* **INTELLIQIT:**

Developers will upload their code in Git, Jenkins will download it [CONTINUIOUS DOWNLOAD]and triggered build process[CONTINUIOUS BUILD 🡪 Ant/Maven], after building we will get a setup file i.e an artifact, Jenkins will take these artifact and deploy[CONTINUIOUS DEPLOYMENT] it in Testing environment(QA server), Testers will create test scripts using selenium tool, Jenkins will pickup those test scripts and checks whether application working correctly or not[CONTINUIOUS TESTING], if testing passes then Jenkins will deploy the artifact into ProdServer[CONTINUIOUS DELIVERY] or incase if it fails then sends notification to developers to recheck the code

Day 1

====================================================

Jenkins

===============

This is a tool for implementing CI-CD

* **Stages in CI-CD**

===========================

Stage 1 (Continuous Download)

-------------------------------------

Developers create some code and upload that code into the version controlling system. The moment code is uploaded Jenkins gets a notification and it will download that code, this is called Continuous Download.

Stage 2 (Continuous Build)

--------------------------------------

The code downloaded in the previous stage has to converted into an artifact. This artifact can come in the format of war, jar, ear, exe file etc. To convert the code into an artifact jenkins takes the help of certain build tools like ANT, Maven, MsBuild etc.

Stage 3 (Continuous Deployment)

------------------------------------------

The artifact created in the previous stage is now deployed into the testing environment. This Testing environment can be running on some application servers like tomcat, Jboss, Weblogic etc. Jenkins should deploy the artifact into these application servers from where testers can start accessing it.

Stage 4 (Continuous Testing)

-------------------------------------

Jenkins now executes the automation test scripts created by the testers and checks if the application is working according to the clients requirement. If it does not Jenkins will send email notifications to the concerned team members and developers will fix the defects and upload the modified code into the version controlling system. Jenkins will now start from stage 1

Stage 5 (Continuous Delivery)

---------------------------------------

If testing passes Jenkins will now take approvals from the delivery team and deploy the artifact into the prod environment where the end user can start accessing it.

Note: The first four stages are called CI (Continuous Integration)

The last stage is called as CD (Continuous Delivery)

* **Setup of 3 Ubuntu servers using vagrant**

===============================================

Vagrant

===================

1 Download and install Oracle virtual Box

https://www.virtualbox.org/wiki/Downloads

2 Download and install vagrant

https://www.vagrantup.com/downloads.html

3 To check if vagrant is installed

vagrant --version

4 Copy the vagrantfile into an empty folder

5 Open cmd prompt

cd path\_of\_folder\_where\_vagrantfile\_is\_copied

vagrant up

6 Open Oracle virtual box to access these VM's

Username and Password: vagrant

6 To destroy the VM's

vagrant destroy

========================================

Day 2

========================================

* **Setup of Jenkins environment for CI-CD**

========================================

1 Create 3 AWS ubuntu instances and name them

JenkinsServer, QAServer, ProdServer

2 Connect to JenkinsServer AWS instance using git bash

3 Update the apt repository

sudo apt-get update

4 Install jdk

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

5 Install git and maven

sudo apt-get install -y git maven

6 Download jenkins.war

wget https://get.jenkins.io/war-stable/2.263.1/jenkins.war

7 To start Jenkins ----------we shall run this command every time when we want to start Jenkins

java -jar jenkins.war

java -jar jenkins.war --httpPort=9090 -------------------------------------run this command if you want to change port number

8 To access jenkins from browser

public\_ip\_of\_jenkinsserver:8080

9 Unlock jenkins by entering the password

10 Click on "Install suggested plugins"

11 Create First admin user--->Click on Continue--->Finish

-------------------------------------------------------------------------------

* **Setup of tomcat on QA and ProdServers**

-----------------------------------------

1 Connect to QAServer Aws instance using git bash

2 Update the apt repository

sudo apt-get update

3 Install tomcat9

sudo apt-get install -y tomcat9

4 Install tomcat9-admin -------- it contains all the dependencies that require for tomcat

sudo apt-get install -y tomcat9-admin

5 Edit the tomcat-users.xml file

sudo vim /etc/tomcat9/tomcat-users.xml

Delete the entire content of the file and add the below data

<tomcat-users>

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

</tomcat-users>

These credentials we need to pass in Jenkins so that it can deploy artifact in it.

* Here there is a file servers.xml in which you can change your port number.

6 Restart tomcat9

sudo service tomcat9 restart

7 To access tomcat from the level of browser

public\_ip\_qaserver:8080

Repeat the above 7 steps on ProdServer AWS instance

=====================================

Day 3

=============================================================================

**Stage 1 (Continuous Download)**

=======================================

1 Open the dashboard of jenkins

2 Click on New item---->Enter some item name "Development"

3 Select Free Style Project---->OK

4 Go to Source code Management

5 Select Git

6 Enter the github url where developers have uploaded the code

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

7 Click on Apply--->Save

8 Go to the dashboard of Jenkins

9 Go to the Development job and click on Build icon

This job will download all the code uploaded by the developers into the remote git server

**Stage 2 (Continuous Build)**

========================================

1 Open the dashboard of Jenkins

2 Go to the Development job---->Click on configure

3 Go to Build section

4 Click on Add Build step

5 Click on Invoke top level maven targets

6 Enter maven goal: package--------------------its 5th stage of maven life-cycle

7 Click on Apply--->Save

8 Go to the dashboard of Jenkins

9 Go to Development job---->Click on Build icon

This job will convert the java code into an artifact.

This artifact comes in the format of a war file.

**Stage 3 (Continuous Deployment)**

========================================

1 Open the dashboard of Jenkins

2 Click on Manage Jenkins

3 Click on Manage Plugins

4 Go to Available section

5 Search for "Deploy to Container" plugin

6 Click on Install without restart

7 Go to the dashboard of Jenkins

8 Go to the Development job---->Click on Configure

9 Go to Post Build actions

10 Click on Add Post Build actions

11 Click on Deploy war/ear to container.

war/ear files: \*\*/\*.war

Context path: testapp

Click on Add container

Select tomcat9

Enter username and password of tomcat

tomcat url: private\_ip\_of\_qaserver:8080

here what we have done is asking Jenkins to check all subfolders sub-sub folders in current directory and find war files and deploy them to a Tomcat server. Tester can access it by name ‘testapp’ in browser e.g public\_ip\_of\_testing\_server:8080/testapp

12 Apply---->save

13 Go to the dashboard of Jenkins

14 Go to the Development job---->Click on Build icon

This job will deploy the artifact into the qaserver

15 To access the application from the browser

public\_ip\_of\_qaserver:8080/testapp

========================================

Day 4

========================================

**Stage 4 (Continuous Testing)**

======================================

1 Open the dashboard of Jenkins

2 Create a new job and name it "Testing"

3 Select Free Style Project--->OK

4 Select Git--->Enter the github url where testers have uploaded the code.

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

5 Go to Build section

6 Click on Add Build step

7 Click on Execute shell

java -jar path/testing.jar

8 Apply--->Save

9 Go to the dashboard of jenkins

10 Go to the testing job--->Click on Build icon

This job will download the selenium test scripts and execute them.

Linking the Development job with the Testing job

============================================

The Development job has to be linked with the testing job so that all the stages of CI-CD can happen in a flow. This is called as upstream/downstream config in jenkins

1 Go to the dashboard of jenkins

2 Go to the development job

3 Click on Configure

4 Go to Post build actions

5 Click on Add post build action

6 Click on Build other projects

7 Enter project name as "Testing"

8 Apply--->Save

============================================

Copying artifacts from Development job and Testing job

============================================

1 Open the dashboard of Jenkins

2 Click on Manage Jenkins--->Manage Plugins

3 Go to Available section

4 Search for "Copy Artifact" plugin

5 Click on Install without restart

6 Go to the dashboard of Jenkins

7 Go to the Development job--->Click on configure

8 Go to Post Build actions

9 Click on Add Post Build actions

10 Click on Archive the artifacts

File to archive: \*\*\\*.war

11 Apply--->Save

12 Go to the dashboard of Jenkins

13 Go to the Testing job---->Click on Configure

14 Go to Build section

15 Click on Add Build step

16 Click on Copy artifacts from other projects

17 enter project name as Development

18 Save

**Stage 5 (Continuous Delivery)**

===============================

1 Open the dashboard of Jenkins

2 Go to the Testing job

3 Click on Configure

4 Go to Post Build actions

5 Click on Add post build action

6 Click on Deploy war/ear to container

7 war/ear files: \*\*/\*.war

Context path: prodapp

Click on Add container

Select tomcat9

Enter username and password of tomcat9

Tomcat url: private\_ip\_of\_prodserver:8080

8 Apply--->Save

============================================

Day 5

============================================ **User Administration in Jenkins**

===========================================

**Creating users in Jenkins**

===========================

1 Open the dashboard of jenkins

2 click on manage jenkins

3 click on manage users

4 click on create users

5 enter user credentials

**Creating roles and assigning**

==============================

1 Open the dashboard of jenkins

2 click on manage jenkins

3 click on manage plugins

4 click on role based authorization strategy plugin

5 install it

6 go to dashboard-->manage jenkins

7 click on configure global security

8 check enable security checkbox

9 go to authorization section-->click on role based strategy radio button

10 apply-->save

11 go to dashboard of jenkins

12 click on manage jenkins

13 click on manage and assign roles

14 click on manage roles

15 go to global roles and create a role "employee"

16 for this employee in overall give read access

and in view section give all access

17 go to item roles-->Give the role as developer

and pattern as Dev.\* (i.e developer role can access

only those jobs whose name start with Dev)

18 similarly create another role as tester and assign the pattern as "Test.\*"

19 give all permissions to developers and tester

20 apply--save

21 click on assign roles

22 go to global roles and add user1 and user2

23 check user1 and user2 as employees

24 go to item roles

25 add user1 and user2

26 check user1 as developer and user2 as tester

27 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

========================================================================

* **Alternate ways of Setup of Jenkins**

=========================================

**Setups of jenkins as a direct installation**

-------------------------------------------------

1 Update the apt repository

sudo apt-get update

2 Install java

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

3 Add the jenkins repository keys to the apt repository

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

4 Add the debain package address to jenkins.list file

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

5 Update the apt repository

sudo apt-get update

6 Install jenkins

sudo apt-get install -y jenkins

Setup of Jenkins using tomcat

---------------------------------

1 Update apt repository

sudo apt-get update

2 Install tomcat9

sudo apt-get install -y tomcat9

3 Install tomcat9-admin

sudo apt-get install -y tomcat9-admin

4 Download the jenkins.war file

wget https://get.jenkins.io/war-stable/2.249.2/jenkins.war

5 Give write permission on tomcat9 folder to others

sudo chmod o+w -R /var/lib/tomcat9

6 Copy the jenkins.war into the tomcat9 webapps folder

cp jenkins.war /var/lib/tomcat9/webapps

7 To access tomcat from browser

public\_ip:8080

To access jenkins running on tomcat

public\_ip:8080/jenkins

Day 6

==========================================================================

* **Master Slave Architecture of Jenkins**

============================================

This is used distribute the workload to additional linux servers called as slaves. This is used when we want to run multiple jobs on jenkins parallelly.

Setup

============

1 Create a new AWS ubuntu20 instance

2 Install the same version of java as present in the master

sudo apt-get update

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

3 Setup passwordless SSH between Master and slave

a) Connect to slave and set password to default user i.e ubuntu

sudo passwd ubuntu

b) Edit the ssh config file

sudo vim /etc/ssh/sshd\_config

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

sudo nano /etc/ssh/sshd\_config

ctrl w --------------to search

search for PasswordAuthentication

make changes

ctrl x ----------------to save file

y -----------------to yes

c) Restart ssh

sudo service sshd restart

d) Connect to Master using git bash

e) Generate the ssh keys

sudo su jenkins

ssh-keygen

f) Copy the ssh keys

ssh-copy-id ubuntu@private\_ip\_of\_slave

g) make sure if pw less connection is established or not

ssh ubuntu@private\_ip\_qa\_server (from Jenkins server)

This will copy the content of the public keys to a file called

"authorised\_keys" on the slave machine

===========================================================================

Connect to slave using git bash

4 Download the slave.jar file

# cd /home/ubuntu

wget <http://private_ip_of_jenkinsserver:8080/jnlpJars/slave.jar> 🡨 can we use same command to copy files from other server

wget <http://172.31.33.159:8080/jnlpJars/slave.jar>

5 Give execute permissions to the slave.jar

chmod u+x slave.jar

6 Create an empty folder that will be the workspace of jenkins

mkdir workspace

7 Open the dashboard of Jenkins

8 Click on Manage Jenkins--->Click on Manage Nodes and Clouds

9 Click on New node---->Enter some node name as Slave1

10 Select Permanent Agent--->OK

12 Enter remote root directory as /home/ubuntu/workspace

12.1 usage : choose 2nd option

13 Labels: myslave (This label is associated with a job in jenkins

and then that job will run on that slave)

14 Go to Launch Method and select "Launch agent via execution of command on master"

Launch command 🡪 ssh ubuntu@private\_ip\_of\_slave java -jar slave.jar

with this command as we run it from master machine 🡪 we execute ‘java-jar slave.jar’ command on slave machine from master machine without actually connecting to slave machine

15 Click on Save

16 Go to the dashboard of Jenkins

17 Go to the job that we want to run on slave---->Click on Configure

18 Go to General section

19 Check restrict where this project can be run

20 Enter slave label as myslave

* Build pipeline plugin:
* We can download ‘Build pipeline plugin’ which shows all CI-CD steps graphically that we performed above.
* Here we can run/configure/delete the job and also can see logs
* Its not recommended to use
* Why do we need to use scripted pipeline:
* Incase Jenkins server get crashed, you will have to re-configure it along with jobs that you created. It will consume time.
* Other reason is in case Jenkins get restarted in between execution of any job then if we are following our regular process then that job execution will stop over there itself and won’t start automatically once it get restarted. But if we are doing job execution with scripted pipeline then it will get started automatically though Jenkins get restarted.
* While doing CI-CD with regular process we installed so many plugins (piece of software that enhance Jenkins capabilities) which consumes space and use resources of Jenkins server. It may slow down the Jenkins and its also not always secure to use them.
* By use of scripted pipeline we will have no need to install plugins always.
* There must be action if ci-cd fails at any stage which is very easy by using scripted pipeline
* It will be easy to find errors in scripted pipeline. If someone did any mistake in freestyle project then it gets difficult to find it.
* It becomes easy to keep versions on Jenkinsfile when we push it into Github.

Day 7

============================================================================

=========================================================================

Pipeline as Code

======================

Implementing all the stages of CI-CD from the level of a Groovy script

file is called as Pipeline as Code. This groovy script files is called

as Jenkinsfile and generally it is uploaded into the remote git server

along with the application code. From the remote git server this Jenkinsfile

will trigger all the stages of CI-CD

Advantages

====================

1 Since the stages of CI-CD are implemented from the level of code

it can perform version controlling on it i.e it gives the team members

ability to edit and review the code and yet maintain multiple versions

of jenkinsfile

2 Jenkinsfile can withstand planned and unplanned restarts of the Jenkins

master

3 It can perform all stages of CI-CD with minimum number of plugins as

a result of which it give better performance than the free style projects

4 It can handle all the real time challenges like if conditions,

error handling etc

Pipeline can be implemented in 2 ways

1 Scripted Pipeline

2 Declarative Pipeline

Syntax of Scripted Pipeline

=================================

node('master')

{

stage('Stage name in CI-CD')

{

Groovy script code for implementing this stage

}

}

Syntax of Declarative Pipeline

====================================

pipeline

{

agent any

stages

{

stage('Stage name in CI-CD')

{

steps

{

Groovy script for implementing this stage

}

}

}

}

=========================================================================

Day 8

=========================================================================

Scripted piepline

======================================

node('master')

{

stage('ContinuousDownload')

{

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

}

stage('ContinuousBuild')

{

sh 'mvn package'

}

stage('ContinuousDeployment')

{

deploy adapters: [tomcat9(credentialsId: '9ddc009f-64d7-48e3-afcf-ba859a4ebd07', path: '', url: 'http://172.31.82.52:8080')], contextPath: 'testapp1', war: '\*\*/\*.war'

}

stage('ContinuousTesting')

{

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

sh 'java -jar /home/ubuntu/.jenkins/workspace/ScriptedPipeline/testing.jar'

}

stage('ContinuousDelivery')

{

input message: 'Waiting for approval from the DM!', submitter: 'srinivas'

deploy adapters: [tomcat9(credentialsId: '9ddc009f-64d7-48e3-afcf-ba859a4ebd07', path: '', url: 'http://172.31.92.195:8080')], contextPath: 'prodapp', war: '\*\*/\*.war'

}

}

=======================================================================

Day 9

=======================================================================

Scripted pipeline without deploy to container plugin: in this case no need to edit tomcat-users.xml file or no need to add username , pw in that file. Also there in no need to know username and pw of Tomcat to deploy code into it.

node('master')

{

stage('ContinuousDownload')

{

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

}

stage('ContinuousBuild')

{

sh 'mvn package'

}

stage('ContinuousDeployment')

{

sh 'scp /home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp/target/webapp.war ubuntu@172.31.82.52:/var/lib/tomcat9/webapps/qaapp.war'

}

stage('ContinuousTesting')

{

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

sh 'java -jar /home/ubuntu/.jenkins/workspace/ScriptedPipeline/testing.jar'

}

stage('ContinuousDelivery')

{

input message: 'Waiting for approval from the DM!', submitter: 'srinivas'

sh 'scp /home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp/target/webapp.war ubuntu@172.31.92.195:/var/lib/tomcat9/webapps/prodapp.war'

}

}

* Whenever we are unable to find ‘deploy war/ear to container’ step in pipeline we have to go with other method i.e scp

1. First we need to establish ssh connectivity between Jenkins server and qa-server, Jenkins server and prodserver to use scp command
2. Go to qaserver
3. $ whoami (ubuntu)
4. $ sudo passwd ubuntu (set some pw)
5. $ sudo vim /etc/ssh/sshd\_config
6. Passwordauthentication yes
7. Restart ssh service🡪 $ sudo service tomcat9 restart
8. Connect to Jenkins server
9. $ sudo su jenkins
10. $ ssh-keygen
11. $ ssh-copy-id ubuntu@private\_ip\_of\_qaserver
12. $ ssh-copy-id ubuntu@private\_ip\_of\_prodserver
13. To check pw less connection established or not

ssh ubuntu@private\_ip\_of\_qa-server

1. Connect to qa-server 🡪 $ cd /var/lib/tomcat9/conf
2. Vi tomcat-users.xml 🡨 do not delete anything as we did in usual process

Format of scp command:

$ scp srcfile username@ipaddress: destination

e.g suppose we have a file f1 in Jenkins server and we need to copy it in qaserver with other name say file1

$ scp f1 [ubuntu@172.42.51.7:/temp/file1](mailto:ubuntu@172.42.51.7:/temp/file1)

1. Go to logs and find out in which path Jenkins is creating artifacts war file. It is🡪

/home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp/target/webapp.war

Copy that path

1. Hence command becomes🡪

$ scp /home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp/target/webapp.war ubuntu@private\_ip\_of\_qaserver : /var/lib/tomcat9/webapps/qaapp.war

qaapp.war is nothing but a context path

1. Copy this command and go pipeline syntax generator

Steps: sh: shell script

Shell-script: paste copied command

Click on ‘Generate Pipeline script’

1. Copy generated groovy script and paste it in 3rd stage i.e ‘ContinuousDeployment’
2. Similarly you can also do for last stage i.e ‘ContinuousDelivery’ by making necessary changes i.e private ip and context path
3. With this build will fail
4. Connect to qaserver
5. $ cd var/lib
6. $ ls -ld tomcat9/
7. We need to add write permission to other users
8. $ sudo chmod o+w -R tomcat9/
9. Repeat 12
10. Now build
11. Repeat steps 5 to 16 for prodserver with context path 🡪 prodapp.war

* Normally in organization this groovy script will be uploaded in github and settings will be made in Jenkins such that it will check if new code is been uploaded in gihub or not OR settings will be made in Jenkins such that it will get a notification to build if new code is been uploaded in gihub

Lets see first case in which settings will be made in Jenkins such that it will check if new code is been uploaded in github. If new code is there then build will start automatically

1. scriptedPipeline🡪 pipeline 🡪 Definition 🡪 Pipeline script from SCM
2. SCM 🡪 git
3. Paste github-url in which Jenkins-file is uploaded
4. Scriptpath 🡪 name of that file (Jenkinsfile)
5. Apply 🡪 save
6. Build-triggers 🡪 check ‘Poll scm’
7. Schedule 🡪 \* \* \* \* \*
8. Save
9. Make some changes in code and push in github
10. Job would start running automatically

Day 10

===========================================================================

Declarative Pipeline

==================================

pipeline

{

agent any

stages

{

stage('ContinuousDownload')

{

steps

{

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

}

}

stage('ContinuousBuild')

{

steps

{

sh 'mvn package'

}

}

stage('ContinuousDeployment')

{

steps

{

sh 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.82.52:/var/lib/tomcat9/webapps/myapp.war'

}

}

stage('ContinuousTesting')

{

steps

{

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

sh 'java -jar /home/ubuntu/.jenkins/workspace/DeclarativePipeline/testing.jar'

}

}

stage('ContinuousDelivery')

{

steps

{

input message: 'Required approval from the DM!', submitter: 'srinivas'

sh 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.92.195:/var/lib/tomcat9/webapps/myprodapp.war'

}

}

}

}

=======================================================================

Declarative Pipeline with post conditions

==============================================

Post conditions are execute after all the stages are done

pipeline

{

agent any

stages

{

stage('Stage name in CI-CD')

{

steps

{

Groovy code

}

}

}

post

{

success

{

}

failure

{

}

always

{

}

abort

{

}

changed

{

}

}

}

===============================================================

pipeline

{

agent any

stages

{

stage('ContinuousDownload')

{

steps

{

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

}

}

stage('ContinuousBuild')

{

steps

{

sh label: '', script: 'mvn package'

}

}

stage('ContinuousDeployment')

{

steps

{

sh label: '', script: 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.31.15:/var/lib/tomcat8/webapps/testwebapp.war'

}

}

stage('ContinuousTesting')

{

steps

{

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

sh label: '', script: 'java -jar /home/ubuntu/.jenkins/workspace/DeclarativePipeline/testing.jar'

}

}

}

post

{

success

{

input message: 'Waiting for Approval!', submitter: 'naresh'

sh label: '', script: 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.26.41:/var/lib/tomcat8/webapps/prodwebapp.war'

}

failure

{

mail bcc: '', body: '', cc: '', from: '', replyTo: '', subject: 'Jenkins CI-CD Failed', to: 'gandham.saikrishna@gmail.com'

}

}

}

===========================================================================

Declarative Pipeline with exception handling

================================================

Exception handling is the process of overcoming an exception and

resuming the execution of the remaining steps in the program

Exception handing is implemented using try, catch

The section of code that might generate an error should be given

in the try part. If it fails the control comes to the catch section

Syntax

---------

try

{

}

catch(Exception e)

{

}

======================================================================

pipeline

{

agent any

stages

{

stage('ContinuousDownload')

{

steps

{

script

{

try

{

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

}

catch(Exception e1)

{

mail bcc: '', body: 'Jenkins is unable to download the development code from the github', cc: '', from: '', replyTo: '', subject: 'Download failed', to: 'git.admin@gmail.com'

exit(1)

}

}

}

}

stage('ContinuousBuild')

{

steps

{

script

{

try

{

sh 'mvn package'

}

catch(Exception e2)

{

mail bcc: '', body: 'Jenkins is unable to create an artifact with maven', cc: '', from: '', replyTo: '', subject: 'Build failed', to: 'developers@gmail.com'

exit(1)

}

}

}

}

stage('ContinuousDeployment')

{

steps

{

script

{

try

{

sh 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.82.52:/var/lib/tomcat9/webapps/myapp.war'

}

catch(Exception e3)

{

mail bcc: '', body: 'Jenkins is unable to deploy into tomcat on the QA environment', cc: '', from: '', replyTo: '', subject: 'Deployment failed', to: 'middleware.team@gmail.com'

exit(1)

}

}

}

}

stage('ContinuousTesting')

{

steps

{

script

{

try

{

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

sh 'java -jar /home/ubuntu/.jenkins/workspace/DeclarativePipeline/testing.jar'

}

catch(Exception e4)

{

mail bcc: '', body: 'Selenium automation testing has failed', cc: '', from: '', replyTo: '', subject: 'Testing failed', to: 'testing.team@gmail.com'

exit(1)

}

}

}

}

stage('ContinuousDelivery')

{

steps

{

script

{

try

{

input message: 'Required approval from the DM!', submitter: 'srinivas'

sh 'scp /home/ubuntu/.jenkins/workspace/DeclarativePipeline/webapp/target/webapp.war ubuntu@172.31.92.195:/var/lib/tomcat9/webapps/myprodapp.war'

}

catch(Exception e5)

{

mail bcc: '', body: 'Unable to deploy into tomcat on the ProdServers', cc: '', from: '', replyTo: '', subject: 'Delivery failed', to: 'delivery.team@gmail.com'

}

}

}

}

}

}

Day 11

==========================================================================

MultiBranch Pipeline

===========================

Developers upload multiple functionalities code on different branches

On each of these branches there will be a copy of the Jenkinsfile

which has CI instructions of what should be done on that branch

Developers Activity

=========================

1 Clone the maven repository

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

2 Move into this cloned repository and delete .git folder

cd maven

rm -rf .git

3 Initilise a new git repository

git init

4 Send the files into stagging area and local repository

git add .

git commit -m "a"

5 Create a jenkins file and put the stages of CI that should happen

on master branch

vim Jenkinsfile

6 Send it to stagging and local repository

git add .

git commit -m "b"

7 Create a new branch called loans and create a create a new Jenkinsfile

git checkout -b loans

vim Jenkinsfile

Use the CI instructions that should be done on Loans branch

8 Send this to stagging and local repoistory

git add .

git commit -m "c"

9 Open github.com---->Create a new repository

10 Push all the branches from local machine to remote github

git push origin --all

Jenkins Admin Activity

==============================

1 Open the dashboard of Jenkins

2 Click on New item---->Enter item name as MultBranchPipeline

3 Select MultiBranchPipeline--->OK

4 Go to Branch Sounrces---->Select Git-->enter github url where developers

uploaded the code

5 Go to Scan Multi branch pipeline triggers

6 check ‘Periodically if not otherwise run’ and select ‘1 minute’

7 Apply--->Save

==================================================================

=======================================================================

Webhooks

===========

This is used to send notifications from github to jenkins

Whenever any code changes are done and that is checkdin into

github, webhook will send an immediate notifiction ot JEnkins

and Jenkins will triggern the job

1 Open github.com---->Click on the reposiotry that we are working on

2 On the right corner clikc on Setting

3 Click on Webhooks in the left pannel

4 Click on Add Webhook

5 In Payload URL: http://public\_ip\_jenkinsserver:8080/github-webhook/

6 In Content type select :application/json

7 Click on Add Webhook

8 Open the dashboaard of Jenkins

9 Go to the job that we want to configure

10 Go to Build triggers

11 Check GitHub hook trigger for GITScm polling

12 Click on Apply--->Save

Now if we make any changes to the code in github then github

will send an notification to jenkins and jenkins will run that job

=================================================================================

Day 9

================================================================================

Master Slave if Jenkins is installed as a permanant installation

=======================================================================

Setups of jenkins as a direct installation

-------------------------------------------------

1 Update the apt repository

sudo apt-get update

2 Install java

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

3 Add the jenkins repository keys to the apt repository

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key

| sudo apt-key add -

4 Add the debain package address to jenkins.list file

sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \

/etc/apt/sources.list.d/jenkins.list'

5 Update the apt repository

sudo apt-get update

6 Install jenkins

sudo apt-get install -y jenkins

--------------------------------------------------------------------------

Setup

============

1 Create a new AWS ubuntu20 instance

2 Install the same version of java as present in the master

sudo apt-get update

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

3 Setup passwordless SSH betwen Master and slave

a) Connect to slave and set password to default user

sudo passwd ubuntu

b) Edit the ssh config file

sudo vim /etc/ssh/sshd\_config

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

c) Restart ssh

sudo service ssh restart

d) Connect to Master using git bash

f) Switch to JEnkins user

sudo su - jenkins

e) Generate the ssh keys

ssh-keygen

f) Copy the ssh keys

ssh-copy-id ubuntu@private\_ip\_of\_slave

This will copy the content of the public keys to a file called

"authorised\_keys" on the slave machine

===========================================================================

Connect to slave using git bash

4 Downlaod the slave.jar file

wget http://private\_ip\_of\_jenkinsserver:8080/jnlpJars/slave.jar

5 Give execute permissions to others on the slave.jar

chmod o+x slave.jar

6 Create an empty folder that will be the workspace of jenkins

mkdir workspace

7 Give execute permissions to others on this directory

chmod o+w workspace

7 Open the dashboard of Jenkins

8 Click on Manage Jenkins--->Click on Manage Nodes and Clouds

9 Click on New node---->Enter some node name as Slave1

10 Select Permanant Agent--->OK

12 Enter remote root directory as /home/ubuntu/workspace

13 Labels: myslave (This label is associated with a job in jenkins

and then that job will run on that slave)

14 Go to Launch Method and select "Launch agent via execution of command on master"

15 Click on Save

16 Go to the dashboard of Jenkins

17 Go to the job that we want to run on slave---->Click on Configure

18 Go to General section

19 Check restrict where this project can be run

20 Enter slave label as myslave

=================================================================================

Cat Light Notifications

---------------------------------

Catlight is a third party s/w that is used to get notification from

Jenkins in the form of pop up msgs

Catlight is a client side s/w and it has to be installed on every

team members machines

1 Download and install cat light

https://catlight.io/downloads

2 Select the continuous integration tool as Jenkins

3 Enter the public ip\_of\_jenkinserver:8080

5 Enter the username and password of Jenkins

6 Select the jobs for which we want notifications

===============================================================================