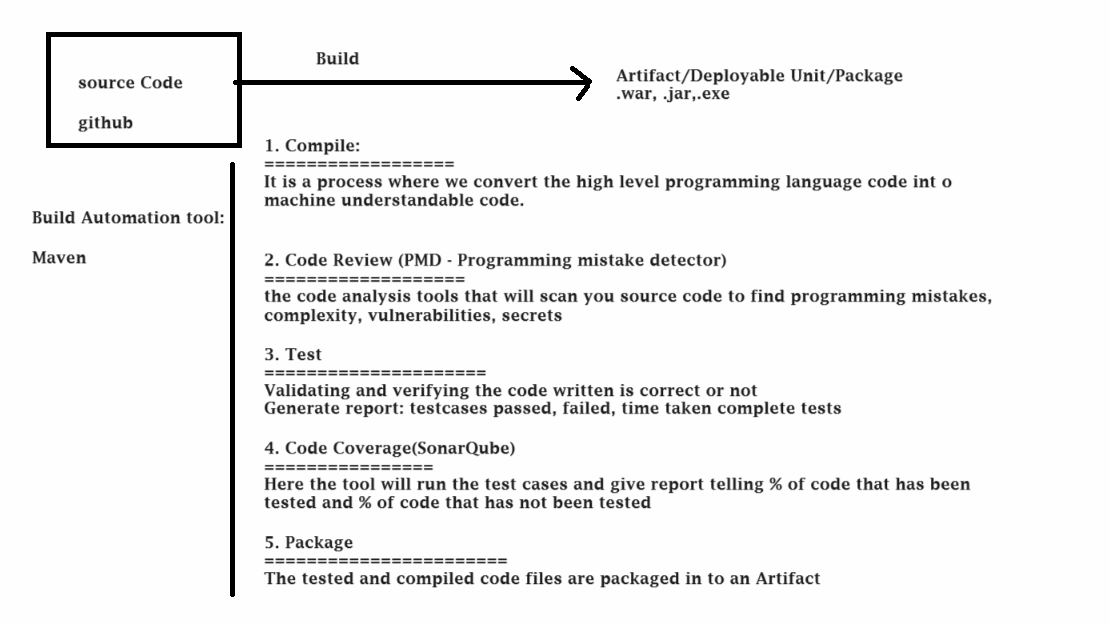
Jenkins Set up:

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



Create an ec2 Instance or use the same instance used in GIT sessions.

If git is not present:

Git installation:

yum install git -y

Check the version of GIT

…

git --version

…

SEE THAT YOU ARE IN ROOT DIRECTORY

# cd

# pwd

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JAVA Installation

SEE THAT YOU ARE IN ROOT DIRECTORY

…

**sudo amazon-linux-extras install java-openjdk11 -y**

…

IF Command to install java on CENTOS 8

**..**

**sudo yum install java-11-openjdk-devel**

**…**

Command to install java 11 on Ubuntu

**…**

**apt-get update**

**apt-get install openjdk-8-jdk**

**…**

Amazon linux 2023

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

Install Java

# sudo dnf update

# sudo dnf install java-11-amazon-corretto -y

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

Install Jenkins:

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

On the browser open https://www.jenkins.io/

Go to downloads section

Select centos OS

Go to Ec2 machine and execute commands:

**# sudo wget -O /etc/yum.repos.d/jenkins.repo** [**https://pkg.jenkins.io/redhat-stable/jenkins.repo**](https://pkg.jenkins.io/redhat-stable/jenkins.repo)

**# sudo rpm --import** [**https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key**](https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key)

**# yum install jenkins -y**

Execute command to start jenkins server:

**systemctl start jenkins**

systemctl status jenkins

Installation is complete on the server

Home Directory of jenkins: /var/lib/jenkins

Directory where Jenkins store its configuration and working:

# cat /etc/sysconfig/jenkins

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

We need to now set up jenkins dashboard

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

For this, take public ip address of ec2 server, copy it and go to your browser and give

publicipaddress:8080

On the ec2 server, execute below command

$ cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the password and paste in the browser (jenkins)

Click on continue

Click on Jenkins suggested plugin tab(on left side)

On the next page

Username: admin

Password : admin

Retype password: admin

Email: admin@gmail.com

Click on continue

Click on finish

You will be on the jenkins dashboard

Link for Jenkins Certification: https://www.cloudbees.com/cloudbees-university/training-certifications/jenkins

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create a Jenkisn job that will execute linux commands.

Create a new job in jenkins

Click on + sign to create new item/job/project

Give a name to the job : Job1

Select freestyle project and click on OK button

Go to build Steps→ select Execute Shell→ give commands like

touch file1

echo “hello Jenkins!”

Save the job → click on build Now button.

See the console output and check the workspace directory.

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

Day 2:

Demo 1: Create a Jenkisn job that will clone a github repository in jenkins workspace

Create a new job in jenkins

Click on + sign to create new item/job/project

Give a name to the job : CloneRepo

Select freestyle project and click on OK button

On the project click on Source code management

Select git option

Give git hub repo path

https://github.com/Sonal0409/myproject06nov.git

Branch name as ===> Master

Save the job

Click on Build now

Repository will be cloned in jenkins workspace

You can go to job → workspace folder to see the files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create a new job → general section→ click on advance → custom workspace

Give directory as /tmp/myworkspace

Save job → build

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Wipe out the workspace

On the job → go to build environments → Select first checkbox →Delete workspace before build

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jenkins Triggers.

To automatically execute a jenkins job, you will use Jenkins triggers

Based on:

Periodic time- schedule

Whenever there is a commit in github repo

A parent job can trigger execution of child job

Execute a job from remote script

You will go to build triggered section in the job to set up a trigger for automatic execution of the job

Trigger prieodically

Go to Job Clone repo --> build trigger--> build preodically --> give \*/2 \* \* \* \* --> Save and build now

All build will be generated automatically every 2 mins

Build Triggers--> Poll SCM --> \* \* \* \* \*

Build will be generated each time there is a chnage in the repository only.

Webhook:

Webhooks allow Jenkins to be notified when certain events happen on the repo. When the specified events happen, we’ll send a POST request to each of the URLs you provide.

GitHub hook trigger for GITScm polling

> Select the above option

Save the job and now go to git hub

Always save the job and then only go to git hub, perform these steps

go to the repository setting

select webhooks on left side

delete if any exisitng webhook

click on createwebhook on right side

provide following information:

Payload URL : jenkinsurl/github-webhook/ ===> http://3.140.252.165:8080/github-webhook/

Content type : select application/json

Secret: no need of any value

Which events would you like to trigger this webhook?

select first option -- Just the push event.

Select Active

click on add webhook.

now make some changes in repo,

go to jenkins--> you will see a new build has been created

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

Day 3: Build pipeline using Jenkins, Maven & GIT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Build too : Maven

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

> Powerful project management tool, This tool will help a developer to write his source code and test scripts and maintain them locally

> maven at heart is a plugin based tool (plugin execution framework)

> It is used for Java based code

> All the work in maven is done by plugins (extensions, which when downloaded and executed will enhance the features of actual tool)

> Compile the code → Maven plugin

> 2 types of plugin in Maven:

> Core Build plugins

Tasks Plugin Command (GOAL)

Compile the code Compiler compile

Execute Test cases Surefire test

Package the code package package

Delete previous

build files Clean clean

Check dependencies install install

Deploy the package to

Artifactory deploy deploy

> Reporting Plugins

Plugins that are needed for validation and review the code and generate reports

Reports by maven plugins are generated in the format : .xml, .csv, .html or .txt

Tasks Plugin Command(GOAL)

Code review

(Static Analysis tool) pmd pmd:pmd

Test reports surefire-reporting test

Code Coverage cobertura(deprecated)

JACOCO

Code review

& Code Coverage Sonarqube sonar:sonar

Code Review with PMD: The PMD Plugin allows you to automatically run the [PMD](https://pmd.github.io/) code analysis tool on your project's source code and generate a site report with its results.

Code Coverage : Using code coverage tool against your compiled classes it helps you determine how well the unit testing and integration testing efforts have been, and it can then be used to identify which parts of your Java program are lacking test coverage.

A developer will create a Maven project locally in his/her IDE and write source code and test cases inside it

Traditionally, they will build all the above steps in their local machine.

Every project will include following folders:

/src/main/java ⇒ main java source code

/src/test/java ⇒ test cases will be present

In a maven project, you will have the main POM.xml

Which created by Java developer

This file will be prepared before you write the code

This file will be updated while the code

In this file Java developer will write :

> name of the artifact which will be build

> type of artifact

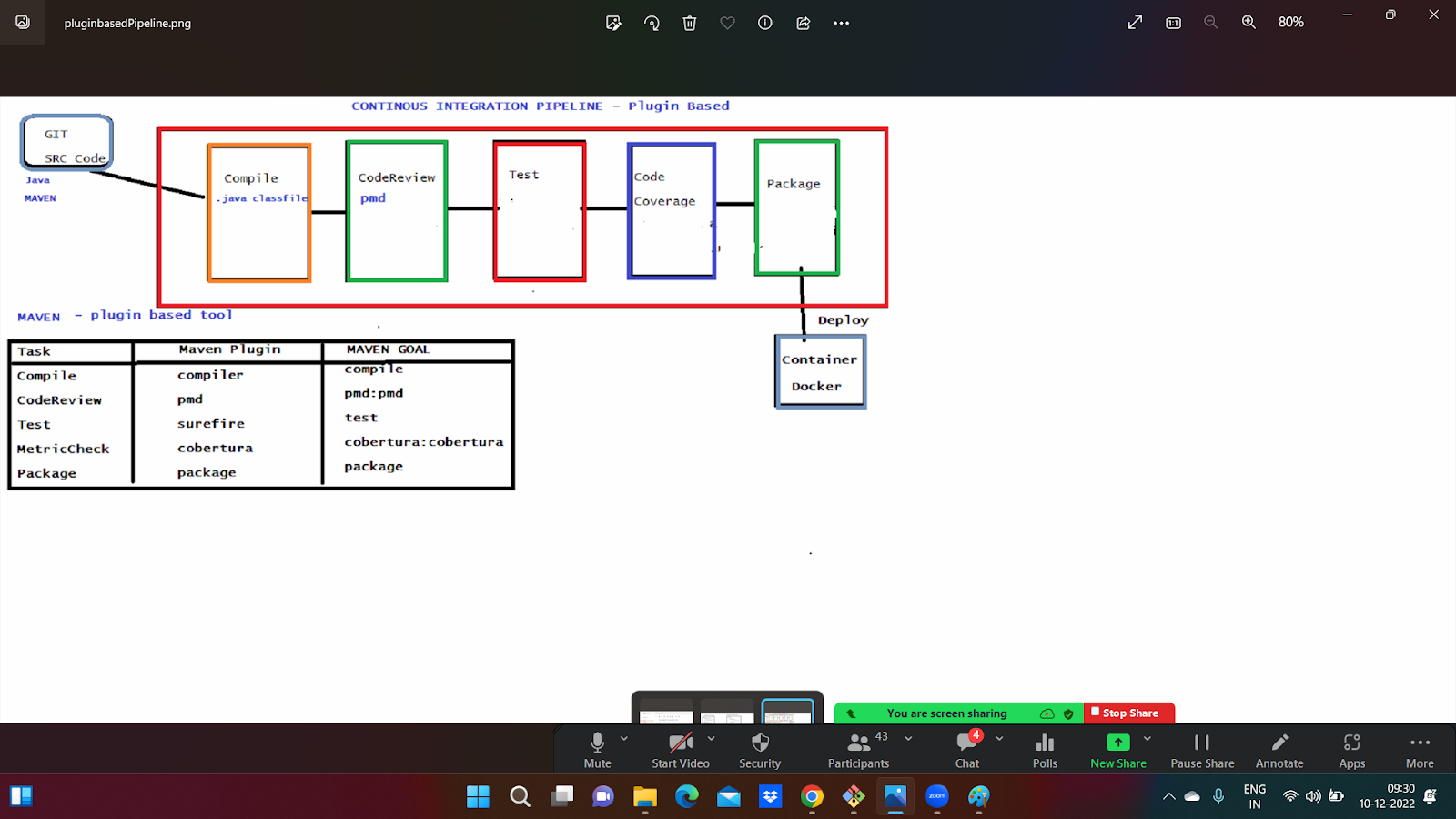
> Dependencies ⇒ softwares that will download automatically for code to work

\*\*\*\*

**Build section:**

**> name and version of plugins**

**Which all plugin we will use for build process**



SetUp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jenkins--> Manage Jenkins-->Global Tool configuration

Under GIT==> leave it same

Under maven ==> let's install it automatically

Type name as mymaven

check the install automatically box.

Save the changes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CREATION OF COMPILE JOB

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. new Job --1.complie-->freestyle project

2. Source code management --> select git

--->give git path ===> https://github.com/Sonal0409/DevOpsCodeDemo.git

3. build ==> select invoke top level maven targets

select maven version===> mymaven

goal ==> compile

4. save ==> build now

So all the compiled files will be present at this location

==> look for this line in console output at the end on jenkins

Compiling 13 source files to /var/lib/jenkins/workspace/compile1/target/classes

5. Go to workspace on the left side of jenkins Job compile

In the folder go to ==> target folder==> go to classes/com/edurekademo

==> click on edurekademo==> got to utilities==> all class files will be there which have recently been compiled.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2nd JOB ==>Code Review

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Jenkins--> new item--> Name= code Review==> freestyle project

==> source codemanagement==>select git==>

give git repo name https://github.com/Sonal0409/DevOpsClassCodes

Step 2: build ==> invoke top level maven target==>

select maven version==>mymaven ==> goal = pmd:pmd

save==> build now

==> click on build number and see the console

Goto Workspace on the left side of jenkins Job code review

In the folder go to ==> target folder ==> you will find pmd.xml file.

CONVERT FILE TO TREND REPORT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

manage jenkins--> manage plugins--> available --> Search for warning next gen plugin--> install it.

Retsrat in case plugin fails installation.

Now go to the job Code Review==> go to post build actions ==> select recordcomplier warnings and static result analysis

Under tool ==> select PMD

under report file format ==> give path of pmd.xml file ie: target/pmd.xml

or as mentioned in message below : copy \*\*/pmd.xml ( no quotes)

Save the file and build now.

After the build is successfull, you will see PMD Warnings

So you will see 12 new warning have been generated on the code.

Scroll down

Under package click on first one to check warnings.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JOB3: Testing --Unit Test report

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. jenkins--> new item--> Name= 3.unitTest==> freestyle project

==> source codemanagement==>select git==> give git repo name

<https://github.com/Sonal0409/DevOpsCodeDemo.git>

2. build ==> invoke top level maven target==> select maven version==>mymaven ==> goal = test

3. Save and Build now.

4. Check the workspace

5. surefire-reports will be present

But we cannot understand them easily. So lets generate understandable reports by using Junit reports option under post build actions

6. Go back to job==> post build actions==> select junit test result job

==> give target/surefire-reports/\*.xml ===>under test report xml

This is path where xml files are there.

Here \*.xml is as we want to use all the xml files

7. Save and build again

Go to build number ===> on left side you will see Test Result

Click on it and you can see all pass and fail details of the tests

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Job4 Package Job

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Go to jenkins--> new item--> Name= Package==> freestyle project

==> source codemanagement==>select git==> give git repo name https://github.com/Sonal0409/DevOpsClassCodes

build ==> invoke top level maven target==>mymaven

goal = package

==> save==> build now

==> click on build number and see the console

==> go to workspace ==> target/addressbook.war

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Connecting all the 4 jobs so that each job can be triggered:

Go to manage plugins → available section

Search for build pipeline plugin

Click on the checkbox and click on Install without restart

Plugin will be installed. It will display all connected jobs on the dashboard

This plugin falls in the category of Jenkins view. It creates a build pipeline view

Lets build the pipeline:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Day4:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Pipeline as a code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Lets create a pipeline using code:

Create a new job ==> select project as pipeline

Build pipeline with parameters:

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

pipeline{

// Any available server, currently master node is the available server

agent any

parameters{

// a static varibale with a static value

string(name: 'Tool', defaultValue: 'Jenkins', description: 'Which tool are you learning')

string(name: 'username', defaultValue: 'admin', description: 'What is the username of Jenkins')

}

stages{

stage('Print Tool\_NAME')

{

steps{

echo "Hello to the tool - ${params.Tool}"

}

}

stage('Print User\_name')

{

steps{

echo "The user name is ${params.username}"

}

}

}

}

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

Example 2: Pipeline with script and if condition

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

pipeline{

agent any

parameters{

string(name: 'branch' , defaultValue: 'test', description: 'Enter the branch Name')

}

stages{

stage('Build on Dev'){

steps{

// if we ahve to give conditions, we use a sectionc alled as script

script{

if(params.branch == 'dev')

{

echo "Do dev Activites"

}

}

}

}

stage('Build on Test'){

steps{

script{

if(params.branch == 'test')

{

echo "Do test Activites"

}

}

}

}

}

}

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

Pipeline 3 - When keyword

pipeline{

agent any

parameters{

choice(name: "ENV", choices: ["","TEST","PRE-PROD","PROD"])

}

stages{

stage('Deploy on TEST')

{

when {

expression {params.ENV == "TEST"}

}

steps{

echo "deploy in Test env"

}

}

stage('Deploy on PRE-PROD')

{

when {

expression {params.ENV == "PRE-PROD"}

}

steps{

echo "deploy in pre-prod env"

}

}

stage('Deploy on PROD')

{

when {

expression {params.ENV == "PROD"}

}

steps{

echo "deploy in prod env"

}

}

}

}

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

Continuous Integration pipeline

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

pipeline{

tools{

// what tool version to use for build stages

maven 'mymaven'

}

agent any

stages{

stage ('CloneRepo')

{

steps{

echo 'This is stage 1'

git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'

}

}

stage ('Compile')

{

steps{

sh 'mvn compile'

}

}

stage ('CodeReview')

{

steps{

sh 'mvn pmd:pmd'

}

post{

success{

recordIssues(tools: [pmdParser(pattern: '\*\*/pmd.xml')])

}

}

}

stage ('Test')

{

steps{

sh 'mvn test'

}

post{

success{

junit 'target/surefire-reports/\*.xml'

}

}

}

stage('package')

{

steps{

sh 'mvn package'

}

}

}

}

> Jenkinsfile

1. It is a simple text file with no extension
2. This file contains declarative pipeline code for CI/CD pipeline
3. The name of this file is always Jenkinsfile
4. This will is maintained in the root directory of your project repo on SCM tool
5. Every team member can now make changes to the pipeline code without logging on the jenkins server.
6. In the github we can also maintain every version of pipeline code
7. We write the pipeline code in IDE and push the Jenkinsfile in the repository
8. In jenkins you will create a job, that will fetch the Jenkinsfile and jenkins will execute the pipeline for you.

VSCODE plugins for write/validate/execute jenkinsfile

<https://www.jenkins.io/blog/2018/11/07/Validate-Jenkinsfile/#validate-your-jenkinsfile-from-within-vs-code>

<https://marketplace.visualstudio.com/items?itemName=tabeyti.jenkins-jack>

Demo:

Click on new Item → give the job name as JenkinsfileDemo → Select pipeline project→ press OK

In the job template → click on pipeline tab → select the dropdown option as Pipeline script from SCM

Select SCM as → git

Repository URL[?](http://18.223.182.58:8080/job/JenkinsFileDemo/configure#) ⇒ <https://github.com/Sonal0409/DevOpsCodeDemo.git>

We will also add a trigger → Go to Build triggers tab above→ POLL SCM → \* \* \* \* \*

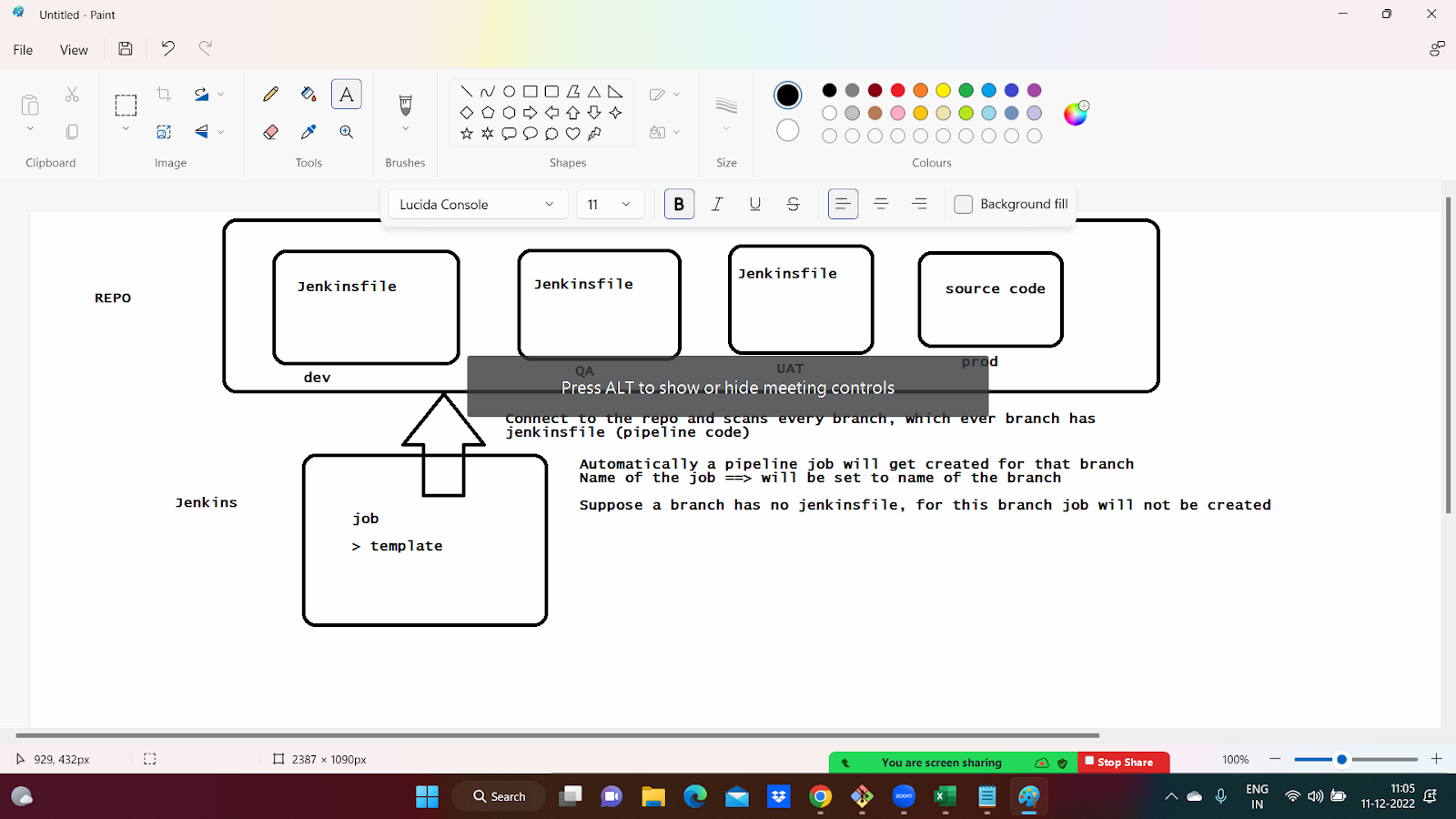
So that if there is any commit, my pipeline should get executed everytime.

> MultiBranch Pipeline

1. Basically we have many branches in github for a repository
2. Each branch may have separate set of tasks/jobs to be executed
3. Whenever a branch is created on the repo, Make sure the branch has a Jenkinsfile init
4. In the Jenkinsfile you write the code which is needed to execute certain tasks related to the branch
5. we will set up a job in jenkins **that will scan every branch of your github repository**
6. **Whichever branch has the jenkinsfile present on it**

> for that branch a jenkins pipeline job will be created with same name as branch name and Jenkinsfile code will be executed

1. Whichever branch of your repo does not have Jenkinsfile on it. No job will be created for it.



Demo:

Click on new item→ create job -> give name as MultibranchdemoJob → select the template as **Multibranch Pipeline → press Ok**

**In General section → give Display name as : Multibranch Pipeline**

**In branch Sources section → Click on Add source → select git**

**Give git Project Repository ->** [**https://github.com/Sonal0409/MultiBranchDemo.git**](https://github.com/Sonal0409/MultiBranchDemo.git)

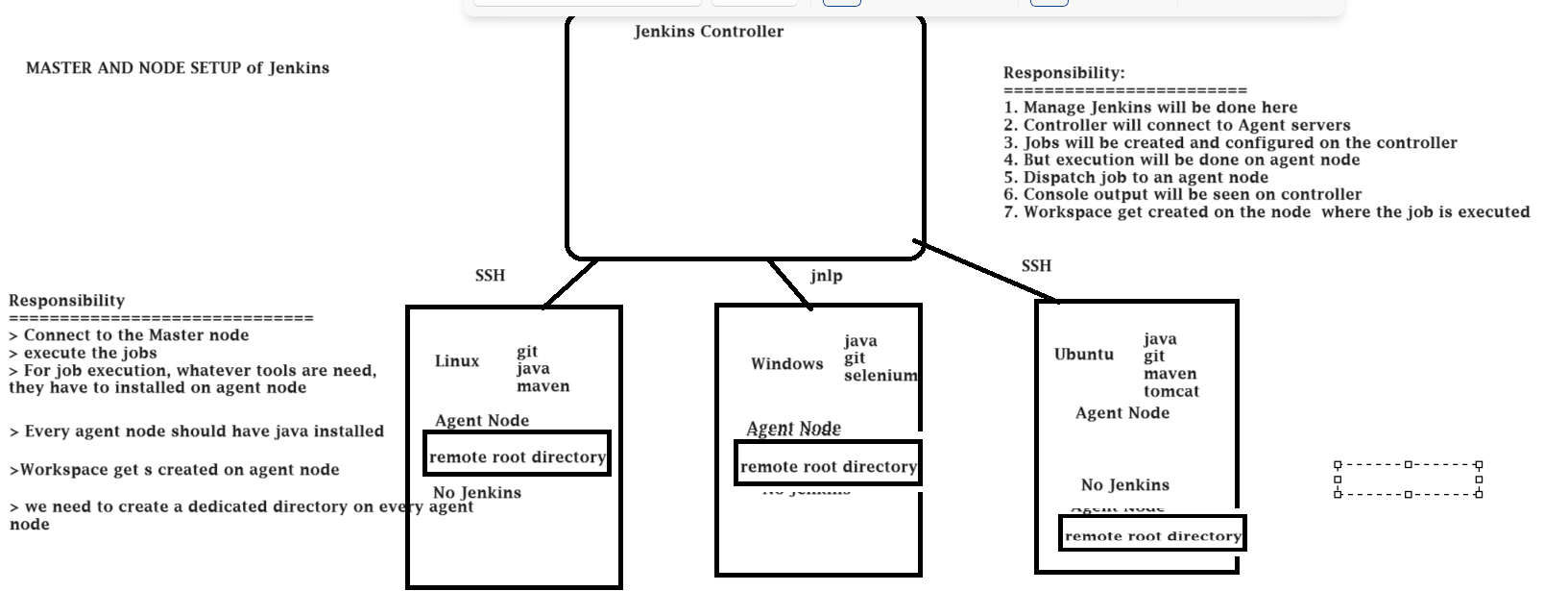
**Nomore changes and save the job.**

**Jenkins will automatically scan the branches of the repo and create pipeline jobs.**

**Click on multibranch pipeline job name and you will see the 3 jobs created for 3 branches.**

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

1. Master and node architecture in Jenkins



1. Create a linux slave and dispatch job from master to slave

Create a new EC2 instance with OS as Amazon linux 2

Provide the security group rule 2 also.

Connect to the instance and install following tools:

Execute command to become root

sudo su -

Java:

sudo amazon-linux-extras install java-openjdk11 -y

GIT:

yum install git -y

**Create a root directory**

This directory will be used exclusively my Jenkisn master to place remote files and create job workspace

You need to create a directory and give read/write/execute access to it

Execute:

$ cd /tmp

$ mkdir jenkinsdir

The path of root directory will be /tmp/jenkinsdir

give read/write/execute access to the root directory

$ chmod -R 777 /tmp/jenkinsdir

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Setup on Jenkins master:

Go to manage jenkins → Configure system → Scroll down to jenkins location and change the Jenkisn URL to be the latest URL

http://3.147.195.251:8080/

Save the page

2. Manage jenkins → configure global security→ scroll down → Agents → select Random radio button

Save the page

Manage Jenkins – Manage nodes and Clouds

Click on + New node

Give node name: slave1

Select Permanent Agent

Click on create

Now give description : anything text

Number of executors: 1

Remote root directory: /tmp/jenkinsdir

Labels : linux\_slave

Usage : select second option from dropdown

Launch method: launch agents via SSH

Host: private IP of agent node

Credentials: click on Add button > click on jenkins

Select Kind : SSH username and private key

username: ec2-user

Private key: you will get the private key of the agent node in your local laptop> downloads folder

got the downloads folder of laptop, search for the key(.pemfile)

open the file with notepad and copy the entire text

paste the text in jenkins > under private key

Now select the credials from the drop down.

step 7: Host Key Verification Strategy → select the 4th option->Non verifying Verification Strategy

Save the node details

On the dashboard you can see the node is now connected to master

Reference material:

https://wiki.jenkins.io/display/JENKINS/Distributed+builds#Distributedbuilds-Schedulingstrategy

**Windows Agent:**

**Download git on windows server:**

[**https://git-scm.com/download/win**](https://git-scm.com/download/win)

**Download java 11 on windows server**

[**https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html**](https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html)

restart the instance.

Restart the jenkins server

now take public ip and loginto jenkins

Step 1: Go to Manage Jenkins ---> CONFIGURE SYSTEM ---> Check the Jenkins URL is same as URL of browser --> Save

Step 2: Go to Manage Jenkins---> CONFIGURE GLOBAL SECURITY --> Scroll down to AGENTS-->select RANDOM radiobutton--> Save

Step 3: Go to Manage Jenkins---> MAnage Nodes and Clouds --> New Node

Select New Node--> give name as Winslave--> select radio button permanamet Agent

Name : winslave

Description: give windows 10 machine

==> # of executors = 1 ==> means number of jobs that we want to run on this slave node. Let it be 1 for now

===> Remote root directory ==> C:\jenkinsdir

Location of a directory folder on your windows machine , so that jenkins can create a workspace folder and store all the files. As we have seen earlier while executiong jobs.

For this --> got to C drive and create an emapty folder.

C:\jenkinsdir

====> Label : win\_slave ==> this is like a tag to slave machine. This is how we will identify the slave machine--> remeber this

===> Usage:

Select 2nd option: Only build jobs with label expressionmatching the job :

This means for which ever job we will give label as win\_slave that job only will run on this job.

===> Launch Method : select option : Launch agent by connecting to master.

Scroll down

Add tools

Add git path as C:\Program Files\Git\cmd\git.exe

SAVE the job.

Now go to win\_slave ==> download the agent.jar file and put it in your jenkinsdir folder.

Copy the path and change the public ip in below path

java -jar agent.jar -jnlpUrl http://3.139.60.71:8080/computer/winslave/slave-agent.jnlp -secret 61e8e20badd531b48caeffac8d686f182eb2d77a1c303ee39c5db9d00117c0be -workDir "C:\jenkinsdir"

Go to bash and start jenkins, refresh instance

Go to cmd prompt

change to cd c:\jenkinsdir

copy the above jenkins agent path

System will get connected.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RUN a job on Agent

Go to Dashboard

Select the package job---> configure --> GENERAL --> click on Restrict where this project can be run and enter win\_slave

SAVE THE JOB

CLICK on BUILD NOW -- to start the build

CLick on build number--> got to console --to check it is executing on AGent

Go to C folder-->jekinsdir---> C:\jenkinsdir\workspace\package\target==>

you will see addressbook.war file.

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

Continuous Deployment

Tomcat server- Installation on EC2 (2 VM, other than the jenkins machine)

Deployment of war file on tomcat server using freestyle project

> Jenkins plugin : deploy on container

Deployment of war file on tomcat server using pipeline project

> Jenkins plugin: SSH Agent

Quiz for your end exam

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Deployment Process (for your information)

<https://tomcat.apache.org/tomcat-7.0-doc/appdev/sample/>

The easiest way to run this application is simply to move the war file to your **CATALINA\_HOME/webapps** directory. Tomcat will automatically expand and deploy the application for you. You can view it with the following URL (assuming that you're running tomcat on port 8080 as is the default):

Public ip[:8080/sample](http://localhost:8080/sample)

**Set for tomcat**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Tomcat installation on EC2 instance**

Pre-requisites

EC2 instance with Java 11

**sudo amazon-linux-extras install java-openjdk11 -y**

You can execute the command : [optional]

# alternatives --config java

To check java version, it should be 11

**Install tomcat:**

# Create tomcat directory

Go to the EC2 machine - t2 micro

**cd /opt**

# **wget https://dlcdn.apache.org/tomcat/tomcat-8/v8.5.93/bin/apache-tomcat-8.5.93.tar.gz**

# ls ⇒ you will see the tar file

# tar -xvzf **apache-tomcat-8.5.93.tar.gz**

Rename the directory

# mv apache-tomcat-8.5.93 tomcat

# cd tomcat

# cd bin

# ./startup.sh ==> tomcat will be up and running

Check point :

access tomcat application from browser on port 8080

<Public\_IP>:8080

Lets edit the context.xml file so as to access manager app

# cd ..

# vim /opt/tomcat/webapps/host-manager/META-INF/context.xml

comment this line in the contxt.xml

<Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />

How to comment : : use this symbols:

**<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"**

**allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->**

**# vim /opt/tomcat/webapps/manager/META-INF/context.xml**

comment this line in the contxt.xml

<Valve className="org.apache.catalina.valves.RemoteAddrValve"

allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />

How: use this symbols:

**<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve"**

**allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->**

**Save the files**

**Restart tomcat services**

**# cd bin**

**# ./shutdown.sh**

**# ./startup.sh**

**Create roles and users**

**# cd ..**

**#**

**$ cd conf**

**$ vim tomcat-users.xml**

**Scroll till bottom and add**

**at the end before </tomcat-users> add these roles**

**<role rolename="manager-gui"/>**

**<role rolename="manager-script"/>**

**<role rolename="manager-jmx"/>**

**<role rolename="manager-status"/>**

**<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>**

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

**<user username="tomcat" password="s3cret" roles="manager-gui"/>**

**Save the file**

**Make sure there is no space before each line in the file**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Restart tomcat**

**# cd ..**

**$ cd bin**

**$ ./shutdown.sh**

**$ ./startup.sh**

**publicip:8080**

**click on managerapp**

**Username: tomcat**

**Password: s3cret**

**This completed tomcat installation**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Go to jenkins server and start the jenkins dashboard**

**Download a plugin: Deploy to container**

**Create a free style job**

**Give name to the job**

**Select freestyle project**

**Press okay**

**give the repo name:** [**https://github.com/Sonal0409/DevOpsCodeDemo.git**](https://github.com/Sonal0409/DevOpsCodeDemo.git)

**Build steps : select Invoke top level maven targets**

**Maven Version : mymaven**

**Goal: clean package**

**Post build actions**

**select deploy war/ear on container**

**WAR/EAR files : \*\*/\*.war**

**Containers:**

**click on add container--> select Tomcat 8.x Remote**

**Add credentials. Click on ADD button→ jenkins→**

**Username: deployer , password = deployer**

**Id = tomcat, description = tomcat**

**Click on ADD button to save the credentials**

**In credentials dropdown, select the credentials**

**Tomcat URL**

[**http://publicip:8080/**](http://publicip:8080/)

**Save the job and execute it**

**You can access the application from**

**http://publicip:8080/addressbook/**

**Scenario 2:**

**Deploy code on tomcat using Pipeline as code**

**Algorithm of stages in the pipeline:**

**Stage 1 -> clone the repo**

**Stage2 -> Package the code**

**Stage3 -> Deployment of war file**

**> jenkins will connect to tomcat server over ssh**

**> using SSH command(SCP) we will copy the war file from jenkins workspace to the webapps directory of tomcat server**

**Step 1: Start the tomcat webserver**

**# cd /opt**

**# cd tomcat**

**# cd bin**

**# ./startup.sh**

**Step 2: In jenkins – Download a plugin : ssh agent**

**Install it.**

**Step 3: Create a pipeline job**

**Stage 1 -> clone the repo**

**Stage2 -> Package the code**

**Pipeline code for above 2 stages is :**

**pipeline{**

**tools{**

**maven 'mymaven'**

**}**

**agent any**

**stages{**

**stage ('Clone a repo')**

**{**

**steps {**

**git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'**

**}**

**}**

**stage ('Package the code')**

**{**

**steps{**

**sh 'mvn package'**

**}**

**}**

**}**

**}**

**Now build the job**

**Go to console output of build.. Scroll down and get the path of workspace where war file is present.**

**For me it is:**

**/var/lib/jenkins/workspace/CI\_CDPipeline/target/addressbook.war**

**Step 4:**

**In the pipeline click on pipeline syntax**

**Select from the drop down → SSH Agent**

**Select the credentials by clicking on ADD button**

**Select jenkins**

**Select SSh with username with private key**

**Give id : tomcatid2**

**Give description as : tomcatid2**

**Give username as : ec2-user**

**Under Private Key**

**Click on enter directly**

**Click on Add button**

**Go to AWS instance→ scroll right to see the pem file nae**

**Go to download folder , get your pem file→ open it with notepad.. Copy all content and paste it in jenkins under key**

**Clcik on ADD**

**Under SSH agent in the drop down select your usercredentials**

**Click on generate pipeline script**

**Copy the script and we will add it in pipeline stage**

**stage('Deploy the code on tomcat server')**

**{**

**steps{**

**sshagent(['tomcatid1']) {**

**// script to copy war file on webapps directory**

**}**

**}**

**}**

**Using SSH we will copy the addressbook.war file in the tomcat webapps directory**

**The SSh command to copy the .war file is:**

**command: scp -o StrictHostKeyChecking=no filenamelocation username@publicip:Destinationpath**

**For me it is :**

**sh 'scp -o StrictHostKeyChecking=no /var/lib/jenkins/workspace/PipelineCodeDeployment/target/addressbook.war ec2-user@18.116.238.106:/opt/tomcat/webapps'**

**Save the JOB**

**\*\*\*\*\* Also an important step is :**

**Go to tomcat server and give permissions to ec2-user to make changes to webapps directory**

**Just execute this command on tomcat server:**

**# chown -R ec2-user:ec2-user /opt**

**Now build the job.**

**Complete script for CI Cd is:**

**pipeline{**

**tools{**

**maven 'mymaven'**

**}**

**agent any**

**stages{**

**stage ('Clone a repo')**

**{**

**steps {**

**git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'**

**}**

**}**

**stage ('Package the code')**

**{**

**steps{**

**sh 'mvn package'**

**}**

**}**

**stage('Deploy the code on tomcat server')**

**{**

**steps{**

**sshagent(['tomcatid1']) {**

**sh 'scp -o StrictHostKeyChecking=no /var/lib/jenkins/workspace/PipelineCodeDeployment/target/addressbook.war ec2-user@18.116.238.106:/opt/tomcat/webapps'**

**}**

**}**

**}**

**}**

**}**