Setup done for Jenkins Master-Slave Configuration

Steps to be performed if Slave appears Offline

1.) Ssh to Slave machine with IP 18.233.14.3
2.) Type command: sudo su
3.) Type command: cd /opt/jenkins
4.) Type command: sudo nohup < copy the command from node details> &

(Command in my case was **java -jar agent.jar -jnlpUrl http://107.22.131.176:8080/computer/slave1/slave-agent.jnlp -secret 2afdad9010f84576b62b8e6bdfcf657efc141d99b74dbf7681e0188cc2d0e40a -workDir "/opt/jenkins"**)

5.) At Jenkins end in browser, Open the home page of Jenkins
6.) Click **New item**
7.) Select Free style project
8.) Click OK

9.) On Job details page, In General tab, Select **Restrict where this project can be run** checkbox
10.) Enter the **Label Expression** as entered while configuring the Slave ie. slave1 in our case.
11.) Under Build section, Click **Add build step**
12.) Select **Execute Shell;**
13.) Enter **Command** as echo "Hello , I am running on Slave"
14.) Click Save
15.) Build the job manually
16.) View the Console output of the latest build

It shows that the job has run on Slave machine.

Lab 10: Concepts of Pipeline, exercise to set-up sample pipelines using UpStream – Downstream Job configuration

Pre-requisites:

1. Jenkins already have following plugins installed:-
    a. **Build Pipeline:** This plugin renders upstream and downstream connected jobs that typically form a build pipeline.

Note: To know how to install plugin in Jenkins, follow steps mentioned in ***Installing Plugins in Jenkins***

Steps: Create a Build Pipeline using the following steps:-

1. Navigate to Jenkins DashBoard
2. Click on the **+** Tab next to ALL
3. Enter **Name** such as Build Pipeline
4. Select **Build Pipeline View**
5. Click **OK**
6. Scroll down to **Pipeline Flow section** , Select **Upstream/ Downstream config > Select Initial Job** > <Name of Build Job>
7. Click **OK**
8. Click Run-It will trigger the build job and deploy job once it is completed.

Configuring Tomcat to support deployment through Jenkins

1. Ssh the tomcat server;
2. Type command: sudo su
3. Type command: cd /opt/tomcat/conf
4. Type command: vim tomcat-users.xml
5. Press i
6. Use down arrow till second last line in the file
7. Add the code

<role rolename="manager-script"/>
<user username="tomcat" password="tomcat" roles="manager-script"/>

8. Press escape
9. Press :wq!
10. Type command: cd /opt/tomcat/webapps/manager/META-INF
11. Type command: vim context.xml
12. Press i
13. Use down arrow and take the cursor to the Value section;
14. comment Valve section by prefixing the code with “<!--” and suffixing the code with “-->”

For Eg:-

```
<Context antiResourceLocking="false" privileged="true" >
<!--
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
-->
  <Manager sessionAttributeValueClassNameFilter
="java\.lang\.(?:Boolean|Integer|Long|Number|String)|org\.apache\.catalina\.filters\.CsrfPreventi
onFilter\$LruCache(?:\$1)?|java\.util\.(?:Linked)?HashMap"/>
</Context>
```

15. Press escape
16. Press :wq!
17. To change the port of Tomcat Server.
18. Type command: cd /opt/tomcat/conf
19. Type command: vim server.xml
20. Press i
21. Replace the port 8080 with 9090
22. Press escape
23. Press :wq!

Configuring GitHub Webhook for Jenkins

1. Open GitHub
2. Navigate to Git Repo of Frontend code;
3. Navigate To Settings of repository
4. Click Webhook
5. Click Add Webhook
6. Enter Payload URL-http://<Public IP of Jenkins Server>:8080/github-webhook/
7. Click Save Webhook

Configuring Jenkins for GitHub

1. Navigate to Manage Jenkins ->Open Configure System
2. In GitHub section, Select Add GitHub Server->GitHub Server

a. Enter **Name**-Git Server
b. Click on the Advanced… button present below the Add GitHub Server drop down
c. Select Additional actions->Manage additional github actions->Select Convert login and password to token
d. Select **from login and password** radio button
e. Enter your GitHub login credentials
f. Click **Create token credentials** button
g. Scroll up and select the generated token in the **Credentials** field

3. Click **Save** on Configure System

Configuring Jenkins for SMTP Notifications

1. Navigate to Manage Jenkins ->Open Configure System
2. In section Extended E-mail Notification, Configure the following:-
    a. Enter **SMTP Server**-smtp.gmail.com
    b. Click **Advance** Button
    c. Select **Use SMTP Authentication** checkbox
    d. Enter User Name: <Email ID from where notifications to be sent>
    e. Enter Password:<Enter its password>
    f. Select **Use SSL** checkbox
    g. Enter **SMTP Port**- 465 *(Note: enter port corresponding to SMTP server configured above, 465 is the port for gmail)*
    h. Enter Default Recipients-<comma separated email IDs where build status email is to be sent>
    i. Click **Default Triggers** button
    j. Select **Always**
3. Click **Save** on Configure System.

Installing Plugins in Jenkins

1. Login Jenkins
2. Click **Manage Jenkins** from left panel
3. Click **Manage Plugins**
4. Navigate to **Available** Tab
5. Search the required plugin using search text box provided above;
6. Select the checkbox;
7. Repeat the steps 6 and 7 for more plugins;
8. Click **Install without restart** button
9. After seeing the blue colored Success icon against the installed plugin name.
10. Click **Restart Jenkins when Installation is complete and no jobs are running** checkbox.

(Note:-In case you do not find any of the plugin in Available Tab, please check in Installed Tab.It may got installed as part of default plugins while installing Suggested plugins)

Configure Jenkins for building Maven projects

1. Navigate to Manage Jenkins
2. Click Global Tool Configuration
3. In Section Maven,
    a. Click **Add Maven** button
    b. Enter **Name** - Maven 3.5
    c. Install From Apache, Select **Version** 3.5.4
4. Click **Save**

Setting Up Email Account to be used in Email Notification settings in Jenkins

Turn the email account setting Allow less secure apps to ON from the following URL for your sender email account to be configured for Jenkins:-

https://myaccount.google.com/u/3/lesssecureapps?pli=1&pageId=none

Setup done for Jenkins Master-Slave Configuration

Pre-requisites:

1.) Ubuntu 16 Server with Java and Maven installed in it.
2.) Security Group Inbound Rules should be opened for the required ports and IPs.

Steps at Jenkins end:

1.) Login to Jenkins;
2.) Navigate to **Manage Jenkins**
3.) Click **Configure Global Security**
4.) Under Agents section, Select **TCP port for JNLP agents** radio button as Random
5.) Click **Save**
6.) Navigate to **Manage Jenkins**
7.) Click Manage Nodes
8.) Click **New Node**
9.) Enter Node Name such as testslave1
10.) Select **Permanent Agent** radio button
11.) Enter **Remote root directory** as “/opt/jenkins” (It can be of your choice)
12.) Enter **Labels** as <same as Node Name>
13.) Select **Launch method** as Launch via Java web start
14.) Click **Save**

15.) On the Nodes details page , you will see agent.jar link , R-Click and Copy its link location

Note: This link location is to be used in steps done at Slave server end mentioned below

16.) Copy the command given under section **Run from agent command line:**

Note: This command would need to be run in one of the steps done at slave server end mentioned below.

Steps at Slave Server end:

4. Ssh to Slave machine
5. Type command: sudo -i
6. Type command: useradd -s /bin/bash jenkins
7. Type command: passwd jenkins
8. Type any password
9. Retype password

Note: This sets password for user jenkins

10. Type command visudo
11. Add **jenkins ALL=(ALL) NOPASSWD:ALL** under User privilege specification
12. Press control x
13. Press Y to save to changes
14. Type command: su jenkins
15. Type command: su ls
16. Type command: sudo mkdir /opt/jenkins
17. Type command: cd /opt/jenkins
18. sudo wget <copy link location of agent or slave.jar from node details>

eg. http://34.205.140.144:8080/jnlpJars/agent.jar

19. Type command: sudo nohup < copy the command from node details> &

(Command in my case was **java -jar agent.jar -jnlpUrl http://107.22.131.176:8080/computer/slave1/slave-agent.jnlp -secret 2afdad9010f84576b62b8e6bdfcf657efc141d99b74dbf7681e0188cc2d0e40a -workDir "/opt/jenkins"**)

Note: you may get error about the port of series 40000 not opened. You would need to add rule to allow it in the inbound rule of security group of jenkins master.