Jenkins Deployment On AWS

Project - Understanding of CI/CD tool Jenkins From Scratch.

" Successfully deployed a scalable Jenkins CI/CD pipeline on AWS, automating build, test, and deployment processes to enhance software delivery efficiency and reliability."

Developed a Jenkins CI/CD pipeline on AWS from scratch, starting with basic infrastructure setup and progressing to a fully automated build and deployment system.

Jenkins -This is a tool used for implementing CI-CD

Stage in CI-CD

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

Stage 1 (Continuous Download)

Whenever developers upload some code into the Git repository Jenkins will receive a notification and it will download all that code. This is called as Continuous Download.

Stage 2 (Continuous Build)

The code downloaded in the previous stage had to converted into a setup file commonly known as artifact. To create this artifact jenkins uses certain build tools like ANT, Maven etc
The artifact can be in the format of a .jar, .war, .ear file etc
This stage is called as Continuous Build

Stage 3 (Continuous Deployment)

The artifact created in the previous stage has to be deployed into the QA Servers where a team of testers can start accessing it. This QA environment can be running on some application servers like tomcat, Weblogic etc. Jenkins deploys the artifact into these application servers and this is called Continuous Deployment

Stage 4 (Continuous Testing)

Testers create automation test scripts using tools like selenium, UFT etc Jenkins run these automation test scripts and checks if the application is working according to clients requirement or not, If testing fails Jenkins will send automated email notifications to the corresponding team members and developers will fix the defects and upload the modified code into Git, Jenkins will again start from stage 1

Stage 5 (Continuous Delivery)

Once the application is found to be defect free Jenkins will deploy it into the Prod servers where the end user or client can start accessing it This is called continuous delivery

Here the first 4 stages represent CI (Continuous Integration) the last stage represents CD (Continuous Delivery)

===

Day 1

===

Setup

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

1 Create 3 ubuntu20 AWS instances

  and name them Jenkins Server, QA Server, Prod Server

2 Download and install git to connect to these ubuntu servers

Installing Jenkins

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

1 Connect to the Jenkins Server AWs instance from git bash

2 Update the apt repository--

```
sudo apt-get update
```

3 Install jdk--

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

4 Check if java is installed or not--

```
java --version
```

5 Install git and maven--

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

6 Check if git and maven are installed--

```
git --version
mvn --version
```

7 Download jenkins.war--

```
wget https://get.jenkins.io/war-stable/2.263.4/jenkins.war
```

8 To start Jenkins--

```
java -jar jenkins.war
```

9 To access jenkins from browser--

```
publci_ip_jenkinserver:8080
```

10 Enter the initial admin password to unlock jenkins

11 Click on Install suggested plugin

12 Create first admin user--->Continue--->Finish--->Start using jenkins

==

Setup of tomcat on QA and ProdServers

----

1 Connect to QA server AWS instance using gitbash

2 Update the apt repository--

```
  sudo apt-get update
```

3 Install tomcat9--

```
  sudo apt-get install -y tomcat9
```

4 Install tomcat9-admin--

```
  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 replace with the below code

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

6 Restart tomcat--

```
  sudo servcie tomcat9 restart
```

7 To access tomcat from browser--

```
  public_ip_ofqaserver:8080
```

Repeat the above steps on Prod server AWS instance

===

Day 2

===

Stage 1 (Continuous Download)

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

1 Open the dashboard of jenkins

2 Click on New item---->enter item name as "Development"

3 Select Free style project--->OK

4 Go to Source code management

5 Select Git

6 Enter the github url where developers have uplaoded the code

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

7 Apply--->save

8 Go to the dashboard of Jenkins

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

This job will downlaod all the code uplaoded by the developers into

github

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 steps

5 Click on Invoke top level maven targets

6 Enter the Goal as: package

7 Click on Apply--->Save

8 Go to the dashboard of Jenkins

9 Go to Development job and click on Build icon

This job will create an artifact from the code that is

downloaded. 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 dashboard of Jenkins

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

9 Go to Post Build actions

10 Click on Add Post Build action

11 Click on Deploy war/ear to container

   war/ear files: **/*.war

   context path: qaapp

   Click on Add container---->Select tomcat9

   Enter username and password of tomcat

   Tomcat url: private_ip_of_qaserver:8080

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 QA server tomcat

15 To access the application running on QA server tomcat from browser

  public_ip_of_qaserveR:8080/qaapp

===

Day 3

===

Stage 4 (Continuous Testing)

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

1 Open the dashboard of Jenkins

2 Click on New item--->Enter item name as "Testing"

3 Select Free type project---->OK

4 Go to Source code management

5 Select Git

6 Enter the github url where testers have uploaded the code

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

7 Go to Build section

8 Click on Add build step

9 Click on Execute shell

```
java -jar path/testing.jar
```

10 Click on Apply--->save

11 Go to the dashboard of jenkins

12 Go to the Testing job

13 Click on Build icon

This job will download the selenium test scripts and execute them

----

Linking the Development job with Testing job

----

1 Go to the dashboard of jenkins

2 Go to Development job--->Click on Configure

3 Go to Post Build actions

4 Click on Add Post Build actions

5 Click on Build other projects

6 Enter the project name "Testing"

7 Apply--->Save

  This is called as upstream/downstream config

Copying artifacts from Development job to Testing job

==

1 Open the dashboard of Jenkins

2 Click on Manage jenkins

3 Click on Manage plugins

4 Go to Available section

5 Search for "Copy Artifact" plugin

6 Click on Install without restart

7 Go to the dashboard of Jenkins

8 Go to the Development job

9 Click on Configure

10 Go to Post Build actions

11 Click on Add post build actions

12 Click on Archive the artifacts

13 Enter files to be archived as : ***.war

14 Click on Save

15 Go to the dashboard of Jenkins

16 Go to the Testing job

17 Click on Configure

18 Go to Build section--->Add build step

19 Click on Copy artifacts from other projects

20 Enter project name as "Development"

21 Apply---->Save

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

Stage 5 (Continuous Delivery)

----

1 Open the Dashboard of Jenkins

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

3 Go to Post build actions

4 Click on Add post build actions

5 Click on Deploy war/ear to container

6 war/ear file: **/*.war

  Context path: prod app

  Select container tomcat9

  Enter username and password for tomcat9

  Tomcat url: private_ip_of_prodserver:8080

7 Apply--->Save

Day 4

===========

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 mange 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 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)

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

==

1 Update the apt repository and install java--

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

2 Add the jenkins repository key to the apt repository--

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

3 Add the debain repository address to jenkins.list file--

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

4 Update the apt repository--

```
  sudo apt-get update
```

5 Install Jenkins--

```
  sudo apt-get install -y jenkins
```

Setup of Jenkins on tomcat

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

1 Update the apt repository--

```
  sudo apt-get update
```

2 Install tomcat9 and tomcat9-admin--

```
  sudo apt-get install -y tomcat9 tomcat9-admin
```

3 Download jenkins.war--

```
  wget https://get.jenkins.io/war-stable/2.263.4/jenkins.war
```

4 Copy the jenkins.war into tomcat webapps folder--

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

5 To access tomcat from browser--

```
 public_ip_of_jenkinsserver:8080
```

6 To access jenkins deployed on tomcat--

```
 public_ip_of_jenkinsserver:8080/jenkins
```

==

Day 5

===

==

Master Slave architecture

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

This is used to ensure that the performance of Jenkins does not go down even when we run multiple jenkins jobs parallelly

Setup

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

1 Create a new AWS ubuntu instance and name it slave

2 Install the same version of java as present on master--

```
  sudo apt-get update

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

3 Download slave.jar from master(Jenkins server)--

```
Wgethttp://private_ip_of_jenkinsserver:8080/jnlpJars/slave.jar
```

4 Give execute permissions on slave.jar--

```
chmod u+x slave.jar
```

5 Create a folder for jenkins to store info related to the job--

```
mkdir workspace
```

6 Setup password less ssh between master and slave

On Slave

----

a) Set password for default ubuntu user--

```
sudo passwd ubuntu
```

b) Edit the sshd config file--

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

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

c) Restart ssh service--

```
sudo service ssh restart
```

On Master

----

a) Generate the ssh keys--

```
ssh-keygen
```

b) Copy the keys—

```
ssh-copy-id ubuntu@private_ip_of_slave
```

This command will copy the content of the public keys and paste it on the slave machine in a file called "authorised_keys"

7 Open the Jenkins dashboard---->Click on Manage Jenkins

8 Click on Manage nodes and clouds

9 Click on New node--->Enter some node name--->Select Permanent agent--->OK

10 Enter remote root directory: /home/ubuntu/workspace

11 Labels: my slave (This is an alias for the slave and this alias should be linked with the jenkins job)

12 Launch Method: Select "Launch agent via execution of command on master" ssh ubuntu@private_ip_slave java -jar slave.jar

13 Click on Save

14 go to the dashboard of Jenkins

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

16 Go to General section

17 Check "Restrict where this project can be run"

18 Enter the slave label as my slave ---->Save

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

Pipeline as Code

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

This is the process of running all the stages from the level of

a Groovy script code called as "Jenkinsfile" This Jenkinsfile is genrally uplaoded into the remote git repository from where it triggers all the CI-CD stages.

Advantages

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

1 Since it is code it can be uploaded into a version control

system form where all the team members cab review and edit and still maintin multiple versions.

2 Jenkins files can with stand planned and unplanned restart

of the Jenkins master.

3 Since the jenkins jobs are stored in the form of a Jenkins file

in git hub even if the jenkin server crashes we will not loose much.

4 Jenkins files can perfrom all stages of ci-cd with minimum number of plugins and hence they are much faster.

5 We can handle real world challanges like if conditions, exception handling etc

==

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

Pipeline as Code can be implemented in 2 ways

1 Scripted Pipeline

2 Declarative Pipeline

Scripted Pipeline syntax

```
node('master/salve')
{
 stage('Stage name in ci-cd')
 {
    Exact groovy code to implement this stage
 }
}
```

Declarative Pipeline Syntax

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

```
pipeline
{
 agent any
 stages
 {
    stage('Stage name in CI-CD')
```

```
      {
         steps
         {
            Exact groovy code to implement this stage
         }

      }
  }
}
```

===

Day 6

===

Scripted pipeline to perform CI-CD

===

```
node('master')
{
    stage('Continuous Download')
    {
       git 'https://github.com/intelliqittrainings/maven.git'
    }
    stage('Continuous  Build')
    {
       sh 'mvn package'
    }
    stage('Continuous Deployment')
    {
     sh 'scp
/home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp
/target/webapp.war
ubuntu@172.31.3.108:/var/lib/tomcat9/webapps/qa
app.war'



    }
    stage('Continuous  Testing')
    {
       git
'https://github.com/intelliqittrainings/FunctionalTesting.git'


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

```
    }
    stage('Continuous Delivery')
    {
        sh 'scp
/home/ubuntu/.jenkins/workspace/ScriptedPipeline/webapp
/target/webapp.war
ubuntu@172.31.2.173:/var/lib/tomcat9/webapps/prodapp.w
ar'
    }

}
```

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

Day 7

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

In most cases the Jenkins file is part of the github and from there it will trigger all the stages of CI-CD

1 Download the maven git repo--

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

2 Move into this directory and delete the .git folder--

```
  cd maven
  rm -rf .git
```

3 Initilise a new git repository--

```
  git init
```

4 Create a file with name Jenkinsfile and copy paste the groovy code--

```
 vim Jenkinsfile
```

Copy paste the Groovy code that we created in Jenkins

5 Send it to the stagging area and local repository--

```
  git add .
  git commit -m "a"
```

6 Sigin into github.com and create a new repository

7 Push the existing local repository into gtihub

8 Open the dashboard of Jenkins

9 Click on New item--->Enter item name as "Scripted pipeline"

10 Select Pipleline--->OK

11 Go to Pipeline section

12 In definiton select Pipeline script from scm

13 Select SCM as Git

14 Repository url: Enter the git hub url where the "maven" folder is uploaded

15 Go to Build triggers

16 Click on Poll SCM

  * * * * *

17 Save

 Now when ever the changes are done by developer to the maven code and pushed into github then Jenkins file from github will trigger all stages of CI-CD

==

Scripted Pipeline Code

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

https://github.com/intelliqittrainings/maven/blob/master/Jenkinsfile

https://github.com/intelliqittrainings/maven/blob/master/Jenkinsfile1

==

Cat light notifications

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

This is a third party s/w that is integrated with Jenkins to receive notifications in the form of Desktop popup msgs.Cat light is a client side s/w and it has to installed on all team members machines.

1 Download cat light from

https://catlight.io/downloads

2 Install it

3 It will display the list of CI tool--->Select Jenkins

4 Enter the public ip of jenkinsserver:8080

Enter username and password to connect to jenkins

5 In cat light Select the job for which we need notifications

6 Run that job in jenkins

==

Webhooks

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

If we want github to send notifications to Jenkins whenever code is commited we can use webhooks

1 Opengithub.com

2 Go to the Maven repository where the source code is uploaded.

3 Click on ... on the top right corner---->Click on Settings

4 Select Webhooks

5 Click on Add webhook

6 Payload url: public_ip_jenkinsserver:8080/github-webhook/

7 Content type: Select application/json

 8 click on save

==

Day 9

==

Declarative Pipeline Script for CI-CD

```
pipeline
{
    agent any
    stages
    {
        stage('Continuous Download')
        {
            steps
            {
                git 'https://github.com/intelliqittrainings/maven.git'
            }
        }
        stage('Continuous  Build')
        {
           steps
           {
             sh label: '', script: 'mvn package'
           }

        }
        stage('Continuous Deployment')
        {
           steps
           {
               sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
app/target/webapp.war
ubuntu@172.31.31.15:/var/lib/tomcat8/webapps/testwebap
p.war'
           }

        }
        stage('Continuous  Testing')
        {
            steps
            {
                git
'https://github.com/intelliqittrainings/FunctionalTesting.git'
                sh label: '', script: 'java -jar
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/testi
ng.jar'
            }
        }
        stage('ContinuousDelivery')
        {
           steps
           {
               input message: 'Waiting for Approval from the DM!',
submitter: 'naresh'
               sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
app/target/webapp.war
ubuntu@172.31.26.41:/var/lib/tomcat8/webapps/prodweba
pp.war'
           }
        }
    }
```

```
}
```

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

Declarative Pipeline with Post conditions

==

```
pipeline
{
   agent any
   stages
   {
      stage('Continuous Download')
      {
         steps
         {
            git 'https://github.com/intelliqittrainings/maven.git'
         }
      }
       stage('Continuous Build')
      {
         steps
```

```
          {
             sh label: '', script: 'mvn package'
          }
      }
      stage('Continuous Deployment')
      {
          steps
          {
             sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
app/target/webapp.war
ubuntu@172.31.31.15:/var/lib/tomcat8/webapps/testwebap
p.war'
          }
      }
      stage('Continuous Testing')
      {
          steps
          {
             git
'https://github.com/intelliqittrainings/FunctionalTesting.git'
             sh label: '', script: 'java -jar
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/testi
ng.jar'
```

```
          }
        }


    }
    post
    {
        success
        {
            input message: 'Waiting for Approval!', submitter:
'naresh'
              sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
app/target/webapp.war
ubuntu@172.31.26.41:/var/lib/tomcat8/webapps/prodweba
pp.war'
        }
        failure
        {
            mail bcc: '', body: '', cc: '', from: '', replyTo: '', subject:
'Jenkins CI-CD Failed', to: 'gandham.saikrishna@gmail.com'
        }
    }



}
```

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

Declarative Pipeline with Exception Handling

===

```
pipeline
{
    agent any
    stages
    {
        stage('Continuous Download')
        {
            steps
            {
                script
                {
                   try
                {
                    git
'https://github.com/intelliqittrainings/maven.git'
                }
                catch(Exception e1)
                {
                    mail bcc: '', body: 'Jenkins is unable to download
from remote github', cc: '', from: '', replyTo: '', subject:
'Download failed', to: 'gitadmin@outlook.com'
                    exit(1)
                }
              }


           }
        }
         stage('Continuous Build')
        {
           steps
           {
              script
              {
                try
                {
```

```
                sh label: '', script: 'mvn package'
             }
             catch(Exception e2)
             {
                mail bcc: '', body: 'Jenkins is unable to create an
artifact from the code', cc: '', from: '', replyTo: '', subject:
'Build failed', to: 'developers@outlook.com'
               exit(1)
             }
           }


        }
      }
      stage('Continuous Deployment')
      {
         steps
         {
            script
            {
               try
               {
                  sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
```

```
app/target/webapp.war
ubuntu@172.31.31.15:/var/lib/tomcat8/webapps/testwebap
p.war'
                }
                catch(Exception e3)
                {
                    mail bcc: '', body: 'Jenkins is unable to deploy
into tomcat on the QaServers', cc: '', from: '', replyTo: '',
subject: 'Deployment failed', to: 'middleware@outlook.com'
                    exit(1)
                }
              }


          }
        }
        stage('ContinuousTesting')
        {
          steps
          {
            script
            {

              try
```

```
                {
                  git
'https://github.com/intelliqittrainings/FunctionalTesting.git'
                  sh label: '', script: 'java -jar
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/testi
ng.jar'
                }
                catch(Exception e4)
                {
                  mail bcc: '', body: 'Functional testing of the app
on QAServers failed', cc: '', from: '', replyTo: '', subject:
'Testing failed', to: 'testers@outlook.com'
                  exit(1)
                }
             }


          }
       }
       stage('ContinuousDelivery')
       {
          steps
          {
             script
```

```
            {
              try
              {
                  input message: 'Waiting for Approval!',
submitter: 'naresh'
                  sh label: '', script: 'scp
/home/ubuntu/.jenkins/workspace/DeclarativePipeline/web
app/target/webapp.war
ubuntu@172.31.26.41:/var/lib/tomcat8/webapps/prodweba
pp.war'
               }
               catch(Exception e5)
               {
                   mail bcc: '', body: 'Unable to deploy into
ProdServers', cc: '', from: '', replyTo: '', subject: 'Delivery
failed', to: 'delevery@outlook.com'
               }
            }
         }
      }


   }
```

}

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

Day 10

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

Email Notification

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

1 Open the dashboard of Jenkins

2 Click on Manage Jenkins

3 Click on Configure System

4 Go to Email notifications

5 SMTP server: smtp.gmail.com

6 Click on Advanced

7 Check Use SMTP authentication

8 Enter gmail username and password

9 Check Use SSL

10 Check Test configuration by sending test e-mail

   enter your gmail id and click on Test Configuration

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

Multibranch Pipeline

===

When developers uploads seperate functionalities code on different branches and we want to trigger CI-CD for each branch parallelly we can use Mult Branch pipleline jobs

Developers Activity

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

1 Clone the mave repository

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

2 Move into this repository and delete the .git folder

```
cd maven
rm -rf .git
```

3 Initilise a new repository--

```
git init
```

4 Create a jenkins file for master branch--

vim jenkins file Place all the stage of CI-CD that we should perfrom on that branch

5 Stage and commit it--

```
git add .
git commit -m "a"
```

6 Create a Loans branch and move into it--

```
git checkout -b Loans
```

7 Create another Jenkisn file--

```
vim Jenkinsfile
```

Place the stages of CI-CD that we require for branch

8 Stage and commit it--

```
git add .
git commit -m "b"
```

9 Push this into remote github

===

Jenkins Admin Activity

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

1 Open the dashboard of Jenkins

2 CLick on New item

3 Enter some item name

4 Select Multi Branch Pipeline--->OK

5 Go to Branch Sources--->Select Git

  Enter github url where developers have pushed the code along with

  all the Jenkinsfiles

6 Click on Scan Multi branch pipeline triggers and select the time

  interval as 1 minute

7 Apply--->Save

===
====