To install JenkinsServer

1) Java
2) Git
3) Ant, Maven
4) Jenkins

To install QAServer and ProdServer

1) Tomcat9
2) Tomcat9-admin

To install Jenkins

Setup of Jenkins for CI-CD

1. Create 3 ubuntu servers on AWS and name them as JenkinsServer, QAserver, and ProdServer.
2) Connect to the JenkinsServer
3) Update the apt repository

```
sudo apt-get update
```

4) Install java

```
sudo apt-get install -y openjdk-11-jdk
```

5) Install git and maven

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

6) Download jenkins.war file

```
wget (Past url of Jenkins from the website of Jenkins)
```

7) To start jenkins

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

8) ~~To access jenkins~~

8) To access jenkins from browser

```
public_ip_of_jenkins : 8080
```

9) Unlock jenkins by entering the admin password

10) Install suggested plugins

11) Create first admin user

Setup of tomcat on Qa and Prod Server

1) Connect to Qa Server

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-user.xml file.

```
cd /etc/tomcat9
```

```
sudo vim tomcat-users.xml
```

→ Delete the entire data and add below data

<tomcat-users>

<user username="sudarshan" password="sudarshan@455" roles="manager-script"/>

</tomcat-users>

6) Restart the tomcat service

```
sudo service tomcat9 restart
```

7) To acces tomcat from browser

```
public_ip:8080
```

Repeat the above 7 steps on prodServer

* To edit the file or enter some data in the file we should go into insert mode

```
i
```

* To create a file or open file

```
vim filename
```

* To save and quit

Esc : wq ↵

* To quit without saving

Esc : q! ↵

Jenkins Id and Password

Id: Sudarshansw7

Password: 09pq1a0455

QA Server Id and Password

To open the tomcat in gui mode

<tomcat-users>

<user username = "admin" password = "admin"

roles = "manager-script, ~~manager~~ admin -gui,

manager-gui" />

</tomcat-users>

freestyle Project flow Algorithm

Job-I

1) Take url from github
2) Build the code into packages
3) Deploy into QAServer

Testing - job - II

1) copy url from github
2) Build the code by executing the shell
   java -jar (pass the url of the console o/p
3) Now connect the Job-I to Job-II for executing continously
4) Now go to job-I and archive the war file by **/*.war
5) Now go to Job-II and copy artifacts
6) Now deploy these in ProdServer

---

1) To deploy, we have to post anything from job-I to job-II, use post build action

→ To copy and build we use build action

Git:-

→ It is a distributed version controlling system commonly used in s/w development.

→ It helps us to track the changes in source code files.

→ It creates, branches, merges with master and revert to previous versions of your code.

→ It enables collaboration of multiple developers on a project.

Git { It can preserve oldee and later versions of the code. }

1) To create user

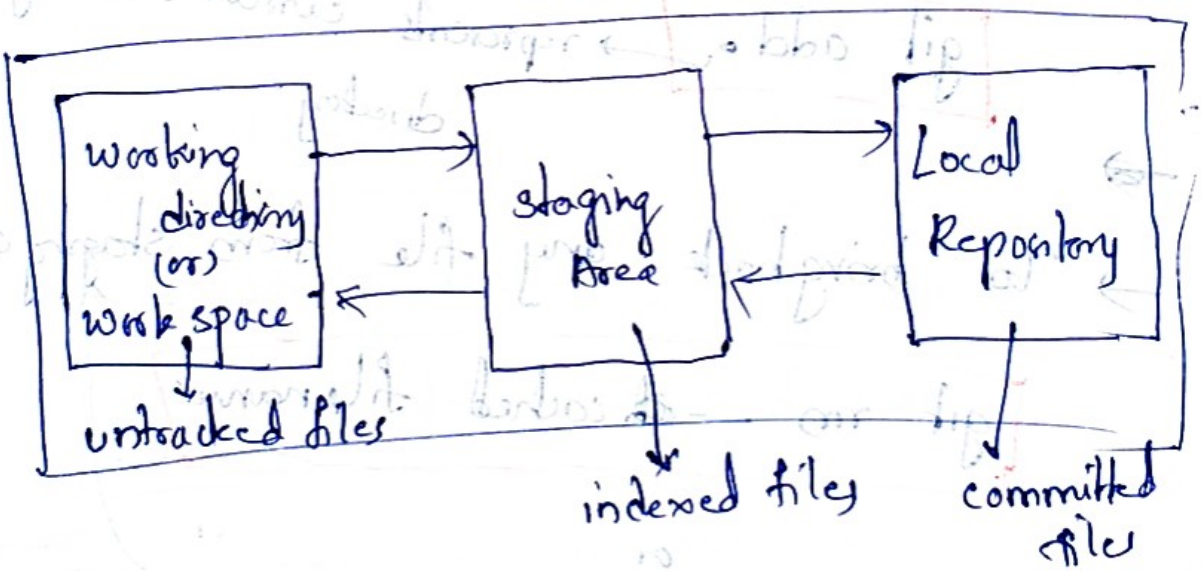
```
git config --global user.name "sudarshan"
```

2) To create email

```
git config --global user.email "sudarshan@gmail.com"
```

3) To know the list of configurations present

```
git config --global --list
```



4) To see the folder of your machine into git

```
cd (Past path of the folder)
```

→ In order to execute the git commands on a new folder, we have to initialize it

→ It also special folder called as .git → in this it store git commands
→ It creates a hidden folder

```
git init
```

→ To know the status of git on that file

```
git status
```

→ To send one file into staging area

```
git add filename
```

→ If 'n' no. of files, then

```
git add .
```
→ represents current working directory

→ To bring back any file from staging area

```
git rm --cached filename
```

or

```
git reset filename
```

or

```
git reset .
```

→ To bring all the

→ To move all the files in local repository

```
git commit -m "First version or any name"
```
→ message

→ To see all the commits we have perfor
-med

```
git log
```

→ To see in oneline all the commits

```
git log --oneline
```

→ To create a branch

```
git branch branch_name
```

→ To move into a branch

```
git checkout branch_name
```

→ To create a branch and move into it

```
git checkout -b branch_name
```

→ To merge a branch

```
git merge branch_name
```

→ To delete a branch that is merged

```
git branch -d branch_name
```

is called soft delete.

→ To delete a branch that is not merged

```
git branch -D branch-name
```

is called hard delete

→ Clone
Fetch
Pull

* To push data into remote server just use `git push`
no need of always creating tokens everytime

① Clone

when you execute clone command it will copy all the files from remote repository.

```
git clone (url of repository)
```

② Fetch

It will copy only the modified files from remote repository, and place in a remote folder, it is automatically created when you execute this command.

```
git fetch
```

→ To see all branches

```
git branch -a
```

After fetch is over, then go into the remote branch (which is created by default) open it, then you will see the modified data, if you want that modified data, then you can merge with master branch

(8) Pull

It directly downloads the modified file and merge with master branch

```
git pull
```

→ To move the head from one commit to another commit

```
git reset --hard paste Id of that commit
```

→ git reflog

It gives the list of all the commit

→ Soft reset :-

It takes one step back, the file was present in staging area, and head was at previous commit

→ Mixed reset :-

It takes two steps back, and the file was in untracked files and the head was in previous commit

git reset --soft (Id of the previous commit)

git reset --mixed (Id of the previous commit)

Git merging and rebasing / fast forward merging

merge :- All the commits coming from the branch are merged with the master branch, they are based on time stamp

rebase! - All the commits coming from the branch are fast forwarded, i.e, they are projected as the latest commits and the head is pointing on it,

→ go to the branch and operate rebase command

```
git rebase master
```

→ then go to the master and fire the command

```
git merge branch_name
```

Drawback :- All the commits are moving into the parent branch (master) with rebase and merge.

→ I have to selectively pick the commits which I want, so I go with the cherry pick

(*) cherry pick :-

We can take the whichever commit we

want, then go to master branch and do

```
git cherry-pick commitId1 commitId2...
```

(*) • gitignore

I created some files and I don't want to send them and share with them, If we store some files in .gitignore file, these files are unaccessible by git

```
vim .gitignore
```

open that file and save all the files you want to ignore,

→ To remove files from .gitignore and share them, then go with and delete the file you want

```
vim .gitignore
```

and delete the file you want and add it to staging area

(*) STASH :- It is also one way of hiding data. / It has ability to hide files

eg:- Currently we are working on some loans project and created some commits and suddenly we have to stop that and work on another payments project, in this case we use stash,

→ It stores the previous work commits temporary, after finishing other work we can resume then work

→ It also hide data from git

① To stash all the files in the staging area:

```
git stash
```

② To stash all the files in the staging area, untracked section

```
git stash -u
```

③ To stash all the files in the staging

area, untracked section and .gitignore

```
git stash -a
```

(4) To see the list of all the stashes

```
git stash list
```

(5) To unstash the latest stash

```
git stash pop
```

(6) To unstash an older stash

```
git stash pop @{stash no}
```

(*) Squash :-

It has ability to merge commits with another commit,

→ but I created many commits but I want to send them as a single commit.

→ In this the first commit will not be modified or can't merge

`git rebase -i HEAD~no. of commits`

eg:- `git rebase -i HEAD~4`

In this latest commits are merged with older commits

→ In this remove that word pick and replace it with squash

(*) Tags :-

→ To identify the version which is important commit related to release activity

`git tag name of the tag`

eg: `git tag release 2.0`

`git tag name of the tag commit ID`

→ To see list of all the tags

`git tag`

→ To delete a tag

`git tag -d name of the tag`

(*) revert

→ It is similar to reset but small difference, i.e

→ It undoes the changes made in the commit.

→ Suppose if we have inserted done any changes in a commit

`git revert commit ID`

→ `rm -rf .git` → to exit from the master to our own window

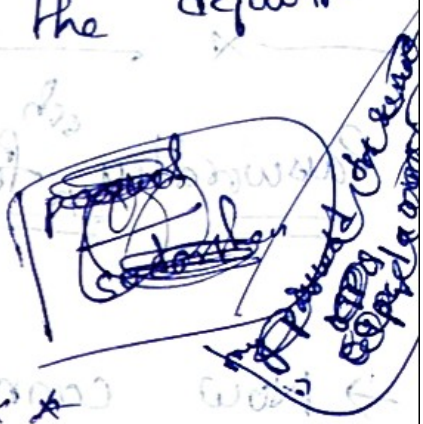
→ No need of creating tokens to push data into remote server after "git push origin master" command, just do "git push" command

To connect one server to another Server on Linux

1) Connect to the server and type the command `whoami`, It gives the default user.

2) Now create password for the default user

`sudo passwd Username`



Then type password *****

→ Now `sudo vim /etc/ssh/sshd_config`

Here search for password authentication

Then change no to yes

→ Now `sudo service ssh restart`

Now connect Server1 and connect, then with Server2 address then the user password

Now connect Server 1, and to connect Server 2 from Server 1

```
ssh username@ pvt-ip-address of
                    Server 2
```

It asks for the password of server 2,

Passwordless ssh from server 1 to Server 2

→ Now connect to Server 1 and generate ssh keys.

→ keys will be available in folder .ssh

```
ssh-keygen
```

→ Now copy the keys

```
ssh-copy-id username@ipaddress of
                    server 2
```

pvt ip of server 2

→ Now from server 1 give the command as

```
.ssh Username@ ip-add-of server 2
```

→ To know the working directory

`pwd`

→ To create a file, it creates emty file

`touch filename`

→ We can create more no! of files with touch

`touch file1 file2 file3 ....`

→ To create a file and to store some data in it.

`cat > filename`

→ To add data into a existing file, it is called as append

`cat >> filename`

→ If you do cat > filename of a existing file, it overwrites

→ `ctrl+d` → end of the file

→ To see only the content of the file

`cat filename`

→ To create a directory

`mkdir name of directory`

→ To move into a directory `cd`

→ To create multiple directories

```
mkdir -p d1/d2/d2/d3/d4/d5
```

→ To check the directories within the directory tree command is used, install tree software in your server.

```
tree directoryname
```

→ To delete file

```
rm filename
```

→ To delete directories, we use

```
rm -r directoryname
```

r → recursive, because in directory there will be sub directories

→ . represents current working directory
.. represents parent directory

```
cd ..
```

```
cd ../../../..
```

so on it will take you to parent directory

→ In linux there is no concept of extensions

eg: file.txt → extensions

→ whenever any file starts with . it is a hidden file

eg:- `touch .filename`

It hidden the file in linux,

→ To see all files `ls -a` → all

→ `ls -l` → It shows long listing of files, shows metadata, it means data about the data

→ To copy a file

`cp filename filename`

sourcefile destinationfile.

→ To copy directory and subdirectories

`cp d1/tmp` `cp d1 /tmp`

All the subdirectories and main directory is copied into tmp folder

~~cp -r d1 /tmp~~

```
cp -r d1 /tmp
```

→ To do cut paste operations mv command is used

eg:- `mv file1 file2`

(or)

`mv file /tmp`

To change permissions of a file

→ In this we have owner, group, other

eg:- drwx rwx r-x 3 ubuntu 4096 Aug 26 3:05 d1
-rwx rwx r-x 2 ubuntu " "

→ If first letter starting with d, then it is a directory

→ If it starts with - then it is ordinary file.

drwx rwx r-x
- owner permissions
- group permissions
- other permissions

r — 4
w — 2
x — 1

To change the permissions

eg:- `chmod 750 file1`

owner < r, w, x ; group < read, execute ; other → no permissions

chmod permissions no filename

Another way of changing permissions is with alphabetical order

u → owner
g → group
o → other

add permissions +
remove " —
assign " =

eg:- chmod u+x, g-rw, o=x file1

→ To change the permissions of main directory and sub directory

chmod -R permission no directory name

Head!- It captures only top 10 lines of a file

```
head filename
```

Tail!- It captures only last 10 lines of a file

```
tail filename
```

eg!- Head

```
head -3 filename
```

→ it displays only top 3 lines

```
tail -3 filename
```

→ it displays last 3 lines

grep!- It is used for capturing a specific word/string.

eg: grep mango filename

eg: grep spool filename

→ To search in uppercase or lower case

eg:

```
grep -i SPOOL filename
```

↳ ignore case

```
grep -n SPOOL filename
```

↳ all those lines where that word is present

`grep -v spool filename`

↳ all those lines where that word is not available

cut :- It is used to cut the data into columns.

```
cut -d ":" -f 1,7 filename
cut -d " " -f 1,7 filename
```

wc → word count

`wc filename`

→ If you want to see only no. of lines `wc -l filename`

" " no. of words `wc -w filename`

" " " characters `wc -c filename`

Sort :- It is used to arrange the data in acending order or decending order

sort filename

→ To do numerical sort

`sort -n filename`

→ `df -hT` is used to check the storage space

→ To change the name of the host

`sudo hostnamectl set-hostname host_name`

Pipelines

→ We perform the pipelines using groovy script.

→ All the steps of CI-CD are done with groovy script.

→ We have two types of scripts in groovy script.

→ It has version controling on the jenkins file.

→ In freestyle projects, when we are runing all these stages, sometimes it may restart, some code is lost, some stages got failed,

→ In pipelines, using script/program we can do all these stages without interuption

→ To implement all these CI-CD forms a programe, this programs is called Jenkins file, this file can be uploaded into git repository and anyone can download and makes changes.

→ It is giving the team members to review the code and edit the code

→ when the CI-CD is runing in the freestyle, if someone is installing the plugins, the job will be aborted.

→ And more plugins are needed in freestyle, these plugins consume some amount of CPU, hardware, it may cause slower proces of CI-CD.

→ We have 60,000 plugins, these plugins comes from different vendors, what is the gaurantee that these plugins are genuine and virus free.

→ In pipeline, we can do all the CI-CD stages without plugins,

→ Pipelines are of two types pipelines they are i) Scripted pipeline

ii) Declarative pipeline

Pipe Line as Code

1) Scripted PipeLine :

```
node ('master/slave')
{
  stage ('stage name in CI-CD')
    {
    Actual groovy code to implement this stage
    }
}
```

2) Declarative Pipe Line

```
Pipeline
{
  agent any
  stages
    {
    stage ('stage name in CI-CD')
      {
        steps
          {
          Actual groovy code to implement this stage
          }
      }
    }
}
```

→ Scripted pipeline is more powerfull,

→ Declarative pipeline more later versions, all the organisations use declarative pipeline

→ Scripted pipeline is loosly structured and powerfull,

→ Now with pipeline perform

(i) continous download

(ii) continous build

(iii) continous deployment in QA server

(iv) continous Testing

(v) continous Delivery in Prod Server

Poll Scm :-
↳ check → Source Code Management

→ If I give this command, it immediately triggers the CI-CD flow automatically therefore I have to specify time date month,

→ The when you clone from remote git repository, there is a file called .git in that file the linking data is there so delete that .git file, then it will become a normal folder in your machine.

→ To trigger all the stages automatically then in build triggers, there is a option Poll Scm, click on that and give the time schedule as

* * * * *

It means every minute, every hour every week and every first day of the month

→ Generally the jenkins file is uploaded by the devops engineer into git repository at that time specify

→ Pipeline Script from SCM

→ SCM → git

→ url of the repository

→ In this process Jenkins is checking for every minute and triggering the CI-CD process.

Suppose that developer is uploading the code and modifying for every once in 10 days, in this Jenkins has trigger every minute, for all days, it is consuming some resources and simply triggers, therefore no use.

→ So to overcome that we go for webhooks process,

Webhooks:-

→ In this, the git remote repository will send a notification when the developer is uploaded code or modifying the code,

→ Then Jenkins starts triggering after receiving the notification.

→ Now go to git remote repository

→ Click on ... → settings → webhooks

→ add webhook

→ Payload URL

~~http://ip-public-Jenkins:8080/github-webhook/~~

http://public-ip-Jenkins:8080/github-webhook/

→ Content type

application/json

→ Add webhook

Interview

(*) How can you integrate Jenkins with git repository in such a way when developer makes changes to the code automatically it should trigger all the CI-CD stages.

Ans! - With Webhooks

→ when the Jenkins file is present in the remote repository, then select Both scm Pipeline Script from scm

→ To copy files from one server to another server use Scp command

→ In tomcat9 server, the artifact which is created in Jenkins server is copied in /var/lib/tomcat9/webapps, in this folder.

→ Scp works when you have SSH connections

→ Scp = Secure copy

→ To switch to any user sudo su - username

Syntax:

scp

```
scp Source file  username@ipaddress:/destination file name
```

eg:- To copy the artifact build in Jenkins server to QA server

source file where package is present.

```
scp /var/lib/Jenkins/workspace/scriptedpipeline/webapp/target/webapp.war  ubuntu@ipadd.of QA server:/var/lib/tomcat9/webapps/testing.war
```

destination file any file name

→ We don't have permissions to deploy into production environment, so we have to wait for approval of Delivery manager

→ So to get approval, just add the lines in the code you developed,

→ go to snippet pipeline generator and search for input: wait for interactive input

To send email notifications whenever failure happens

(i) Manage Jenkins
(ii) System
(iii) Search for email notification

E-mail Notification

→ smtp.gmail.com

↓ this mail will be given by the organization, they have their own mailing systems

(iv) Click on advanced

→ Use SMTP Authentication

Username

sudarshansw7@gmail.com

Password

(v) Now click on SSL

SMTP Port

465

→ SMTP works on port no. 465.

(vi) After that go to the project configure in that Post build actions e-mail notification is there, give details as shown in that popups

Post:-

This is applied only in declarative pipelines to show success message, failure message, about, always, this will be written after all the stages of declarative script

post
{
success
{
}
failure
{
}
abort
{
}
always
{
}

Scripted Pipeline

```
node('built-in')
 {
 stage('continous download')
    {
    git 'https://github.com/sudarshansw7/maven.git'
    }
 stage('continous build')
    {
     sh 'mavn package'
    }
stage('continous deployment')
    {
  sh 'scp /var/lib/jenkins/workspace/scriptedpipeline
  /webapp/target/webapp.war ubuntu@ip add of
  server:/var/lib/tomcat9/webapps/testapp.war'
    }
stage('continous testing')
    {
  git 'https://github.com/sudarshansw7/myfunctional
                                    testing"
  sh 'java -jar /var/lib/jenkins/workspace/scripted
      pipeline/testing.jar'
    }
```

```
stage ('continous delivery')
  {
sh 'scp /var/lib/jenkins/workspace/scripted
   pipeline/webapp/target/webapp.war ubuntu@
   ipadd of prodServer : /var/lib/tomcat9/
   scripted pipeline/webapps/ prodapp.war'
  }
```

Deployment in QA and Prod Server

Shortcut remember this only

```
sh 'scp take file address from build job
   ubuntu@ipadd of QA or Prod Server : /var/lib/
   tomcat9/webapps/ testapp.war (or)
                   prodapp.war
```

Permanent installation of Jenkins

→ Go to google and type Jenkins installation on ubuntu

→ Go to Linux in that search for Debian/ubuntu

→ Copy those commands and paste those commands, then jenkins will be installed

→ Permanent Jenkins user will be present in permanent installation of Jenkins

→ To go into jenkins user

```
sudo su - jenkins
```

→ In slave machine also install same flavor of java

→ For jenkins password will be present in /var/lib/Jenkins/secrets/initialAdminPassword copy that

→ sudo vim patchee, you find password

Connections

Passwordless connections from Jenkins to QaServer/ProdServer

→ Create passwords in QaServer/ProdServer

→ sudo passwd username

→ enter password

→ sudo vim /etc/ssh/sshd_config

→ change no to yes

→ sudo service ssh restart

→ exit

→ Now go to Jenkins server

→ sudo su - jenkins

→ Then generate keys

ssh-keygen

→ ssh-copy-id username of Qa/Prodserver @ pvt-ip-address of Qa/Prod

→ This will establish passwordless connection

→ To copy

→ We are copying the pipeline script into cd /var/lib folder

ls -ld tomcat9/

sudo chmod 777 -R tomcat9/

Post conditions:-

→ It can be applied only in declarative pipeline.

```
Pipeline
{
 agent any
 stages
    {
     stage ('a name')
        {
         steps
            {
              groovy code written here
            }
        }
```

Post conditions:-

It can be implemented in declarative pipeline only

```
pipeline
    {
    agent any
    stages
    {
     stage ('CI-CD name')
        {
        steps
            {
            Actual groovy code written here
            }
        }
    }
    post
    {
     success
        {
        Here write the code for deploying into prod
        }          environment
     failure
        {
          send mail to corresponding teams -> go to
                snippet, search for mail
        }
}
```

→ In this if anyone stage is failed it stops execution and sends notification,

→ which stage are we failing we can't understand this is the serious problem.

→ for every stage we have notice the failures and success and fix the problems

→ If any particular stage fails it have to inform to a particular team members

→ This can be done by exception handling

Exception handling

```
try
{

}
catch (Exception e)
{

}
```

Declarative pipeline

```
Pipeline
{
  agent any
  stages
    {
      stage ('stage name')
        {
          steps
            {
              script
                {
                try
                  {
                  Actual groovy code here
                  }
                catch (Exception e)
                  {
                    mail to particular team
                  } exit(1)
```

Scripted pipeline

```
node ('built-in')
  {
  stage ('stage name')
    {
     try
       {
         Actual groovy code there
       }
     catch (exception e)
       {
         mail to particular team
        exit(1)
       }
    }
  }
```

Shared libraries

→ These are created for reuseability purpose.

→ In any jenkins project whether it is scripted pipeline or declarative pipeline there all certain stages

are same. with some mind differences like cont download, cont deployment,

→ In cont download we are using the same url but ~~different~~ some different name

eg:- https://github.com/sudarshansw7/maven.git

https://github.com/sudarshansw7/functional testing.git

→ For this we can write a user defined functions and we can call it many times.

User defined functions for maths

```
def sum(a,b)
  {
    echo "The sum of ${a} and ${b} is
    ${a+b}"
  }
def mul(a,b)
{
echo "The multiplication of ${a} and ${b} is
    ${a*b}"
```

etc. echo → refers to print in jenkins

→ There above functions can be reused many times, likewise in jenkins also we create user defined functions and we can use them.

→ These userdefined functions are created in a remote git repository, whenever we want to use just call those functions and use them.

To create a shared repository

→ Go to your github account
→ Create new repository
→ Within that repository we have to create a folder structure
→ Click on create file, within that create vars folder within that any filename but extension should be .groovy
→ In that file create user defined functions code, then commit changes.

→ Then take the url of that shared library then

→ go to manage jenkins and click on system

→ In that system section, search for global pipeline Libraries

Name : give anyname

Default version : main

modern scm

then specify the url

@Library('name of your sharedlibrary') _

```
pipeline
{
 agent any
 stages
  {
   stage('name of the CI-CD')
    {
     steps
      {
       script
        {
         groovy function
```

Declarative pipeline:-

```
@Library('name of your shared library')_
pipeline
  {
   agent any
   stages
     {
      stage ('name of the CI-CD stage')
        {
         steps
           {
            script
              {
               file:name
               libraryname.functionname( )  <- pass arguments
              }
           }
        }
  }
```

of github

eg:- math.sum (a,b) → in functionname

Scripted pipeline:-

```
@Library('name of your shared library')_
node('built-in')
{
stage('name of the CI-CD stage')
  {
Library name.function name()
    }
}
```

→ Everything or any file should be created in vars file only

→ In vars file create a new file with some name any add code in that file.

Shared Library syntax / user defined Library

```
def gitdownload (repo)  → for cont download
{
  git "https://github.com/intelliqittraining/
       ${repo}.git
}

def build ( )  → for cont build
{
  sh 'mvn package'
}
def deployment ("jobname", "ip", "appname") → for Cont deploy
{
  sh "scp /var/lib/jenkins/workspace/
  ${jobname}/webapp/target/webapp.war
  ubuntu@${ip}:/var/lib/tomcat9/webapps/
    ${appname}.war
}
def gitdownload (repo) → for downloading test programs
{
  git "https://github.com/intelliqittraining/
       ${repo}.git
}
```

def selenium (jobname) → for executing shell of selenium program

sh "java -jar /var/lib/jenkins/workspace/${jobname}/testing.jar"

def deployment ("jobname", "ip", "appname") → cont delivery

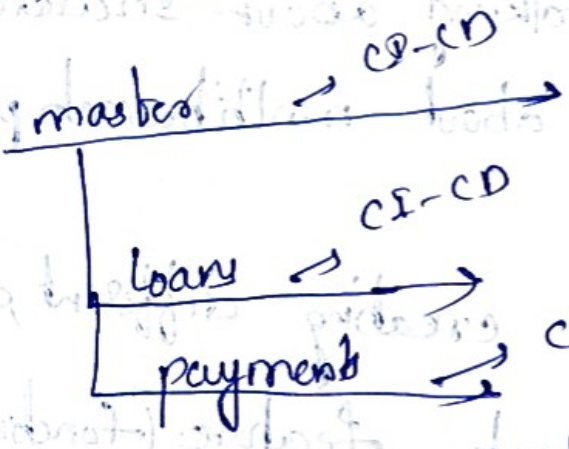
{

sh "scp /var/lib/jenkins/workspace/${jobname}/webapp/target/webapp.war ubuntu@${ip}:/var/lib/tomcat9/webapps/${appname}.war"

}

Note:- This will be created in git repository and function name is given in the pipeline script and the process is same for scripted/declarative pipeline.

Multibranch pipeline:-



→ In every branch we have to perform CI-CD.

→ Developer may create some code for loans and some code for payments

→ We have to perform the CI-CD in every branch, for that we have jenkins files related to that branch.

→ For loans functionality one Jenkins file and for payments one more Jenkins file and for master also one more Jenkins file

→ In some branches we have to perform stages upto decision of developer. depending on the code developed by the developer.

Interview

→ when we are talking about shared libraries we have talk about multibranch pipeline concept.

→ Developers are creating different, different branches for each feature (functionality) and developer is putting code related to that functionality. As a devops engineer I am designing separate Jenkins file for each and every branch, In that branch how many stages of CI-CD is there is decided by the developers. We talk to the developer and how many stages he is expecting, and we put those stages only in Jenkins file.

→ So Now developer is uploading the code in all the branches, each branch is having one, one Jenkins file,

→ Now I have configured the Jenkins file

in such a way that whatever stages whichever branches are modified and uploaded those corresponding Jenkins files will be automatically executed

→ This what we are calling as multibranch pipeline

→ Create Jenkins files for each branch

Vim Jenkinsfile

→ open it and paste the code of scripted pipeline or declarative pipeline

→ commit those files

→ Next on other branch also create jenkins file and commit them

→ Upload all them into git repository

→ To upload all the branches to git repository

```
git push -u origin --all
```

→ Now create a multibranch pipeline

→

Master - Slave - Architecture

→ The machine on which Jenkins is installed is called master and another machine where load/operation is distributed is called slave machine

Scenario:-

→ In an organisation we will be running multiple jobs parallely, it will take much time, and system becomes unresponsive and jobs get aborted.

→ To overcome this problem, one solution is create jenkins on high configuration server, and drawback is we have to pay much amount for whole month even

we are using or running jobs in a. 15 days or 20 days

→ To avoid that problem, master and slave came into picture.

→ Distribute the work or job to slave and run it, and after the job is done, delete that slave server

→ We can create as many slaves as we want.

→ Establish passwordless ssh connection to the master and slave

→ Go to the jenkins user and generate keys and copy into slave machine

→ The flavour of java installed on jenkins is, the same flavour of java is installed on slave too.

→ Now there is a slave.jar file on the master machine, that should be downloaded

into slave machine.

→ Connect to slave machine, there execute the command to download that slave.jar

```
wget http://ip_address of master:8080/jnlpJars/slave.jar
```
(or)

→ Now change the permissions to slave.jar

```
sudo chmod 777 slave.jar
```

→ Now create a folder on slave machine to run that job, it like workspace which is created automatically on Jenkins

→ `mkdir anyname`

eg: mkdir myfolder

→ Take that path of the myfolder and keep that path in jenkins

→ Now go to manage jenkins, in that you have option called as nodes click on that. Nodes are nothing but slave machines

→ Then you click on new node and name it as slave1 and select permanent agent and create

→ Number of executors, i.e no. of jobs you run parallely, give that number there.,

eg:- If you run two jobs give number as 2 and so on.

→ Remote root directory → in this give the path of your folder created on slave

eg. /home/ubuntu/myfolder

→ labels → give name as myslave

→ Usage → select the node as much as possible

→ launch method → Launch agent via execution of command on the controller

this option will pop up only if you install Command Agent Launcher plugin

→ Launch command

```
ssh ubuntu@ip-address(pvt) slave java
        -jar slave.jar
```

→ Save

→ launch Agent

→ Go to manage jenkins → In process Script approval

→ Now go to the jobs section, select

the job that have to run on slave

→ configure → Restrict where this project can be run ? → In that give the name of your slave and

save

→ Then run the job

Parameterized Pipeline

→ These parameterized jobs helps us to reuse the job by passing different arguments

→ This job takes inputs, this provides reusability to the job.

→ You can use the direct option as

☑ This project is parameterized ?

(or)

→ Write the script in pipeline

→ Parameters are (i) Boolean
(ii) Choice
(iii) Credentials
(iv) File → attach a file
(v) multi-line string Parameter
(vi) Password
(vii) Run
(viii) string

eg:- Use string parameter.

→ It is used to print the information

```
pipeline
{
 agent any
 parameters {
   string description: 'Enter first name', name: 'firstname'
   string description: 'Enter last name', name: 'lastName'
 }
 stages {
   stage("Welcome") {
     steps{
       - echo "Hi ${params.firstName}
       ${params.LastName}, Welcome"
     }
   }
 }
}
```

Using conditions in pipeline

→ These conditions can be implemented by "choice" parameter.

eg:- If we want deploy into dev or prod or staging

```
pipeline {
    agent any
    parameters {
        choice choices: ['dev', 'qa', 'stage', prod.]
            descriptn: 'choose the environment to
            deploy', name: 'appEnv'
    }
    stages {
        stage('Deploy to Dev') {
            when {
                expression {
                    params.appEnv == "dev"
                }
            steps {
                echo "Deploying to development"
            }
        stage('Deploy to Prod') {
            when {
                expression {
                    params.appEnv == "prod"
                }
```

```
    steps {
        echo "Deploying to prod"
    }
```

Jenkins environment variables

```
pipeline {
    agent any
    environment {
        TOMCAT_IP = "172.62.32.1"
        TOMCAT_USER = "ec2-user"
    }
    stages {
        stage("Deploy") {
            steps {
                echo "We are deploying to
                ${env.TOMCAT_IP}. using user
                ${env.TOMCAT_USER}"
            }
        }
    }
}
```

Jenkins post build actions

```
pipeline {
    agent any
    stages {
        stage("Hello") {
            steps {
                sh "sudo shutdown --reboot"
            }
        }
    }
    post {
        failure {
            echo "sending email to
                  development team..."
        }
        succes {
            echo "sending email to development
                  team... about successful deployment"
        }
    }
}
```