**Jenkins-Task-04**

**1) Setup jenkins CICD pipeline using freestyle job using Docker containers using below code.**

https://github.com/betawins/hiring-app.git

Stages:

1. Git Clone

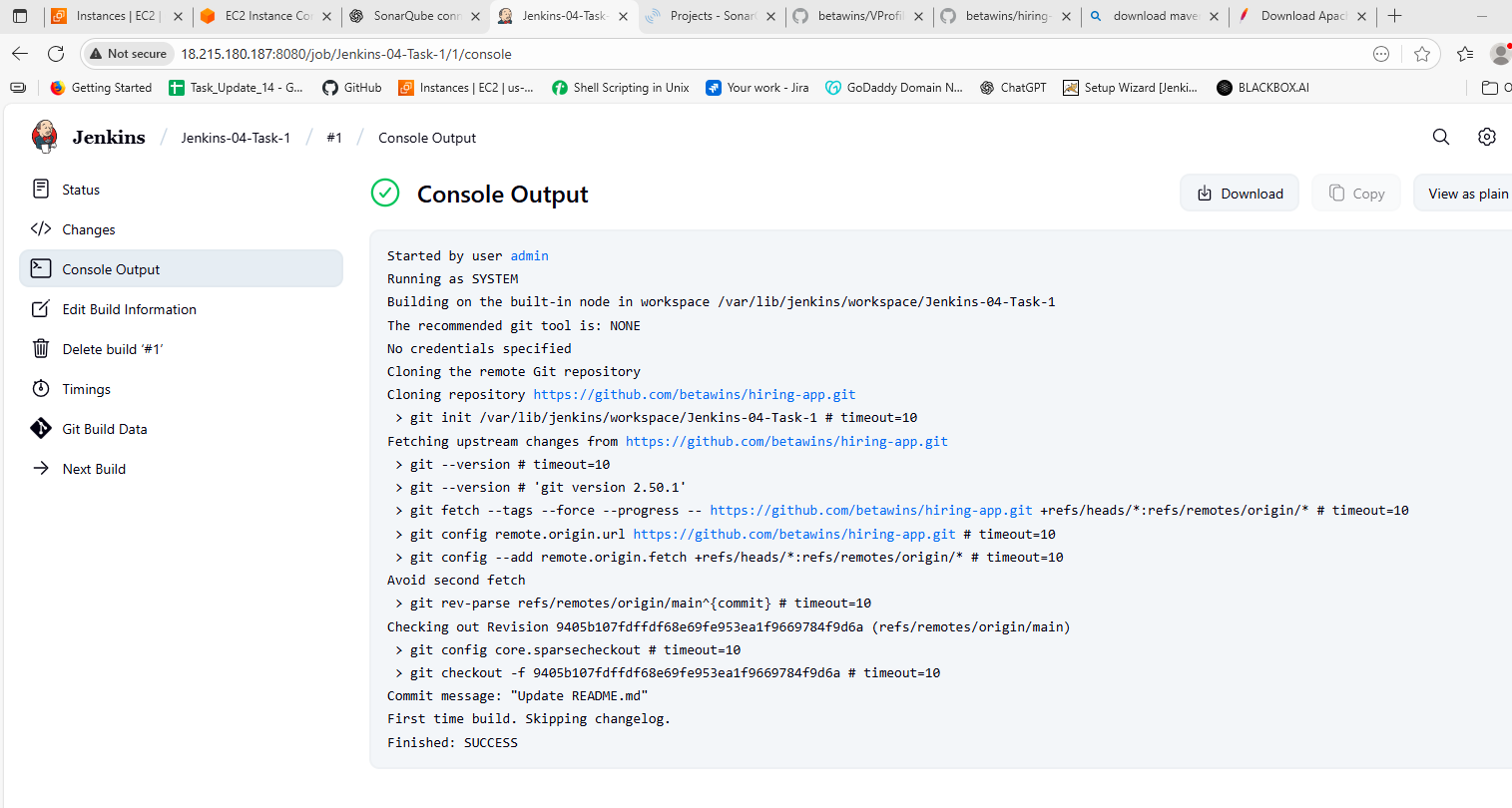
\* First I have created the job name : Jenkins-04-task-1



\* Then attach the git repo url from the given link as I have attach in above pic.

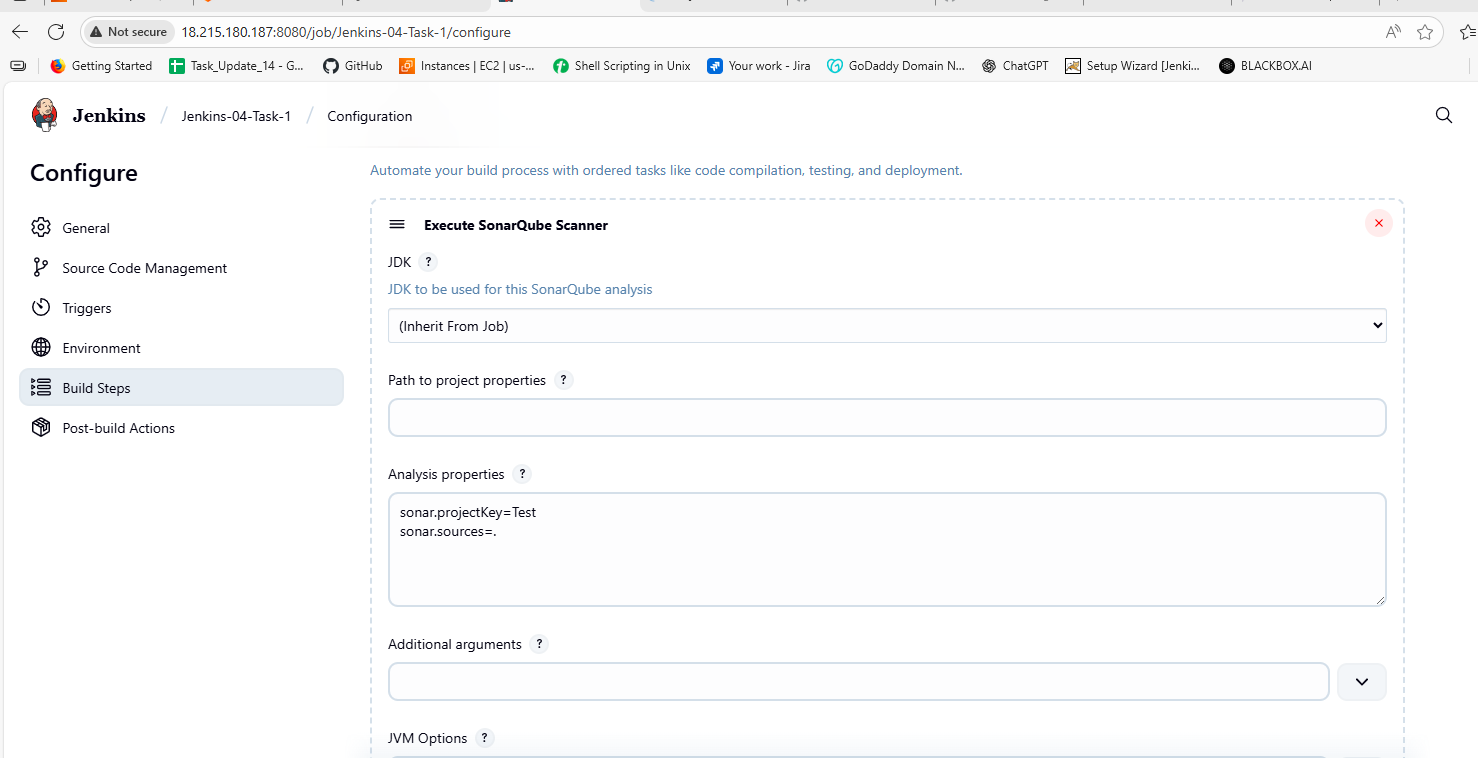
\* Click on save.

\* Then **Build** the job



Now we have done with the first stage.

1. Sonarqube Integration

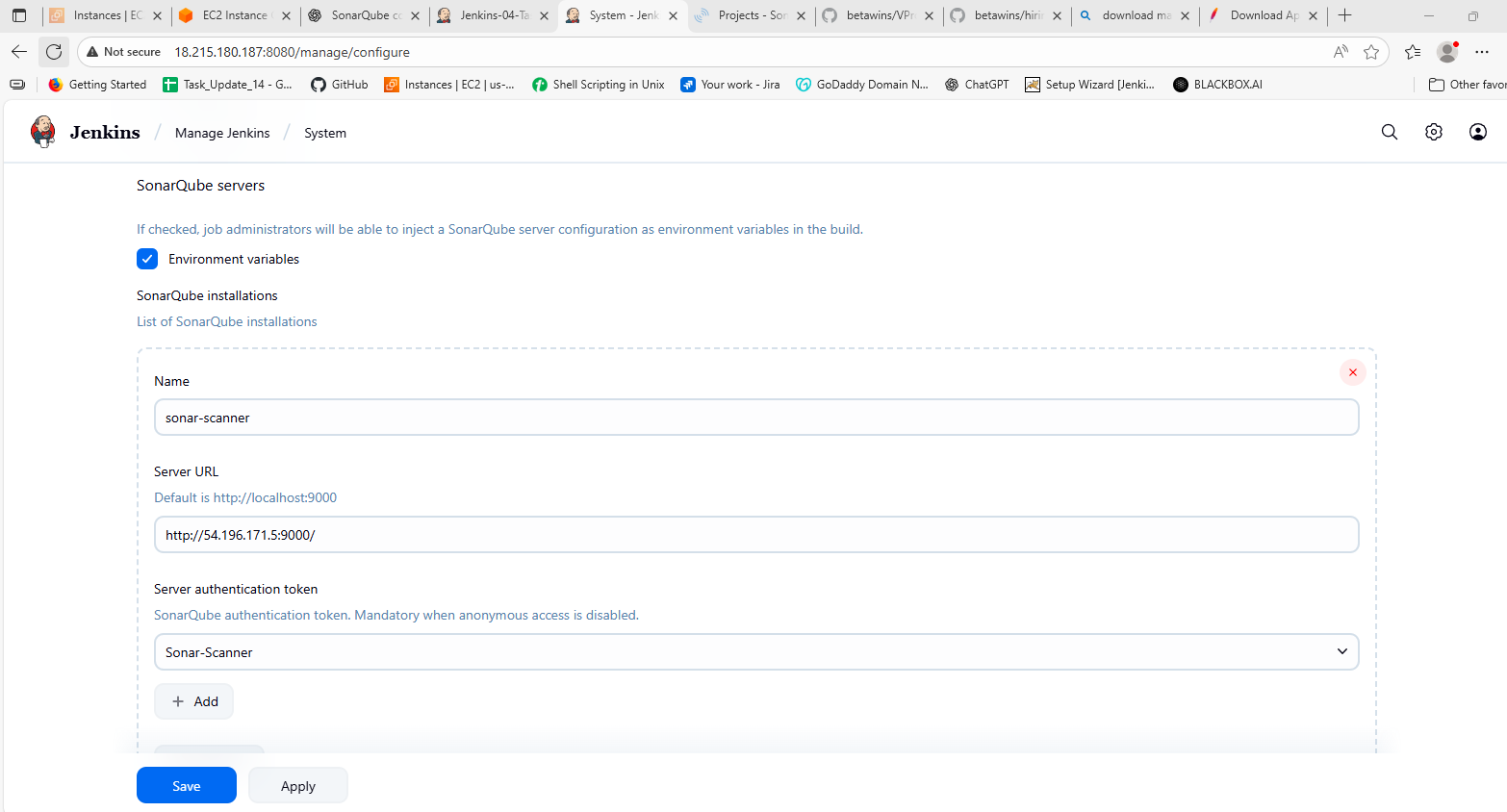


Here first we have to go to the **job** configuration and go to **Execute Sonarqube** **Scanner** then attach the **Analysis properties** in that which will be available in th given repo link at the end. As I have done in the above pic.

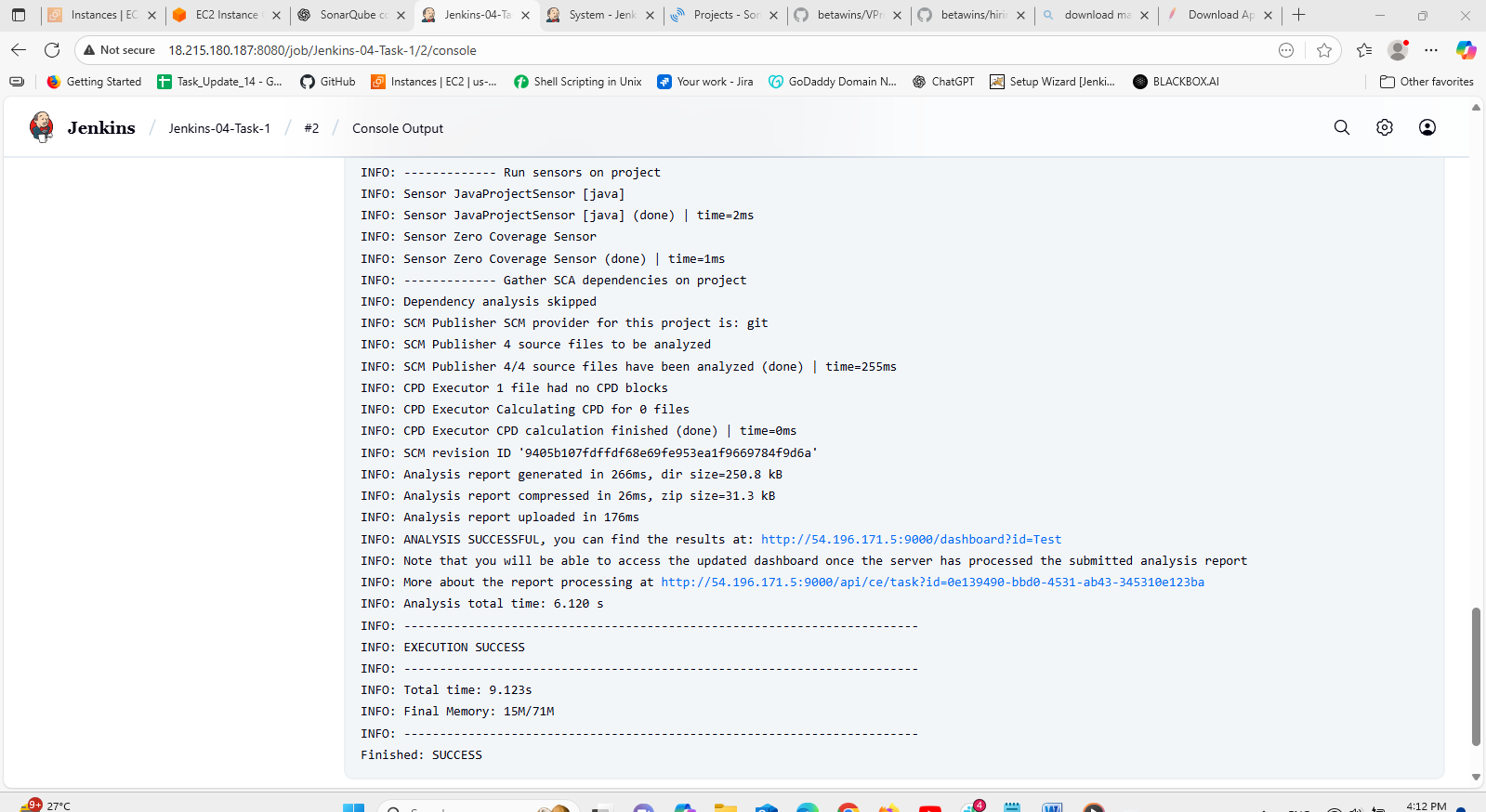
Before this we have to do some configurations in the Settings like

Go to Manage then System then in Sonarqube installation

* Name we have to give which is in the local machine
* Then paste thr server url of with the port number of sonarqube.
* Then we are attaching the Authentication Token
* This token we can create in the Sonarqube dashboard. Go to Account in left side.
* Then go to security click on generate token. You will get a token
* Then you have to paste that in the Jenkins Credentials.

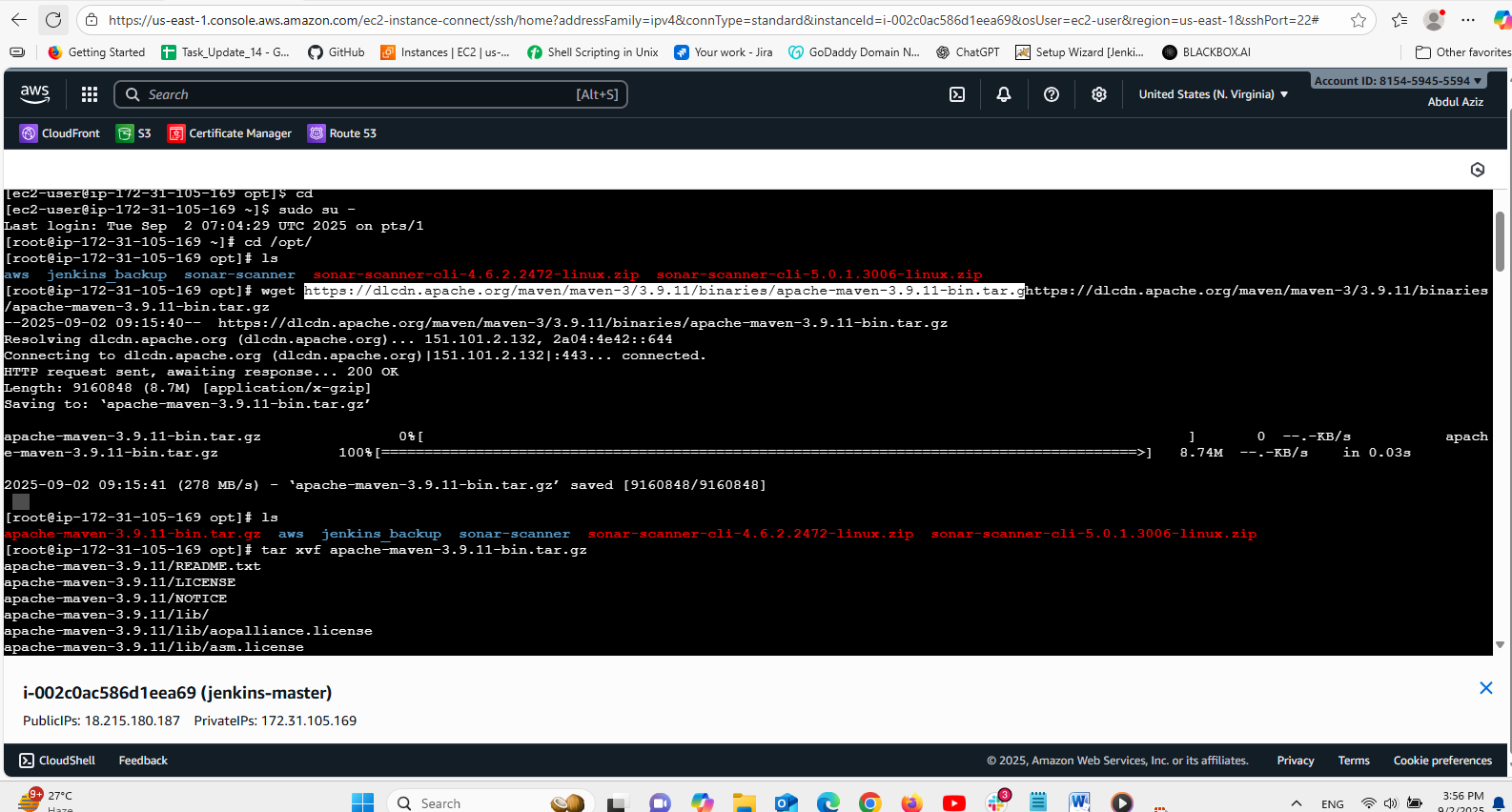


After adding everything just build the job and you should get the output like thiss



1. Maven Compilation

Here we have to install the Maven in the local machine first.

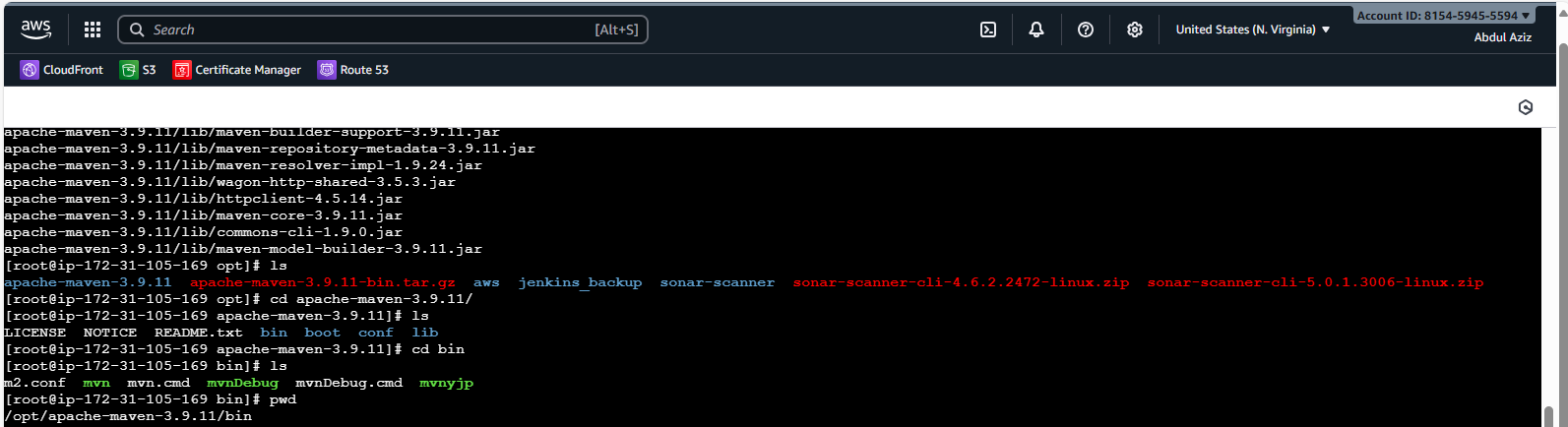


In the above picture we can see the installation

\* First we should get into the opt then copy the link from the browser then paste it with **wget** command then the tar file will be created.

\* extract that tar file

tar xvf “file-name”



After that we have to go to the **.bash\_profile** file then add some pathin this like

# Get the aliases and functions

if [ -f ~/.bashrc ]; then

. ~/.bashrc

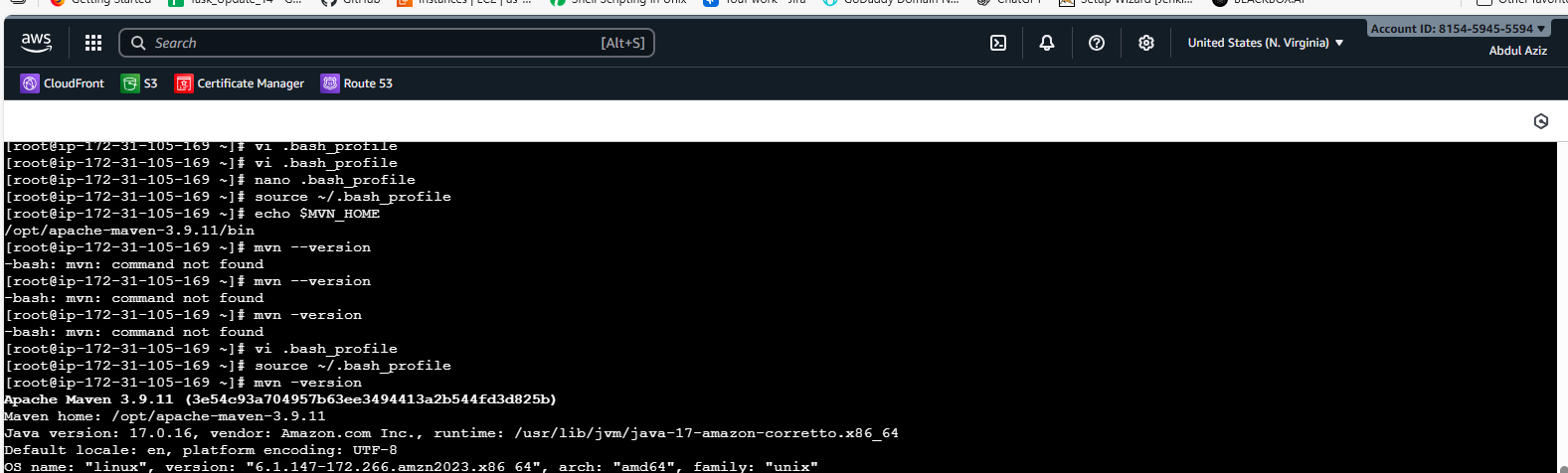
fi

# User specific environment and startup programs

export MVN\_HOME=/opt/apache-maven-3.9.11

export PATH=$PATH:$HOME/bin:$MVN\_HOME/bin

Paste this script in that bash file then save it .

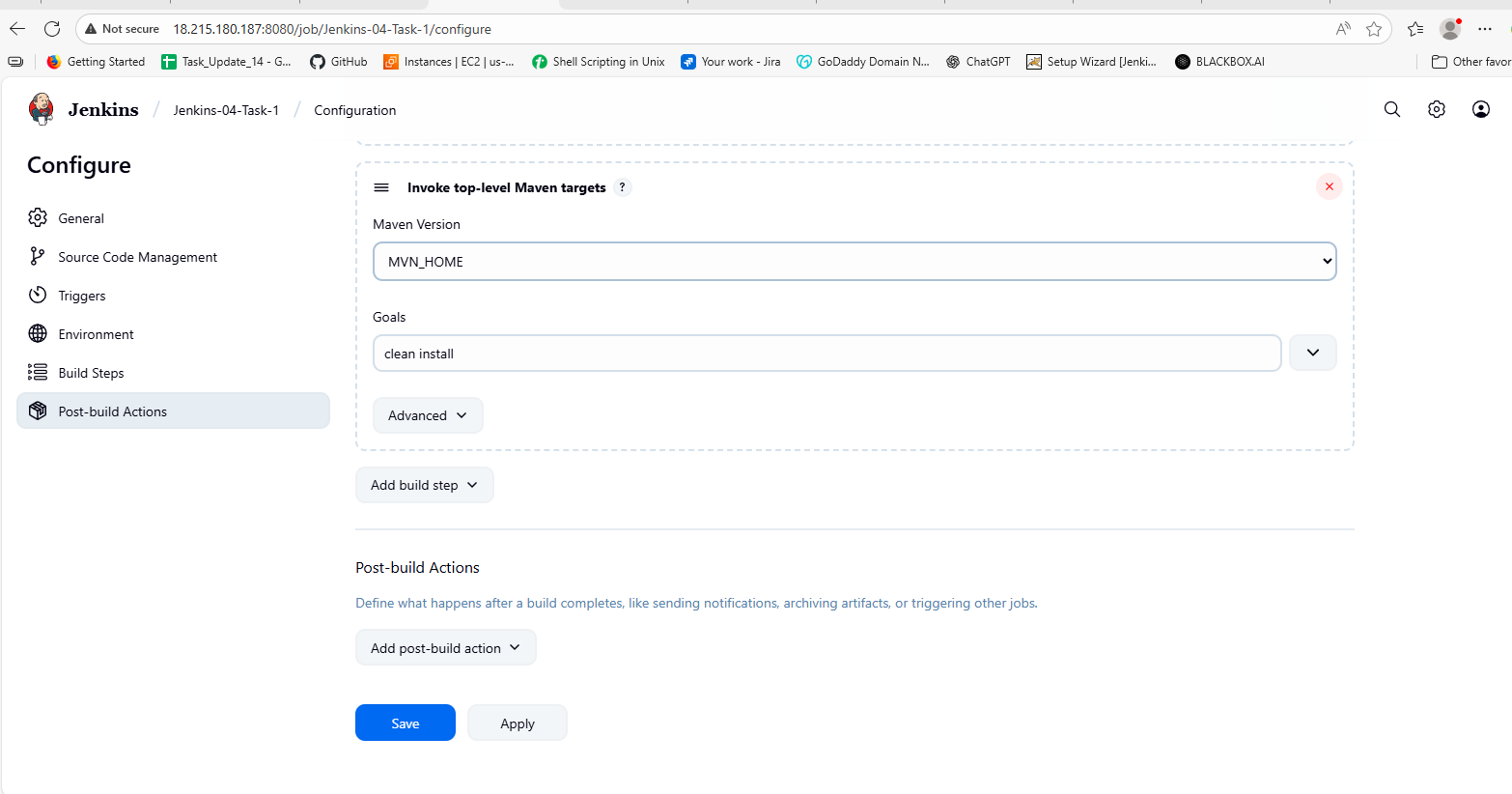


Then check the version of **mvn.**

Previous it is not showing the version we have edited it that’s why it is showing the version

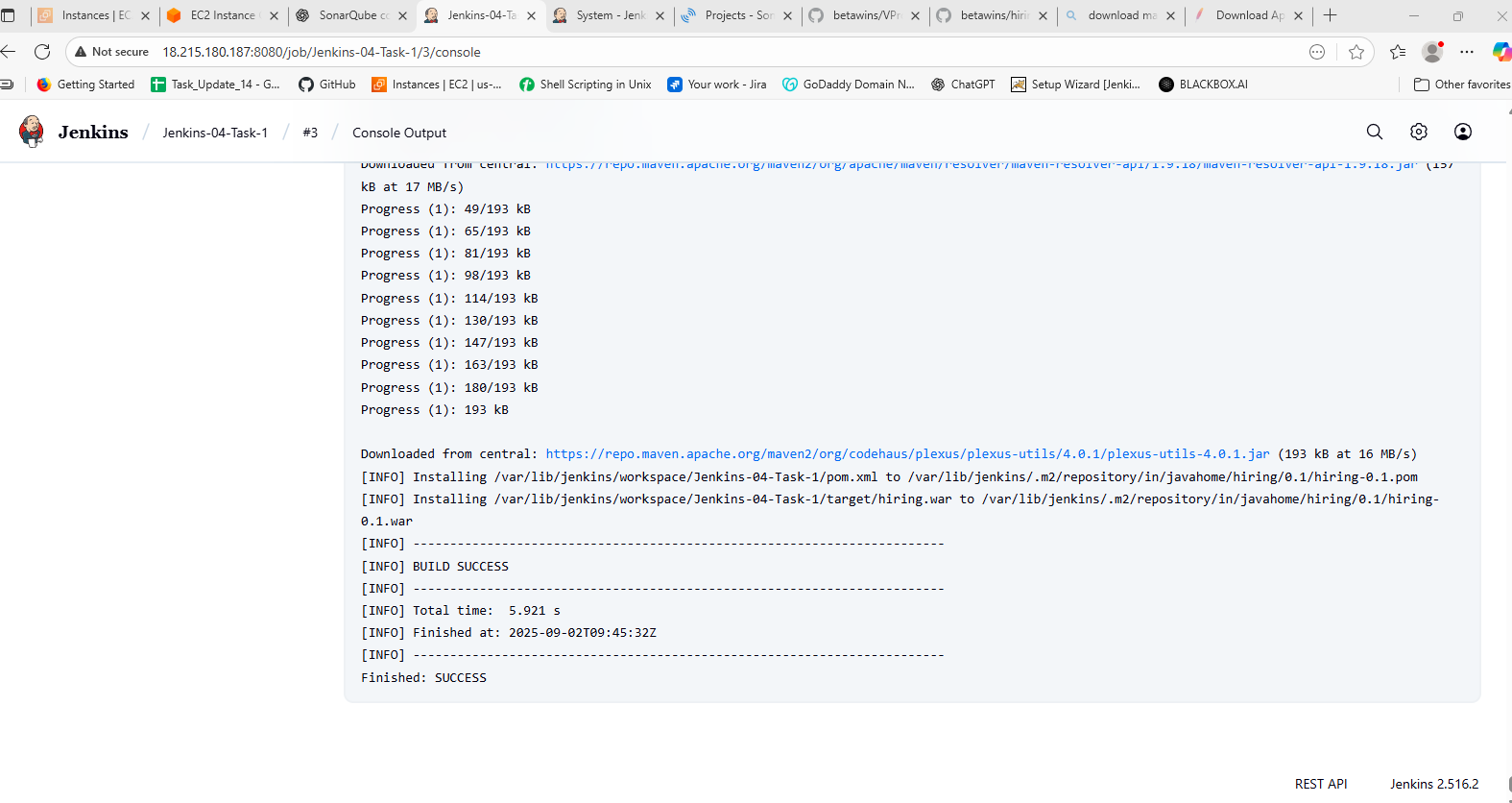
We have given that path should be executed and been check from anywhere in the machine.

After this we have to attach to the Jenkins as given in the below pic.

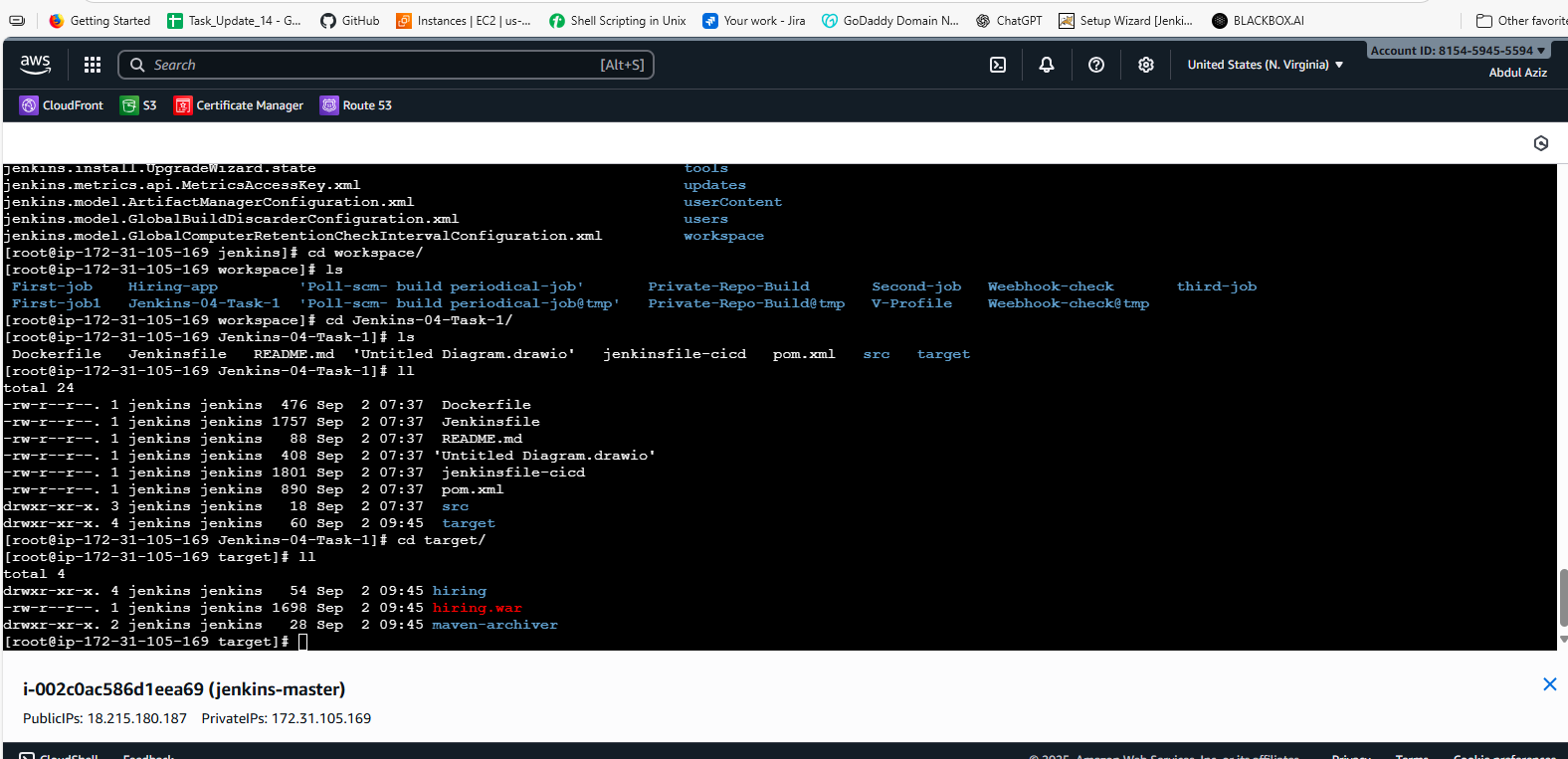


Here in goals we have to attach “clean install” this is a command that normally installs everything.

After attaching this click save. And build the job then the output will be.



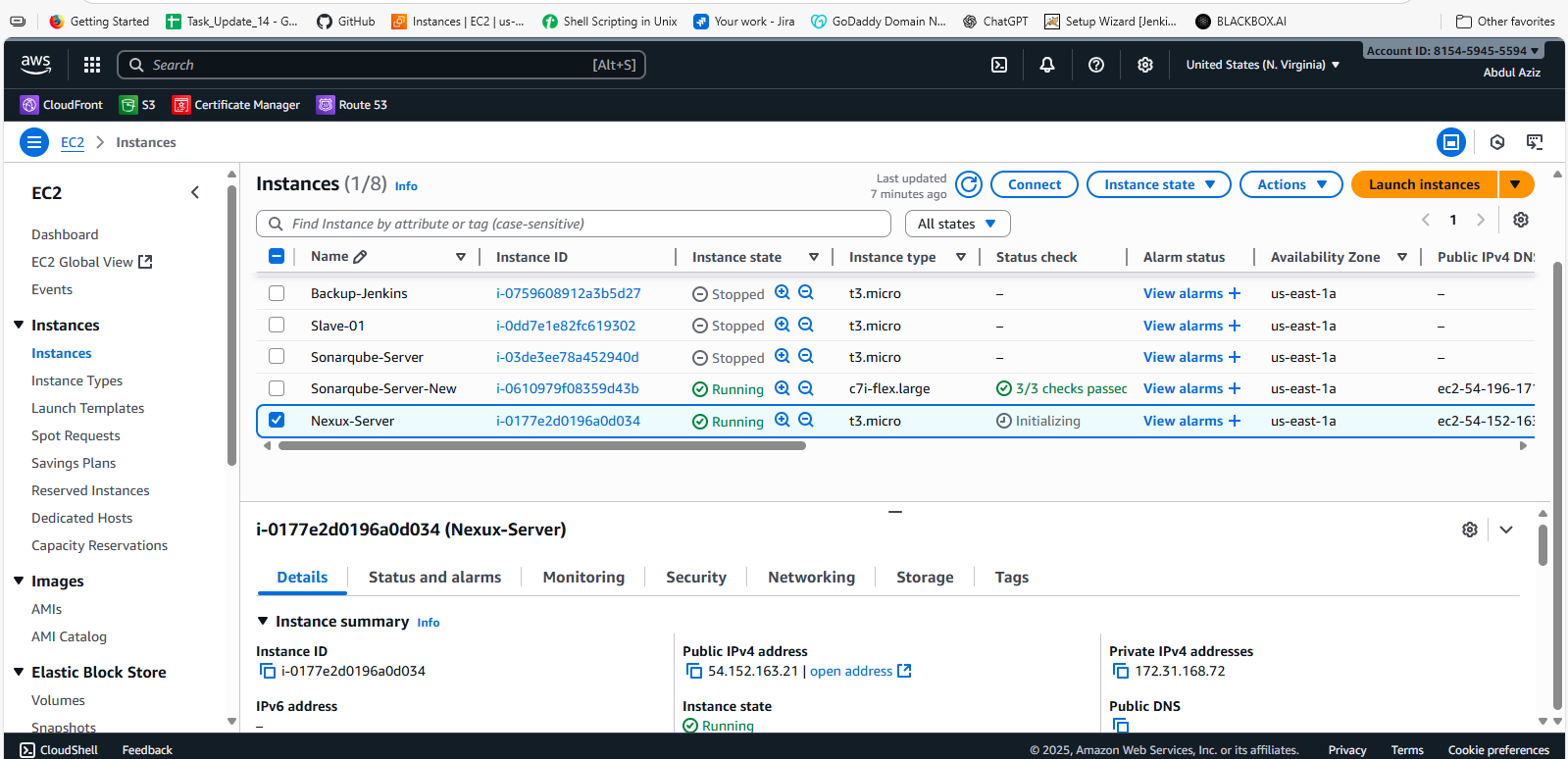
After this installed in our local machine we have see the file **target** has been created as we configure that.we can see below.



Then in target we can see the packages have created automatically after configuration in Jenkins.

1. Nexus Artifactory

First create one nexus server as given below.



Now install the nexus in thiss follow the steps given below.

cd /opt

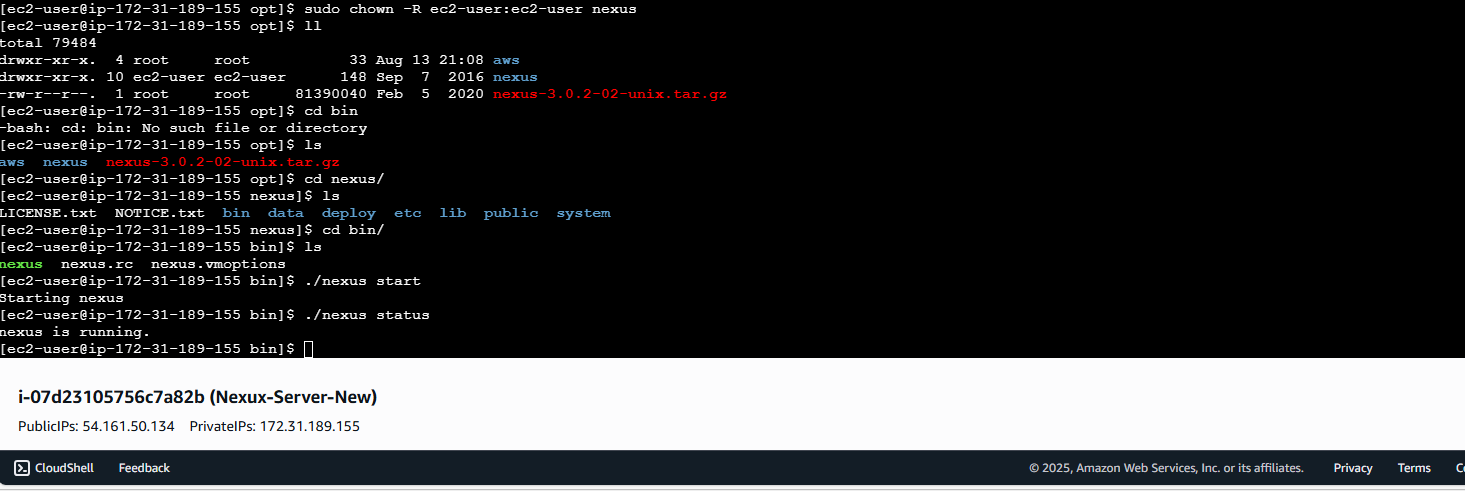
wget https://sonatype-download.global.ssl.fastly.net/nexus/3/nexus-3.0.2-02-unix.tar.gz

tar -zxvf  nexus-3.0.2-02-unix.tar.gz

mv /opt/nexus-3.0.2-02 /opt/nexus

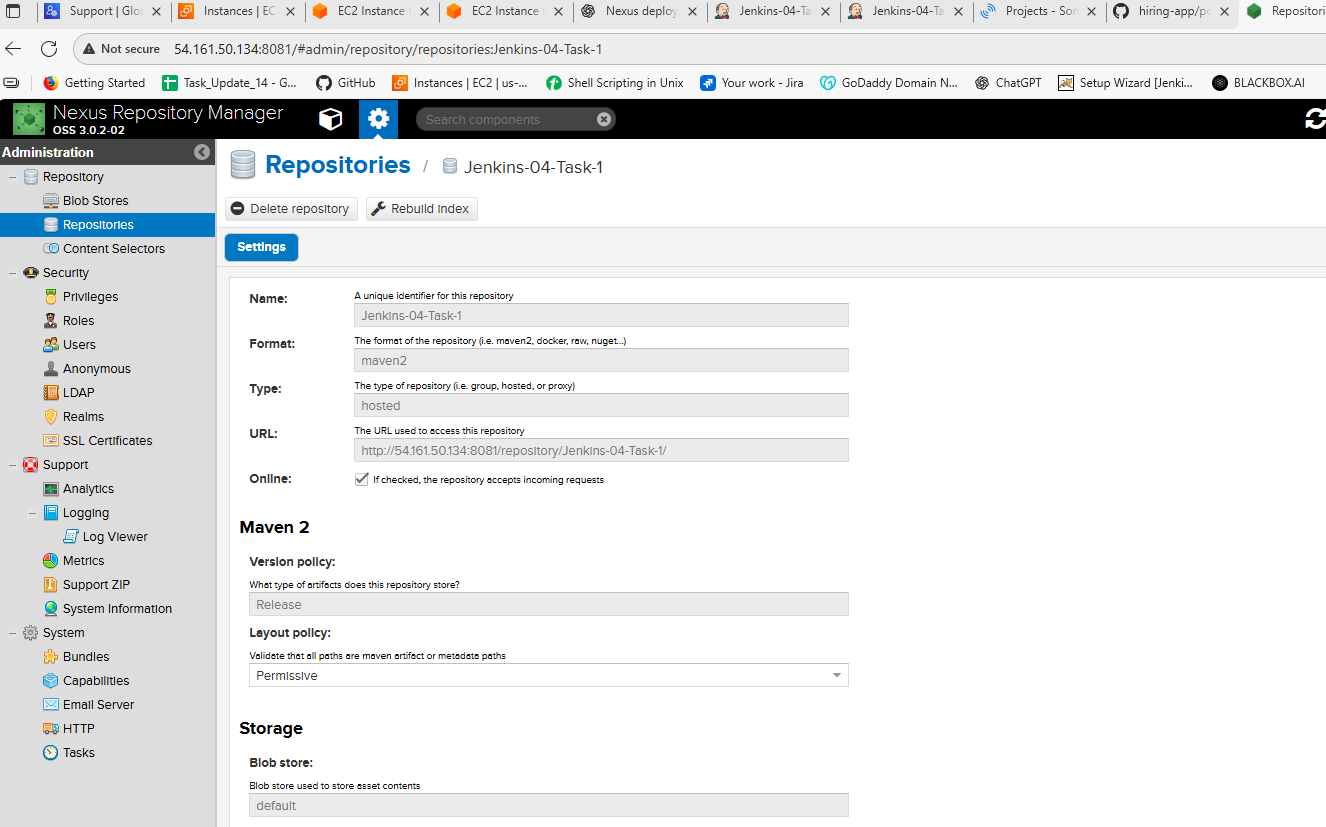
vi /opt/nexus/bin/nexus.rc

Then exit from the root and enter to the ec2 and then cd /opt/nexus then change the ownership of the nexus with chown –R ec2-user:ec2-user nexus then run the command ./nexus start then check the status now it will work.

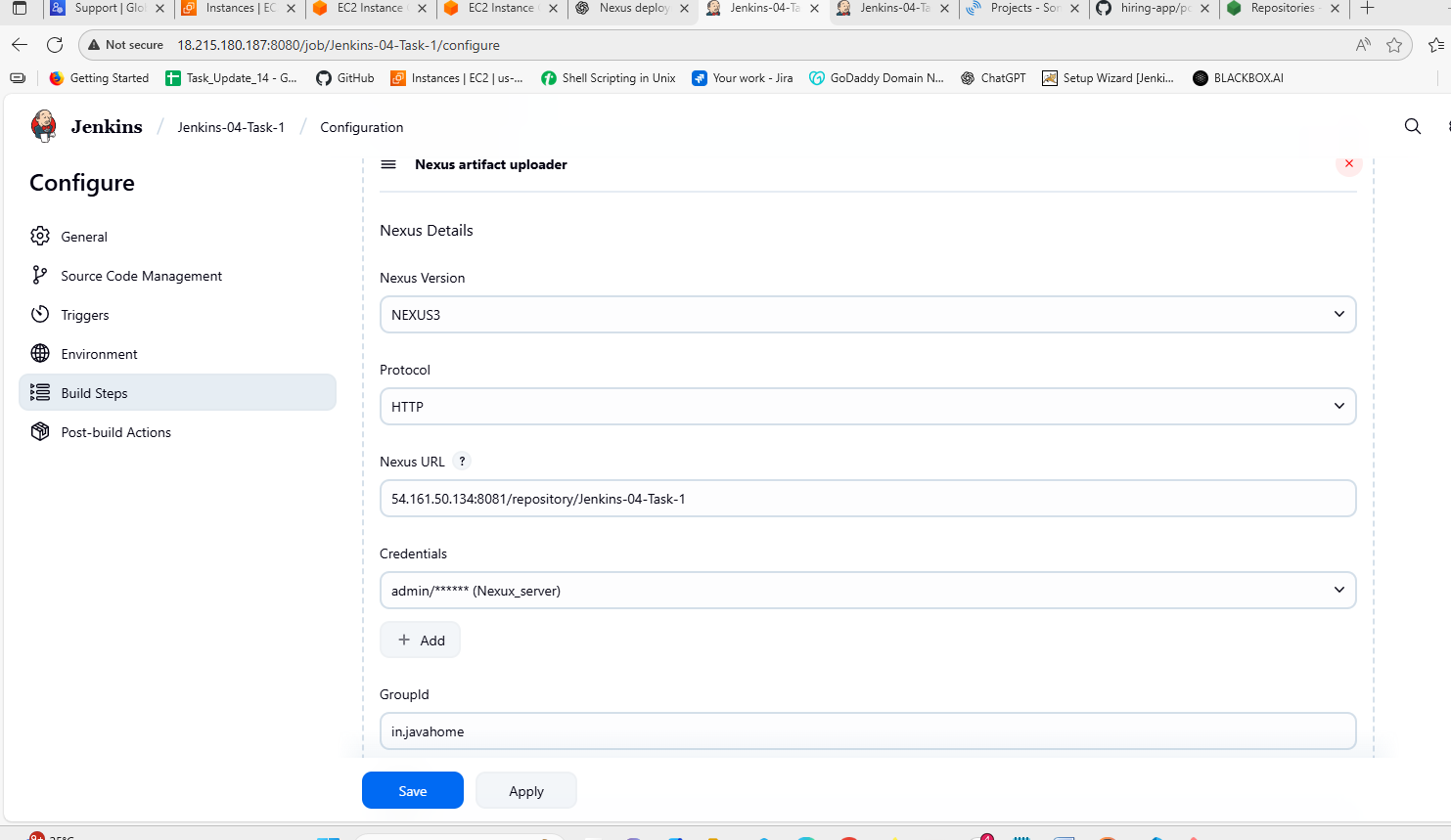


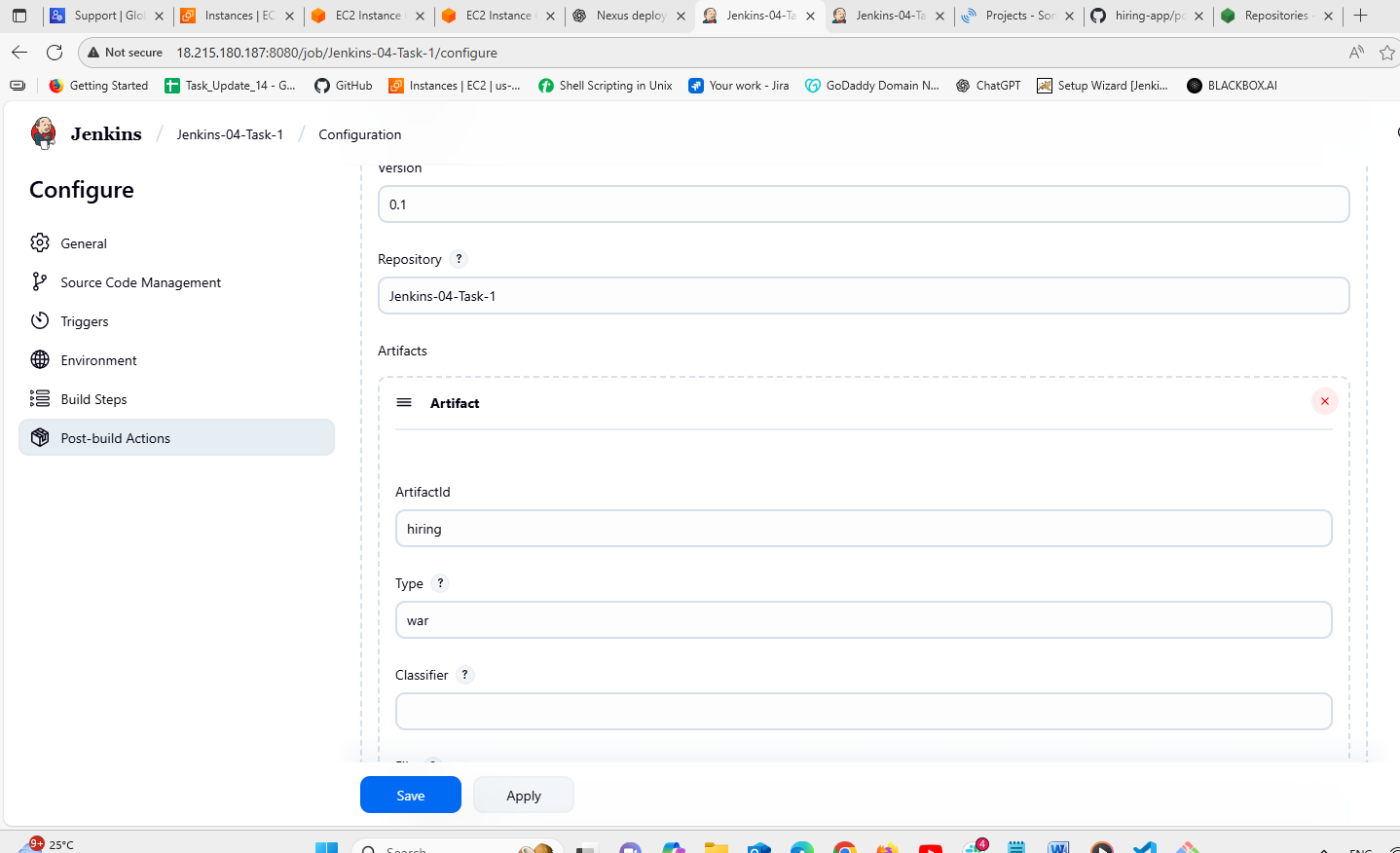
Then go to the browser and paste the public ip with the port number 8081 then the nexux page will come.

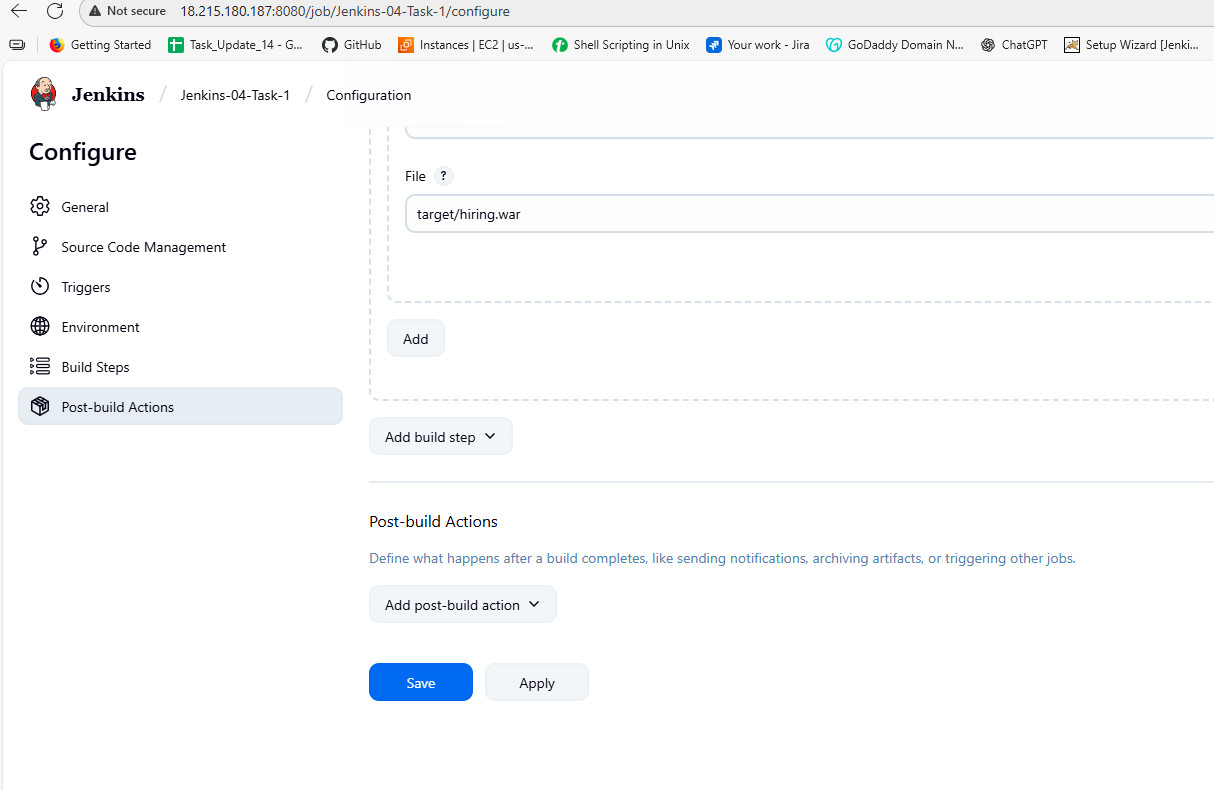
Now create repo in the nexus. As given below



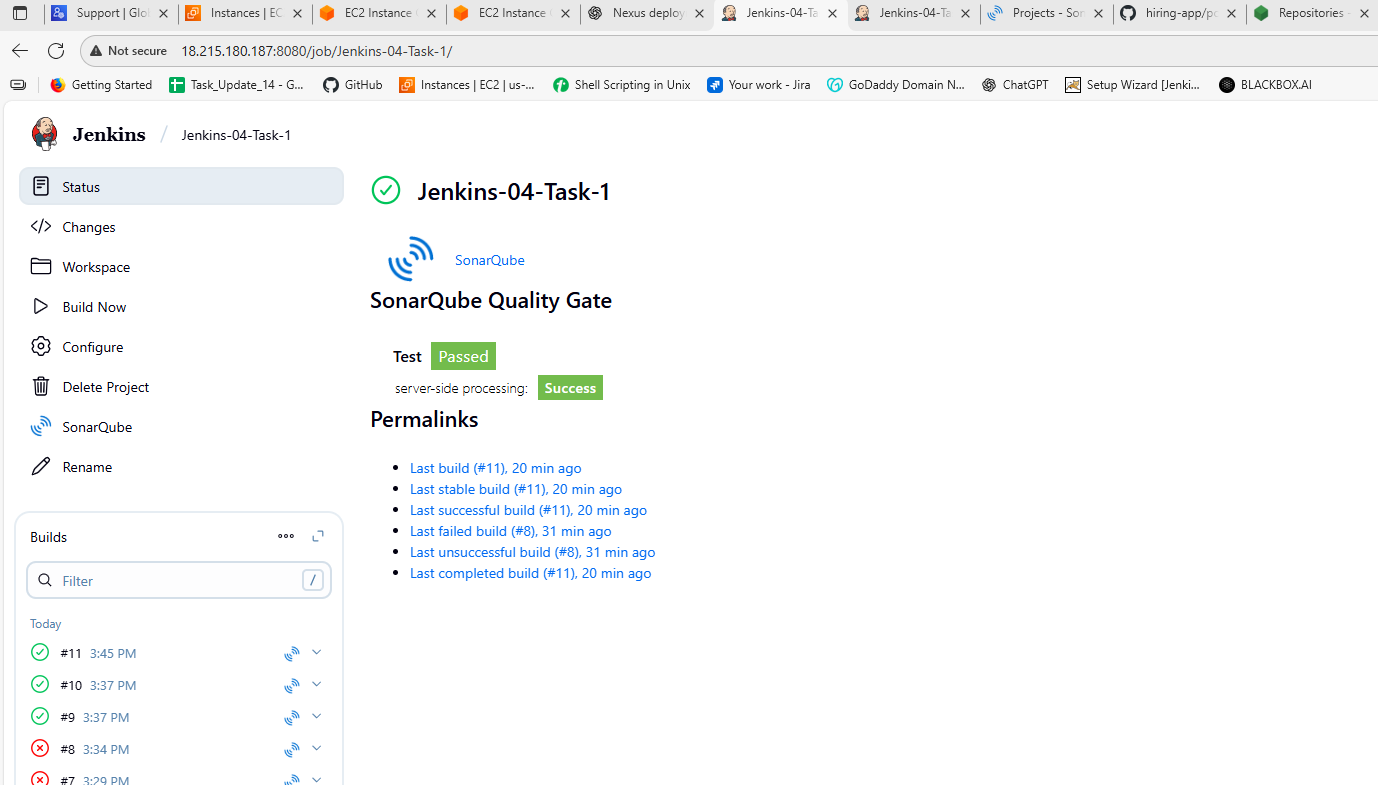
Then configure in Jenkins as given below.







Then built the job see the output below.

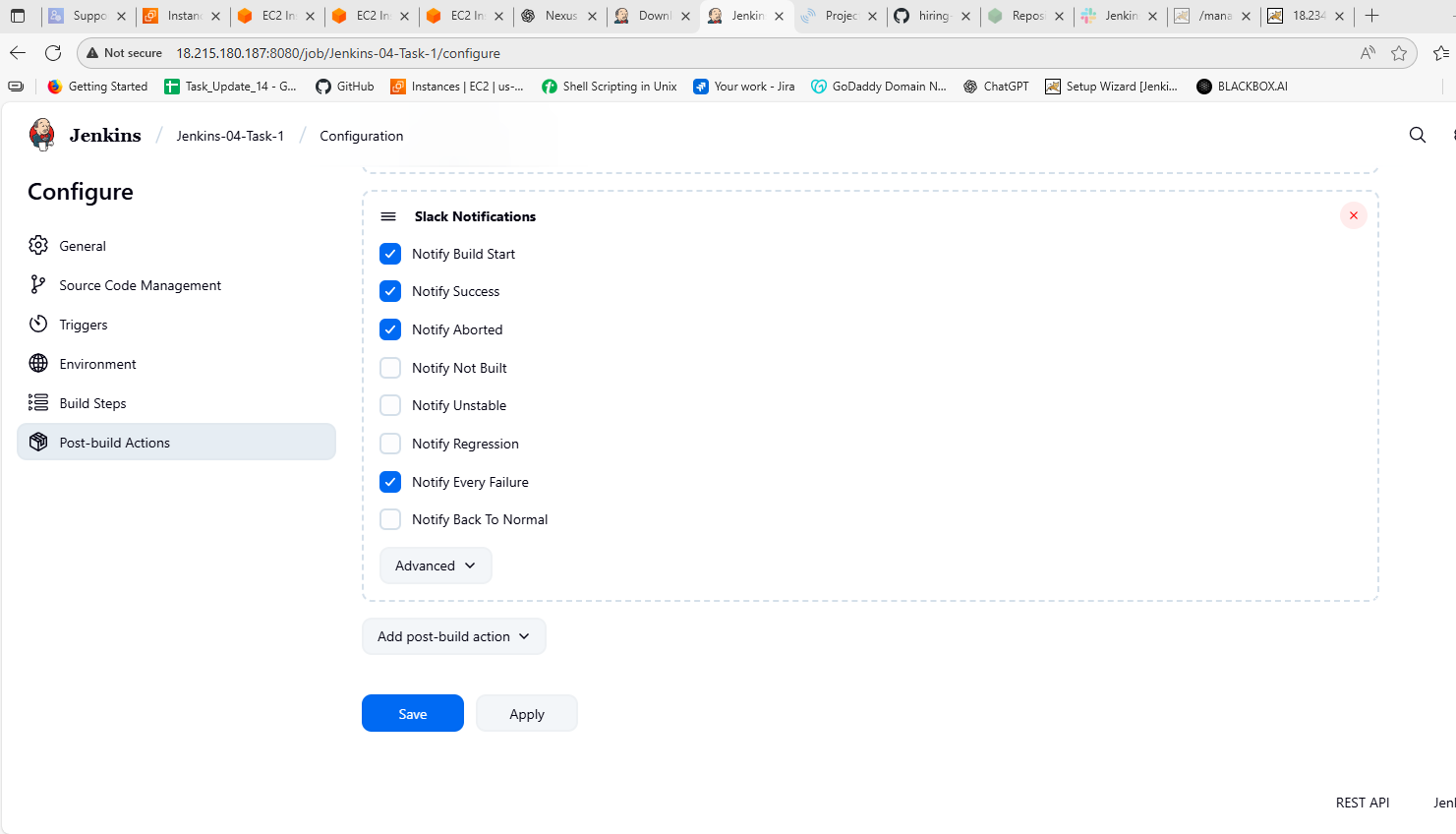


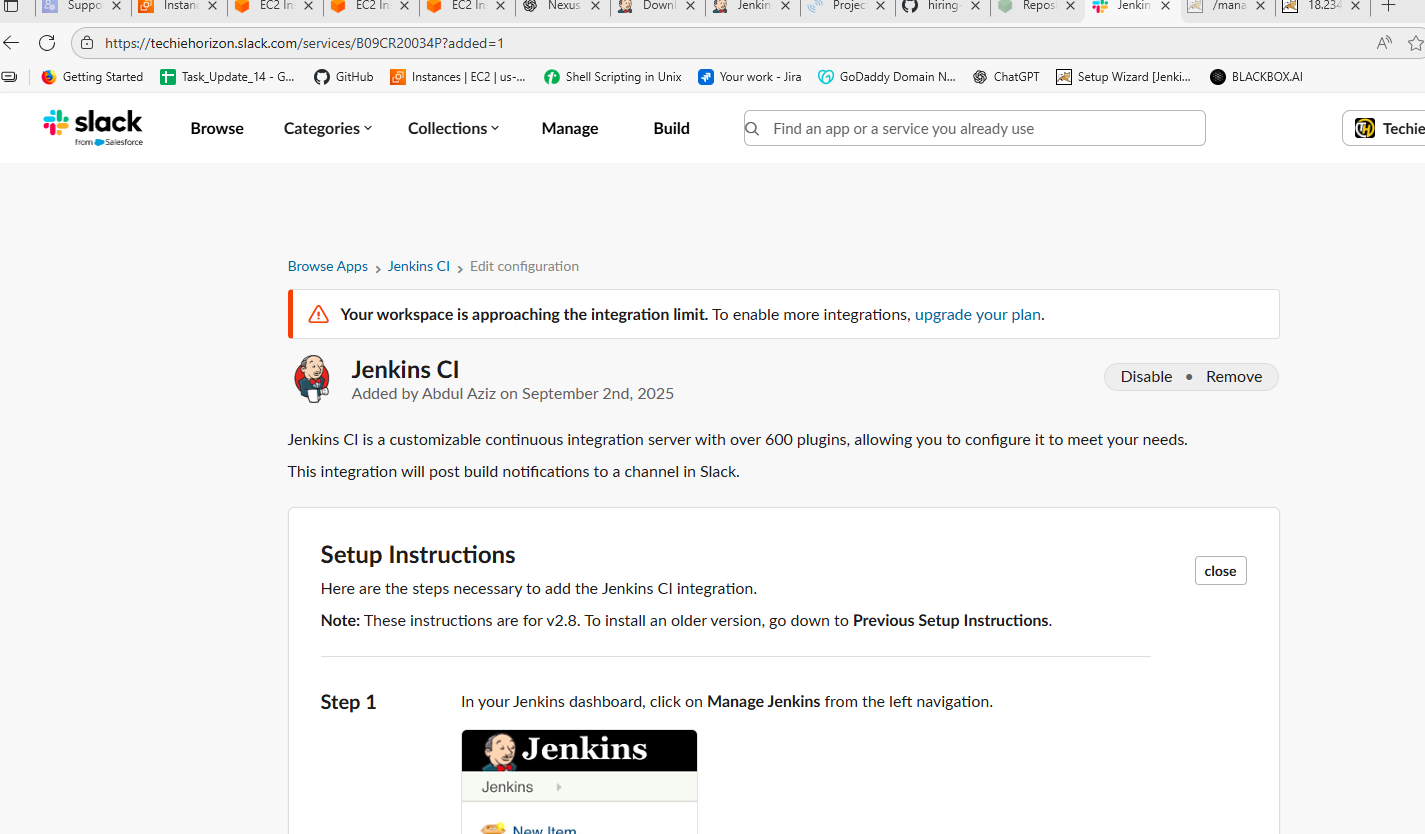
Done with the stage 4.

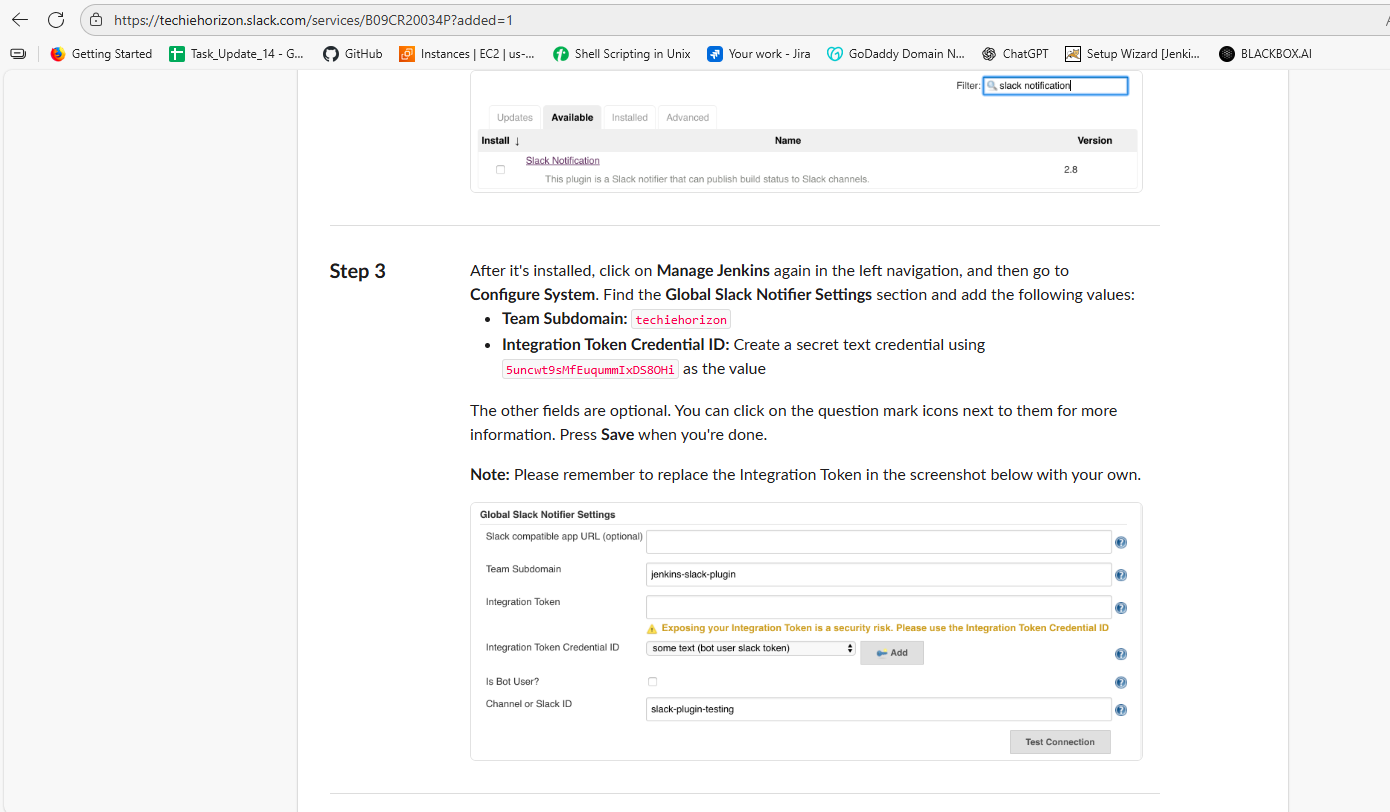
1. Slack Notification

First install the plugin slack

In this we have to configure Jenkins job like given below

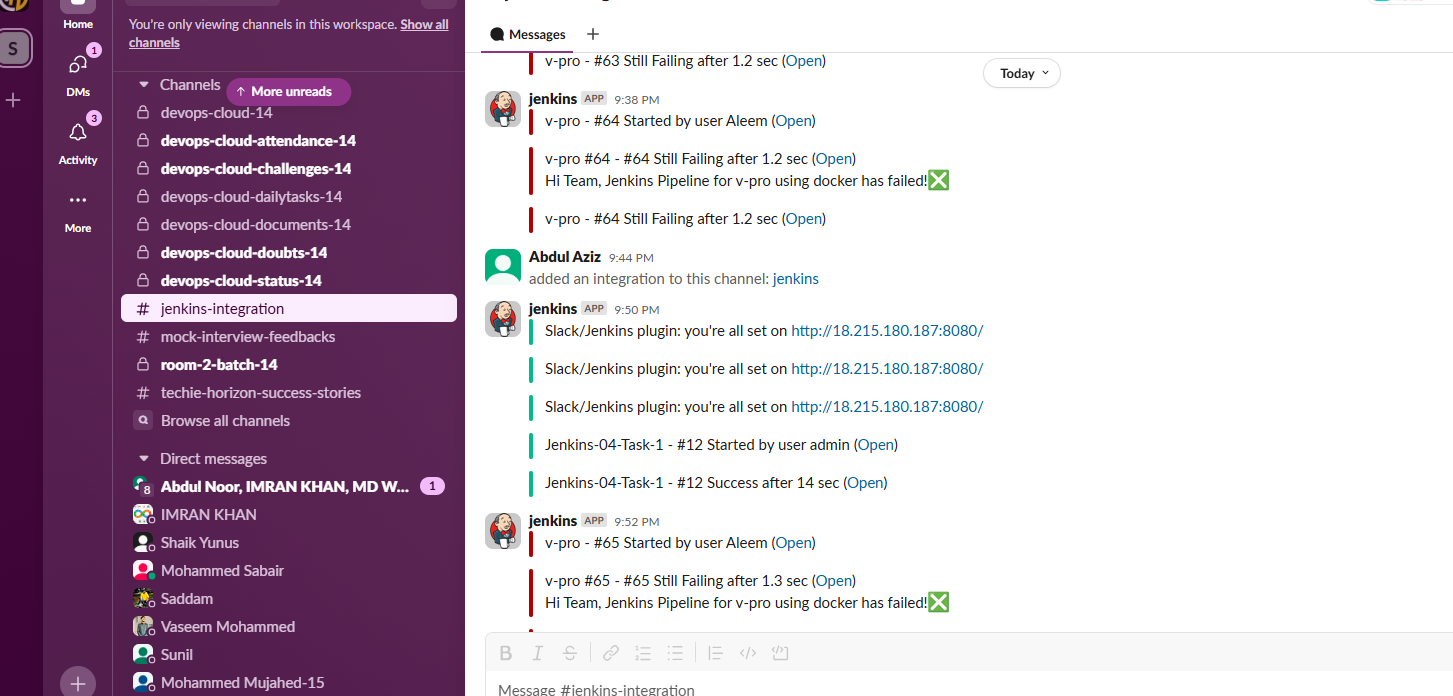






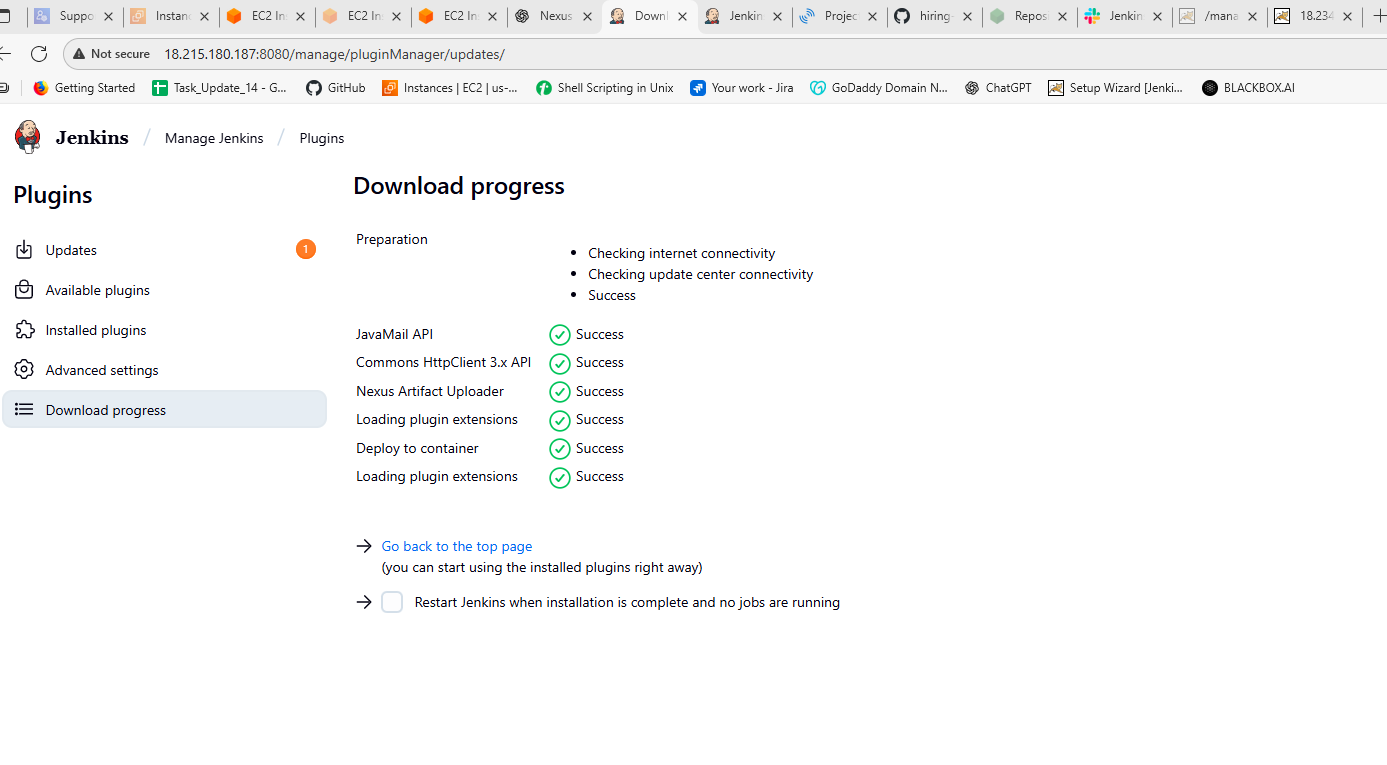
In the above picture we have the key that we have to paste in the credentials and the format should be the secret text. Id name should be the techiehorizon.

Then save it.

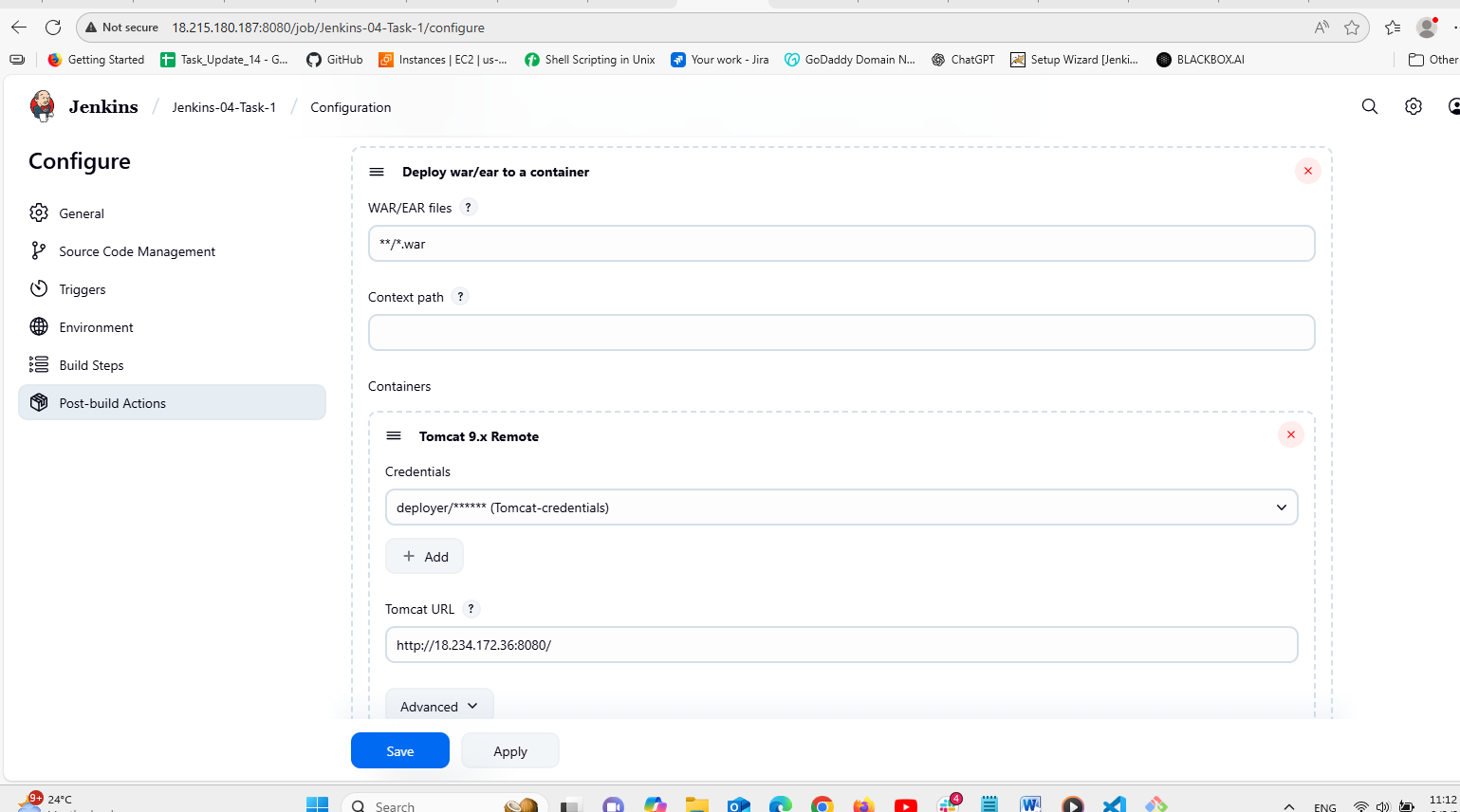


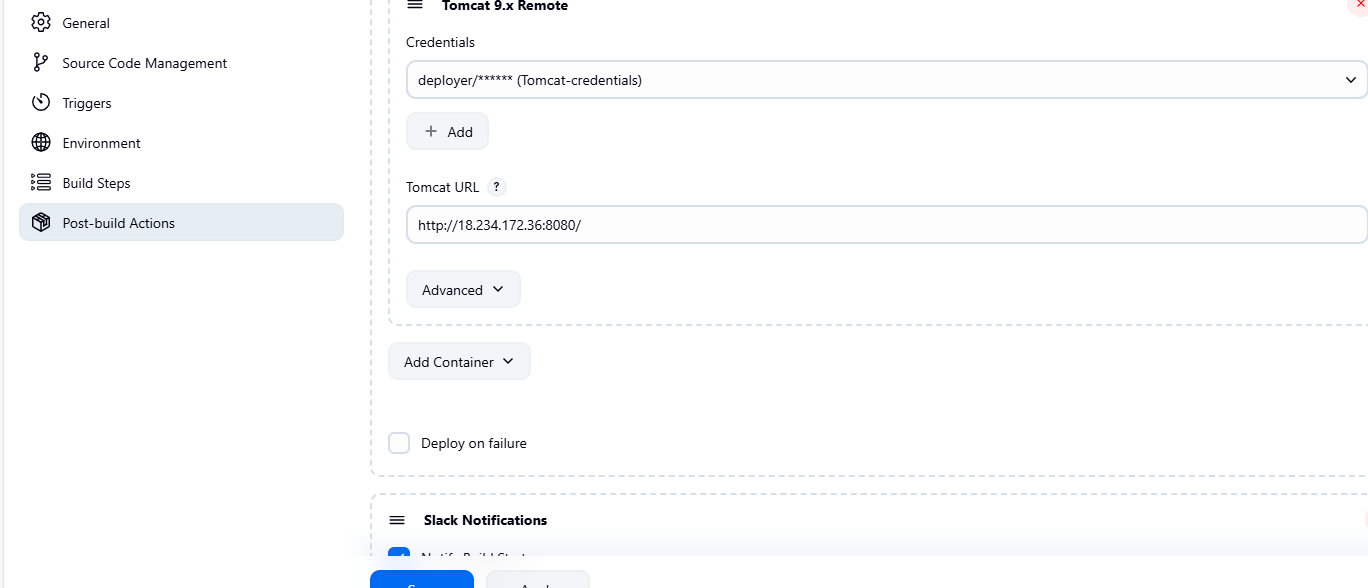
1. Deploy On tomcat

First install the plugin Deployer to..

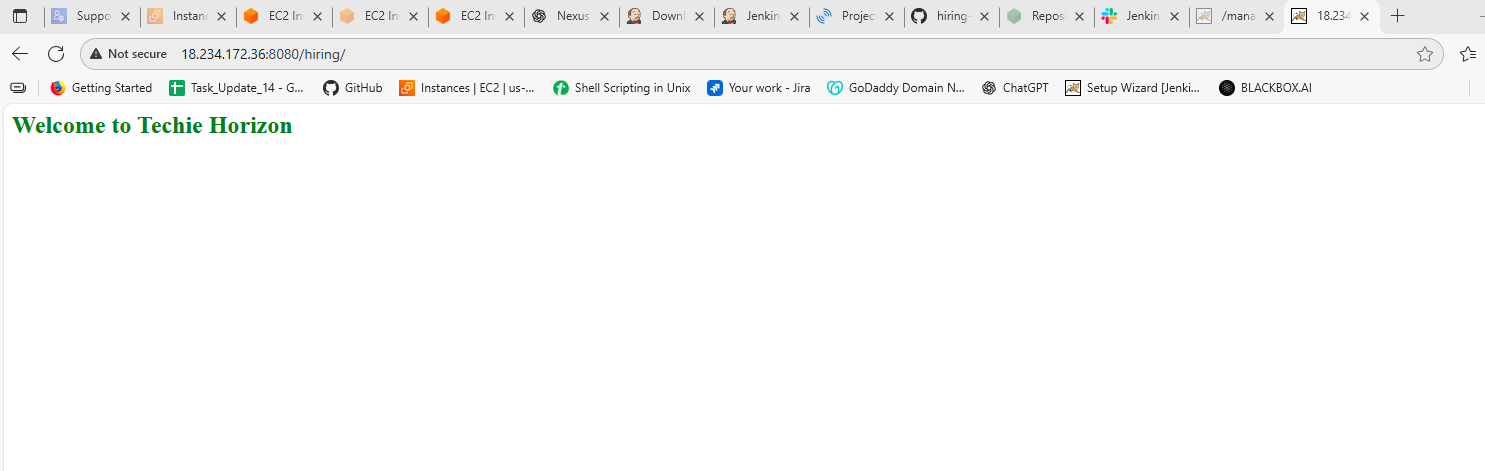


Then do the configurations in the job go to post-built action in that we can see the deploy option click it and configure as shown below.

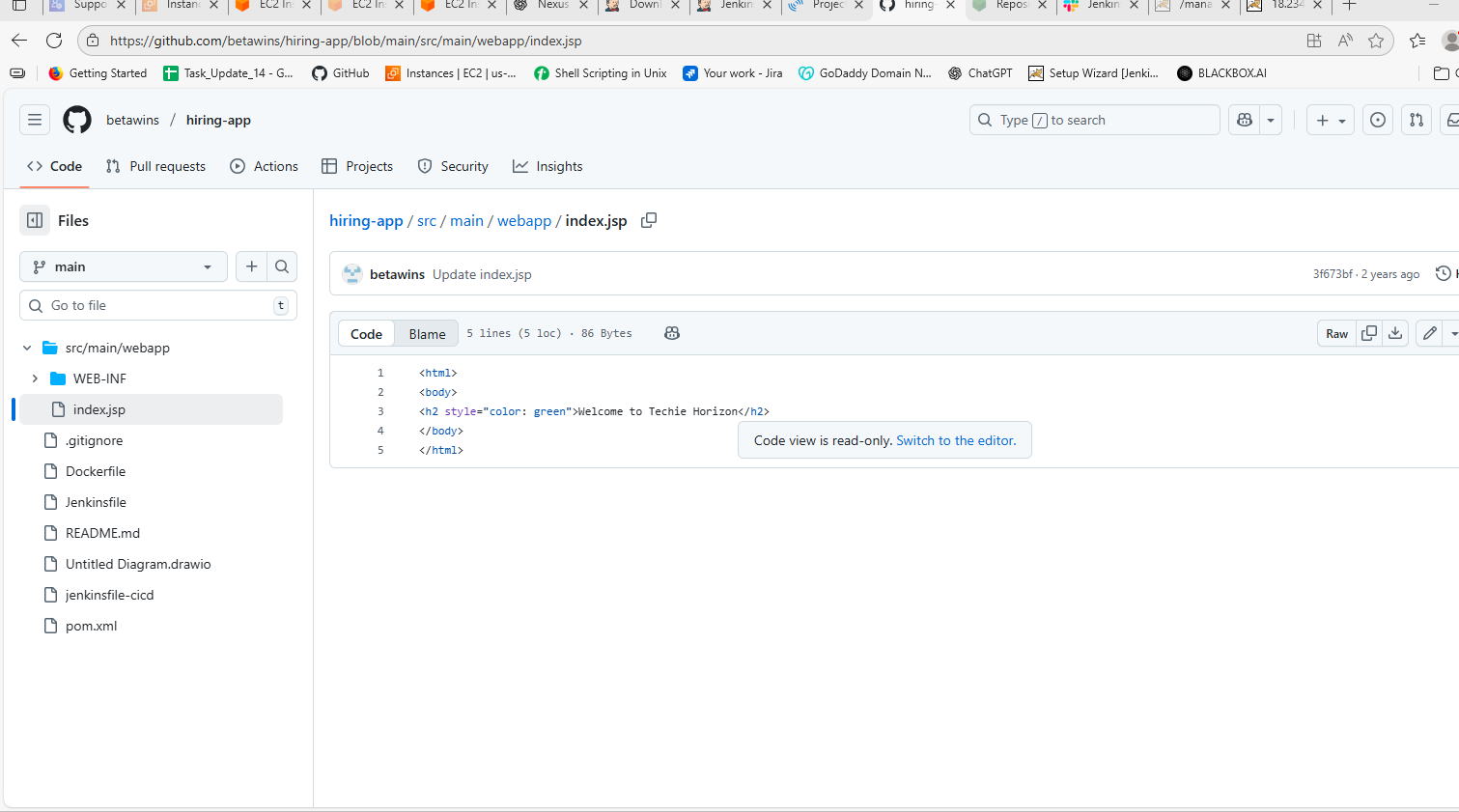




Then add container add as 9th version then save it .



Then finally go to the new page and copy the public ip and tomcat port then /hiring then serach it it shows something like the above pic. We cant edit it because the repo that has only the read access to us see below code available according to that.



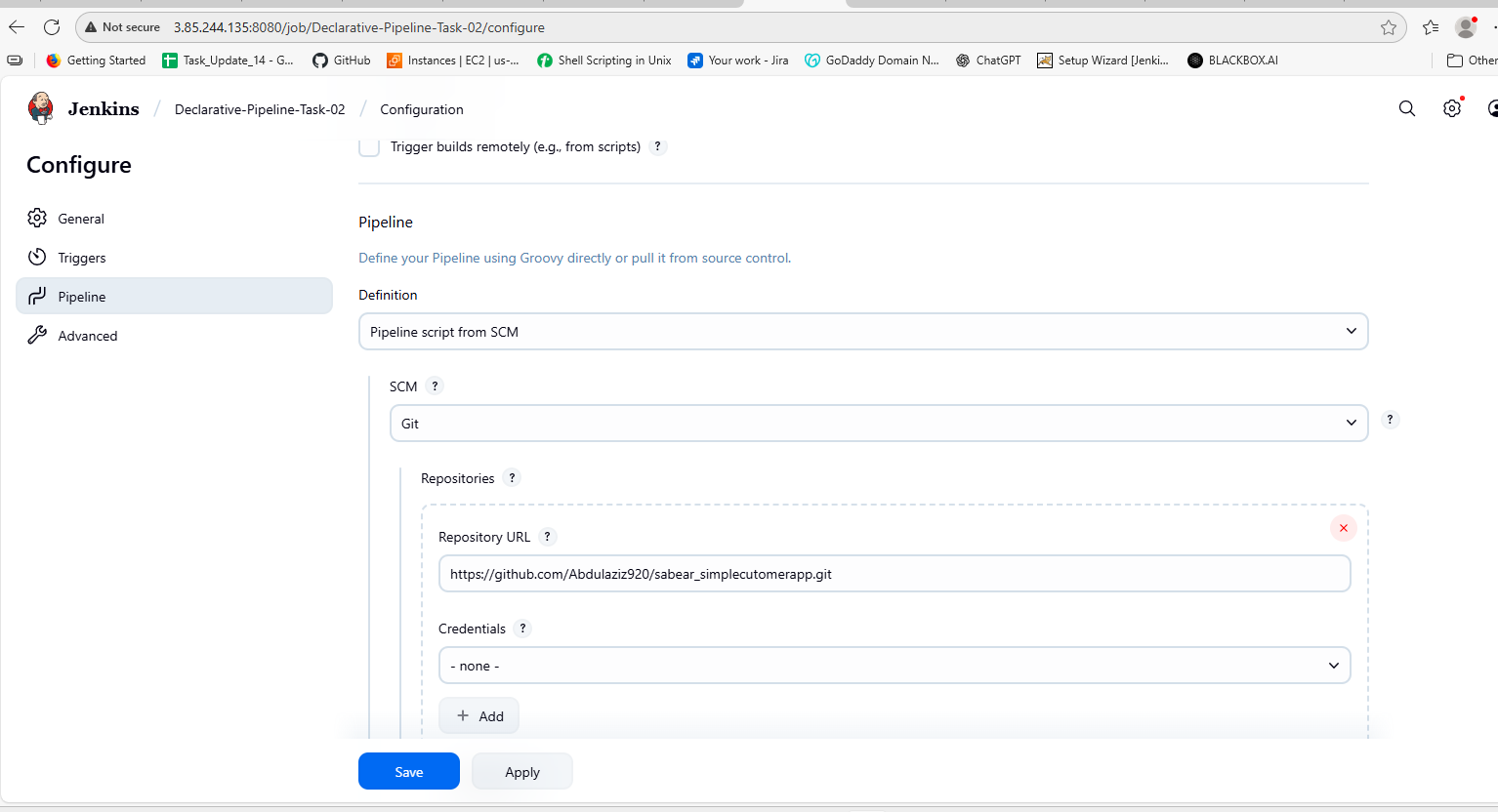
**2) Setup a jenkins CICD pipeline using Declarative pipeline using feature-1.1 branch.**

https://github.com/betawins/sabear\_simplecutomerapp/tree/feature-1.1

stages:

1. Git Clone

I have created a pipeline job see below and attach the git url and lablethe branch and save it



As we are doing the Declarative so we have to take the pipe line as SCM and save it then built job . or else configure the second stage.

1. Sonarqube Integration

Basically we have done the configurations in the Jenkins systems or tools so onlt we have to add the stage in the Jenkins file given below.

stage("SonarCloud") {

steps {

withSonarQubeEnv('sonar-scanner') {

sh '''$SCANNER\_HOME/bin/sonar-scanner \

-Dsonar.projectKey=Ncodeit \

-Dsonar.projectName=Ncodeit \

-Dsonar.projectVersion=2.0 \

-Dsonar.sources=/var/lib/jenkins/workspace/$JOB\_NAME/src/ \

-Dsonar.binaries=target/classes/com/visualpathit/account/controller/ \

-Dsonar.junit.reportsPath=target/surefire-reports \

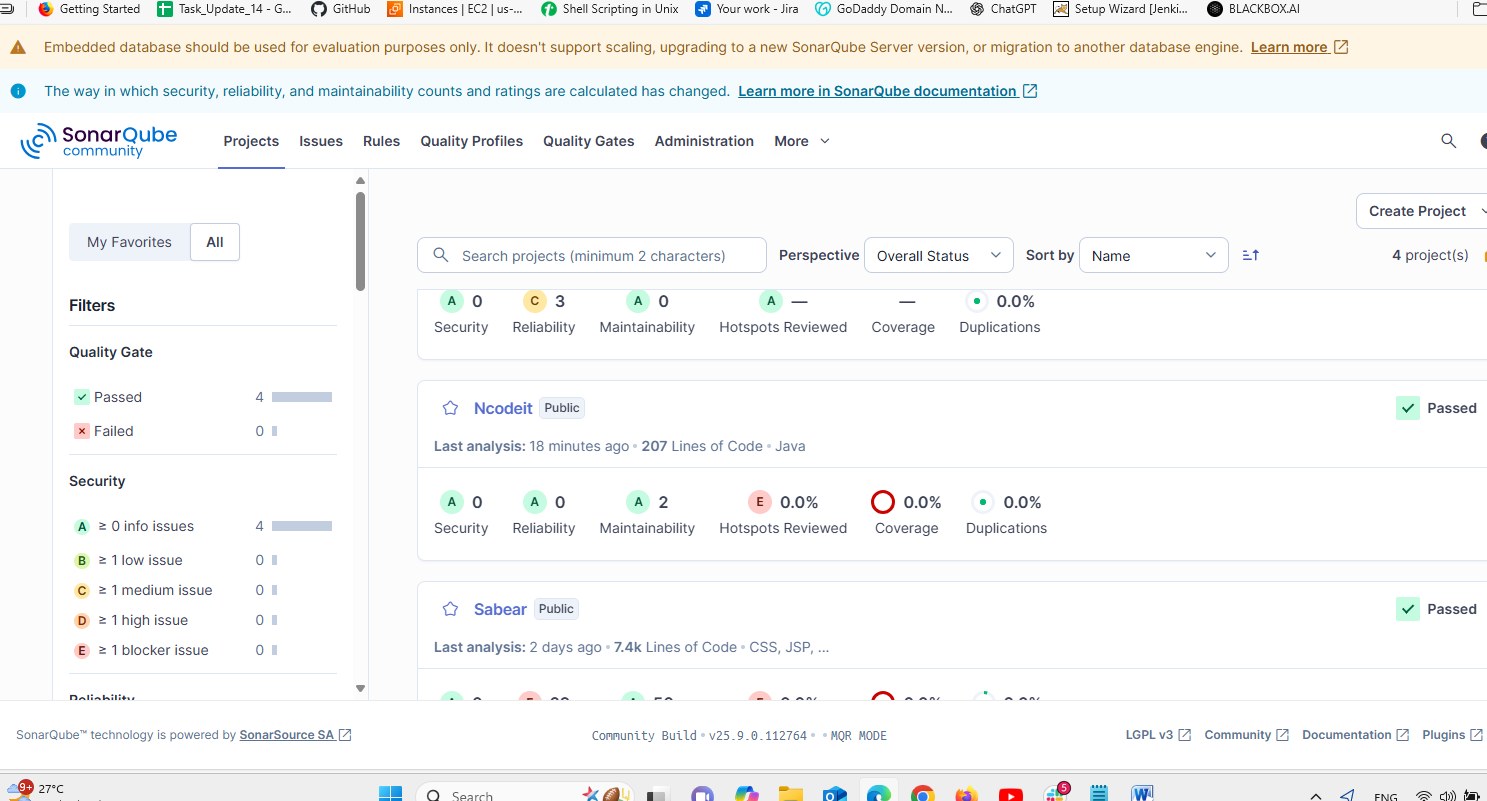
-Dsonar.jacoco.reportPath=target/jacoco.exec \

-Dsonar.java.binaries=src/com/room/sample '''

}

}

}



1. Maven Compilation

Paste the stage in jenkinsfile

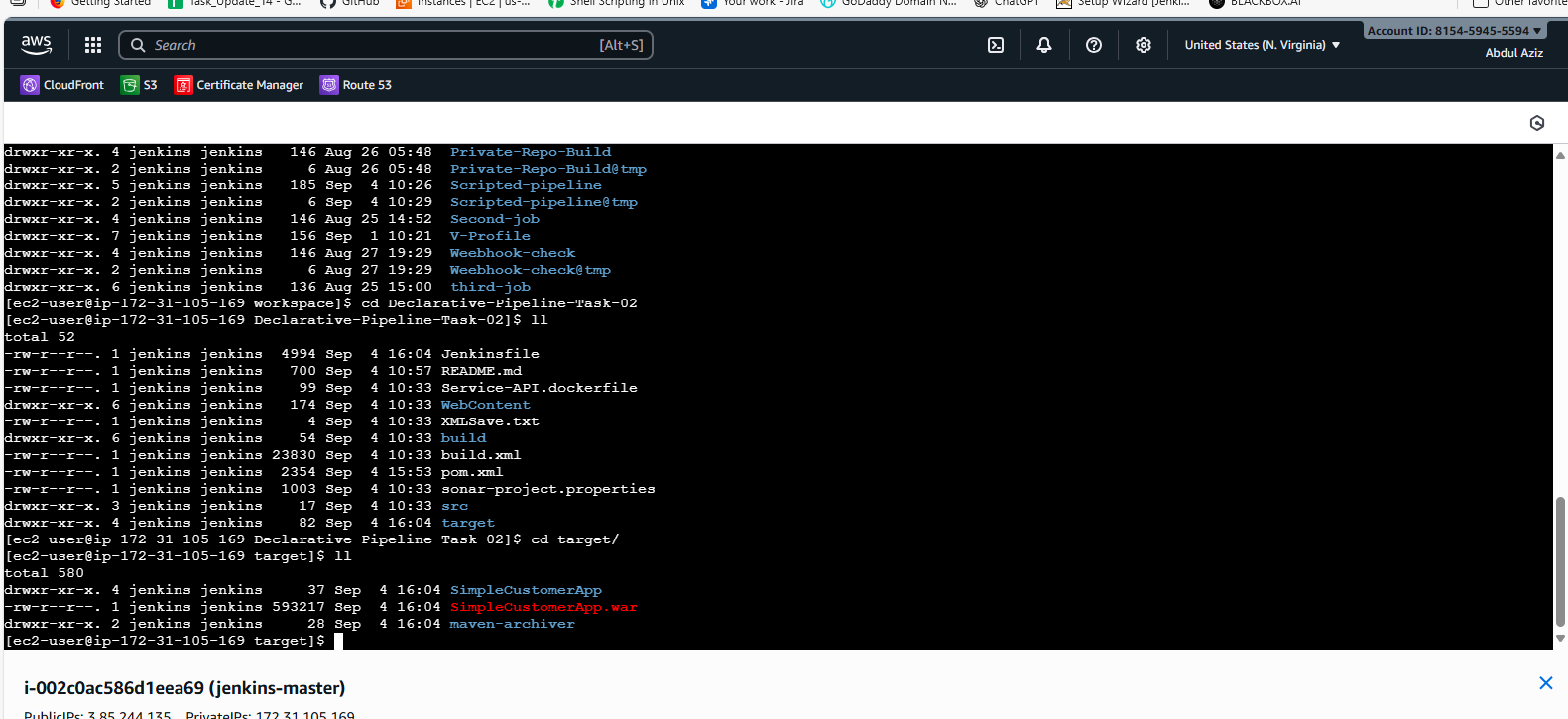
stage('Build') {

steps {

sh 'mvn -Dmaven.test.failure.ignore=true clean install'

}

}



1. Nexus Artifactory

stage("publish to nexus") {

steps {

script {

pom = readMavenPom file: "pom.xml"

filesByGlob = findFiles(glob: "target/\*.${pom.packaging}")

echo "${filesByGlob[0].name} ${filesByGlob[0].path}"

artifactPath = filesByGlob[0].path

artifactExists = fileExists artifactPath

if (artifactExists) {

nexusArtifactUploader(

nexusVersion: NEXUS\_VERSION,

protocol: NEXUS\_PROTOCOL,

nexusUrl: NEXUS\_URL,

groupId: pom.groupId,

version: pom.version,

repository: NEXUS\_REPOSITORY,

credentialsId: NEXUS\_CREDENTIAL\_ID,

artifacts: [

[artifactId: pom.artifactId, classifier: '', file: artifactPath, type: pom.packaging],

[artifactId: pom.artifactId, classifier: '', file: "pom.xml", type: "pom"]

]

)

} else {

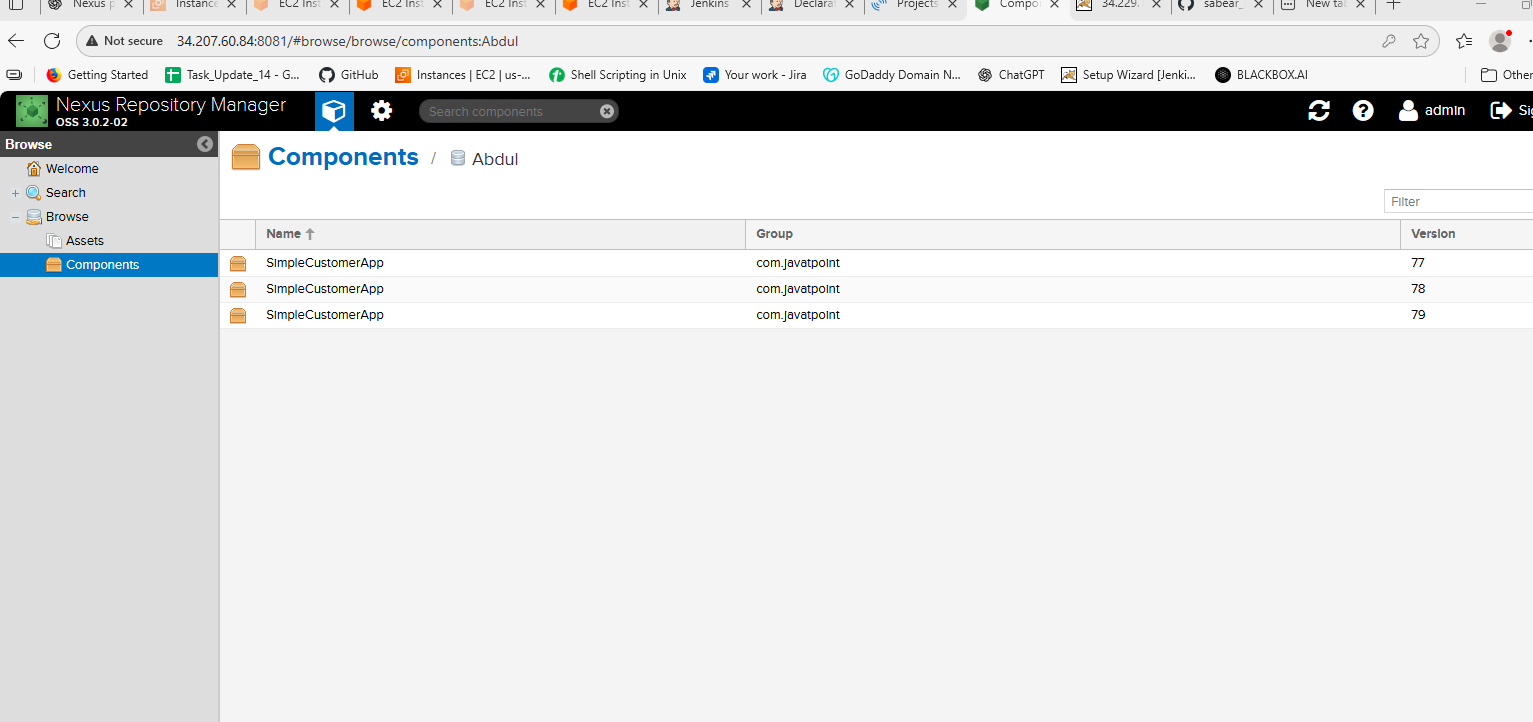
error "\*\*\* File: ${artifactPath}, could not be found"

}

}

}

}



1. Slack Notification

stage("Slack Notification") {

steps {

slackSend(

channel: "${SLACK\_CHANNEL}",

color: "#36A64F",

message: "Declarative pipeline for \*Simple Customer App\* has been successfully deployed in Tomcat :white\_check\_mark: by Abdul Aziz for Job: ${env.JOB\_NAME} [${env.BUILD\_NUMBER}]"

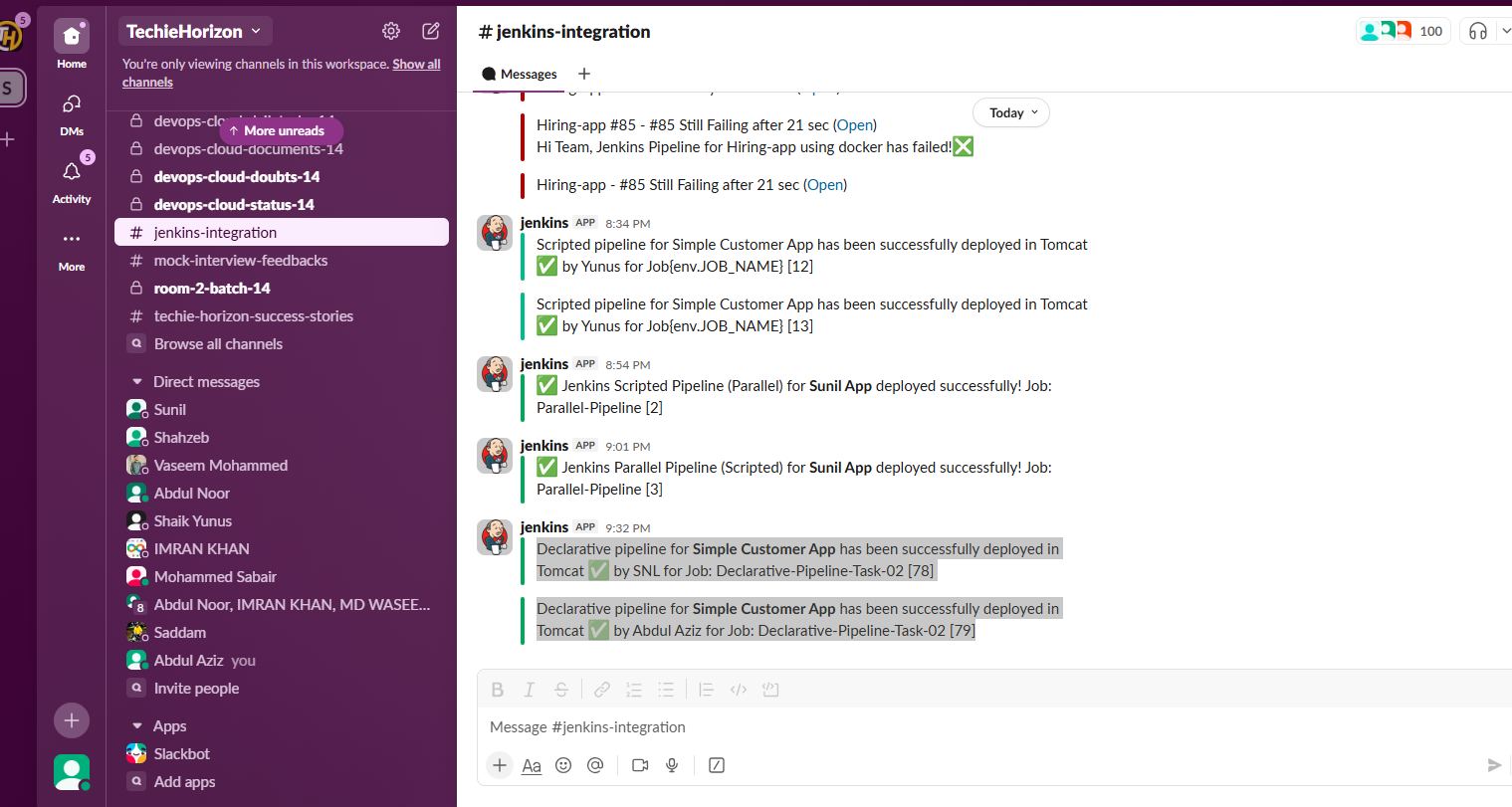
)

}

}

}

}



1. Deploy On tomcat

stage("Deploy to Tomcat") {

steps {

withCredentials([usernamePassword(credentialsId: 'Tomcat-credentials', usernameVariable: 'TOMCAT\_USER', passwordVariable: 'TOMCAT\_PASS')]) {

script {

// Find the WAR file built by Maven

def warFile = sh(script: "ls target/\*.war | head -n 1", returnStdout: true).trim()

echo "Deploying ${warFile} to Tomcat at context path /simplecustomerapp ..."

sh """

curl -u $TOMCAT\_USER:$TOMCAT\_PASS \

-T ${warFile} \

"http://34.229.166.230:8080/manager/text/deploy?path=/simplecustomerapp&update=true"

"""

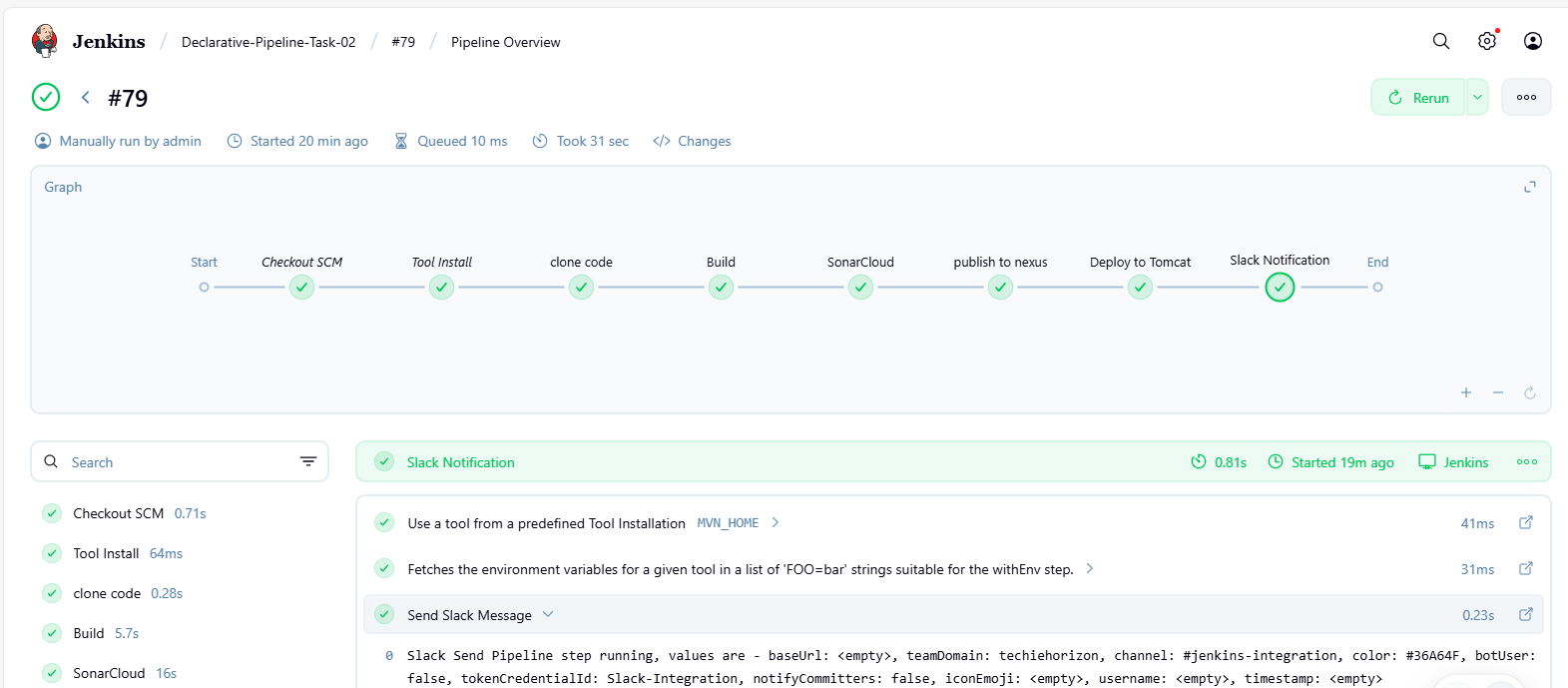
}

}

}

}

After doing this all steps we have to built the job then we can see like thiss



3) **Setup a jenkins CICD pipeline using Scripted pipeline using feature-1.1 branch.**

https://github.com/betawins/sabear\_simplecutomerapp/tree/feature-1.1

stages:

1. Git Clone

2) Sonarqube Integration

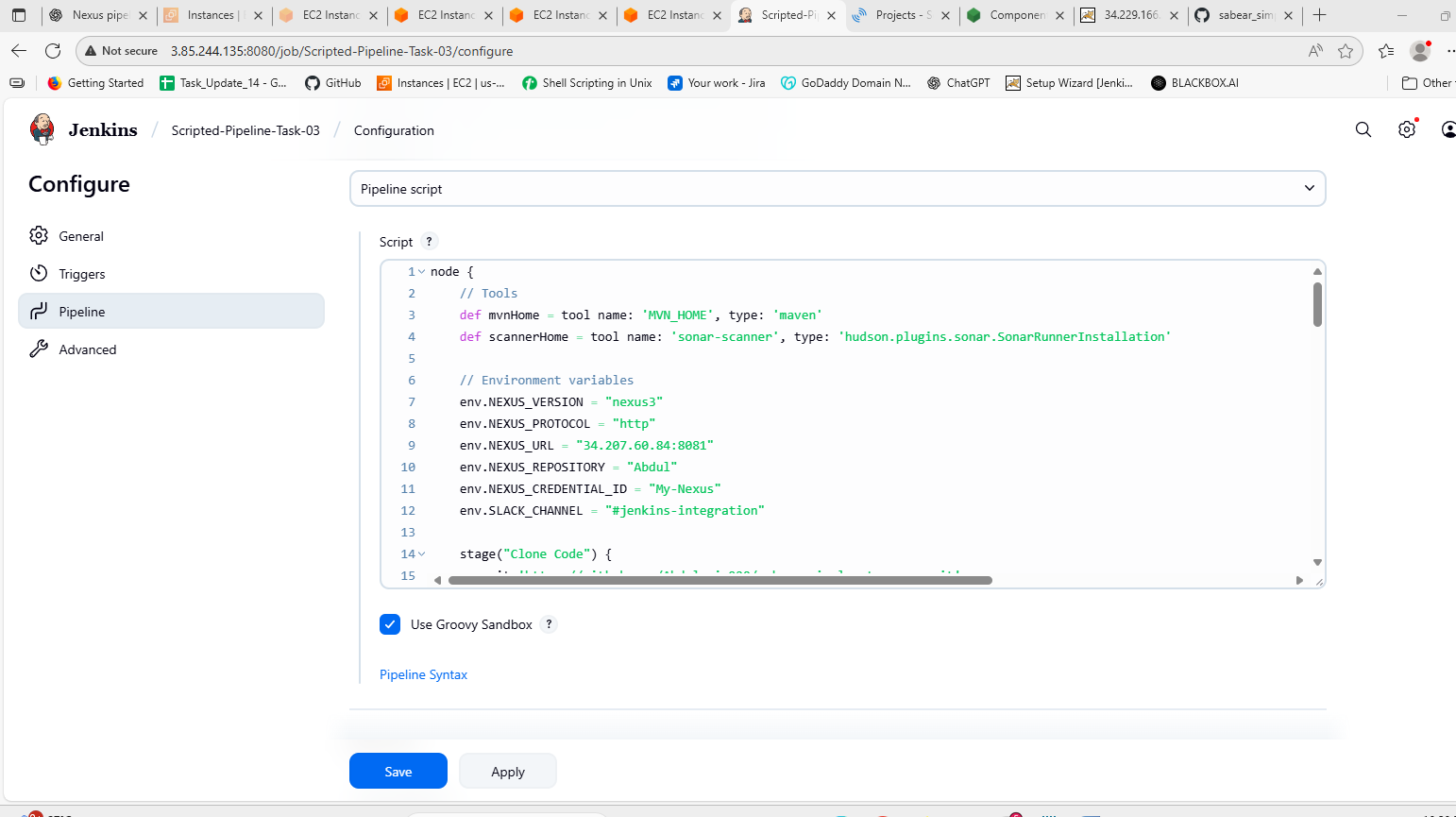
3) Maven Compilation

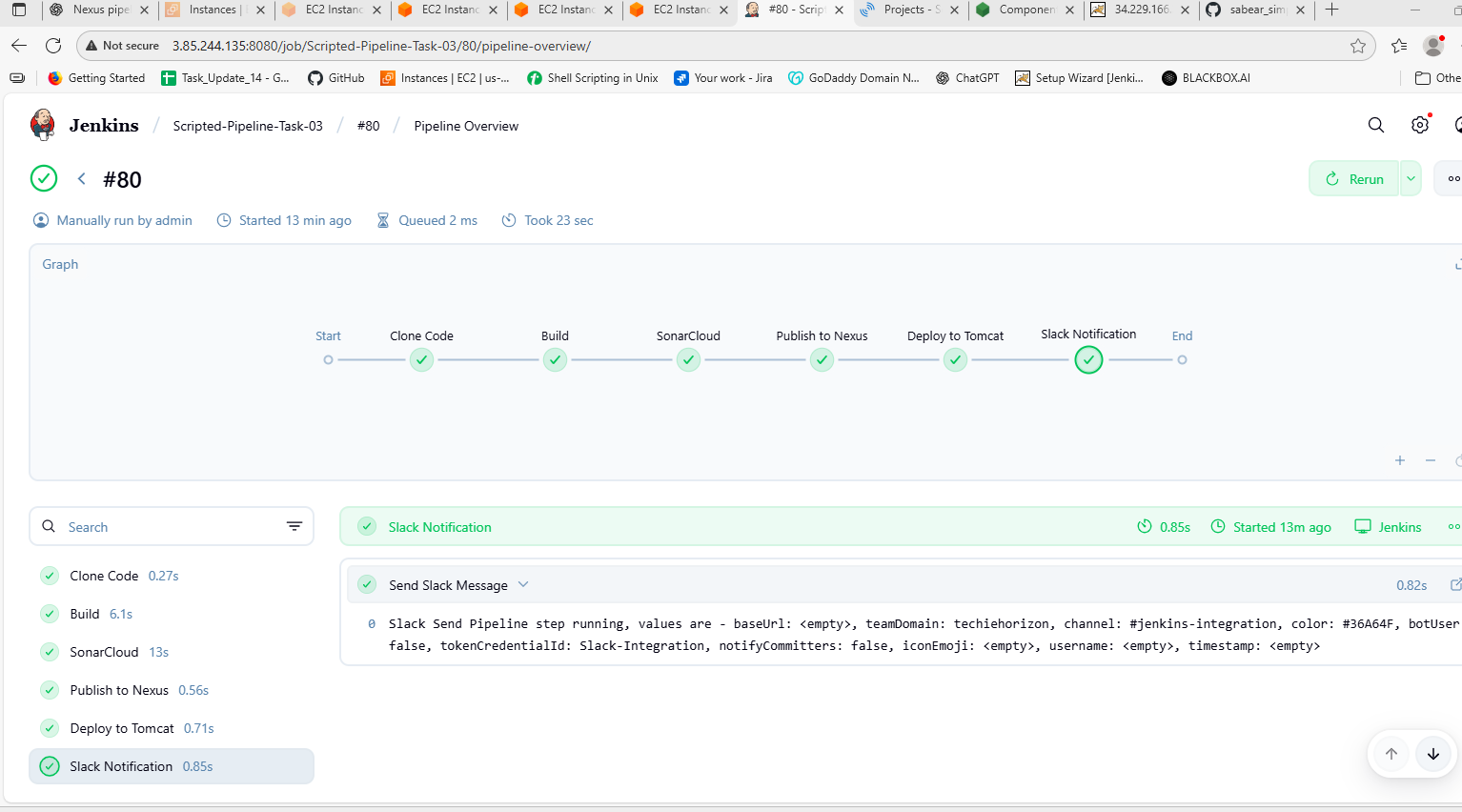
4) Nexus Artifactory

5) Slack Notification

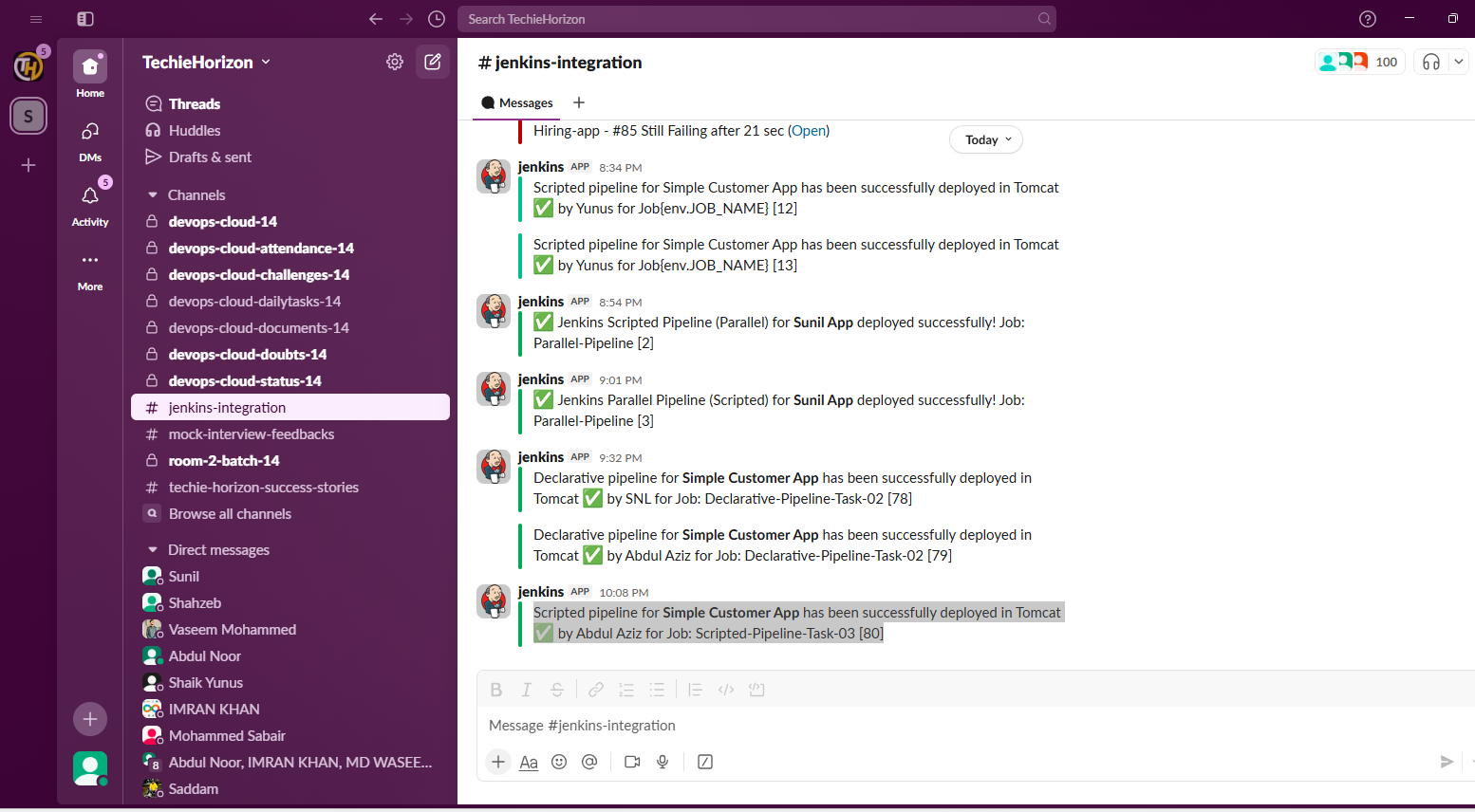
6) Deploy On tomcat

I have created a pipleline and in the we have to select the scripted pipeline





After built the job it is configuring or running the scripte one by one stages so you can see the clear picture in the above picture..



**4) Write sample skeleton of pipelines.**

A **sample skeleton of a Jenkins pipeline** in both **declarative** and **scripted** styles. This is a minimal template showing the structure, stages, and placeholders. You can fill in your commands later.

**Declarative Pipeline Skeleton:**

pipeline {

agent any // Run on any available agent

// Optional parameters

parameters {

string(name: 'GIT\_BRANCH', defaultValue: 'master', description: 'Git branch to build')

string(name: 'BUILD\_VERSION', defaultValue: '1.0', description: 'Build version')

}

environment {

MAVEN\_HOME = '/opt/maven'

JAVA\_HOME = '/usr/lib/jvm/java-17-amazon-corretto.x86\_64'

}

stages {

stage('Checkout') {

steps {

git branch: "${params.GIT\_BRANCH}", url: 'https://github.com/example/repo.git'

}

}

stage('Build') {

steps {

sh "${MAVEN\_HOME}/bin/mvn clean install -Dversion=${params.BUILD\_VERSION}"

}

}

stage('Test') {

steps {

sh "${MAVEN\_HOME}/bin/mvn test"

}

}

stage('Deploy') {

steps {

echo "Deploying application..."

// Add deploy commands here

}

}

}

post {

success {

echo "Pipeline succeeded!"

}

failure {

echo "Pipeline failed!"

}

}

}

**Scripted Pipeline Skeleton:**

node {

// Define tools

def mvnHome = tool name: 'MVN\_HOME', type: 'maven'

def javaHome = tool name: 'JAVA\_HOME', type: 'jdk'

// Parameters

def branch = params.GIT\_BRANCH ?: 'master'

def version = params.BUILD\_VERSION ?: '1.0'

stage('Checkout') {

git branch: branch, url: 'https://github.com/example/repo.git'

}

stage('Build') {

sh "${mvnHome}/bin/mvn clean install -Dversion=${version}"

}

stage('Test') {

sh "${mvnHome}/bin/mvn test"

}

stage('Deploy') {

echo "Deploying application..."

// Add deploy commands here

}

stage('Notification') {

echo "Send notification here if needed"

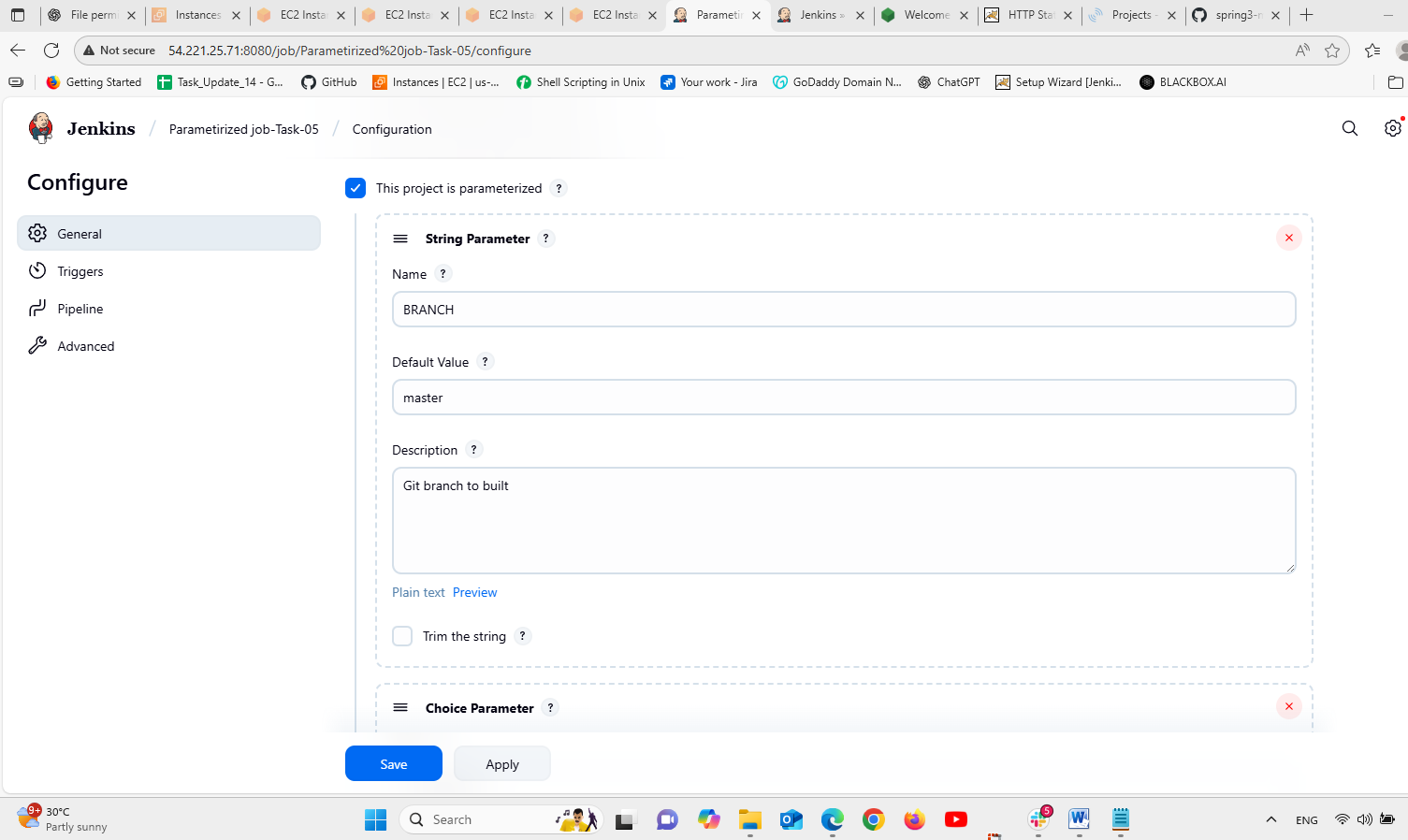
}

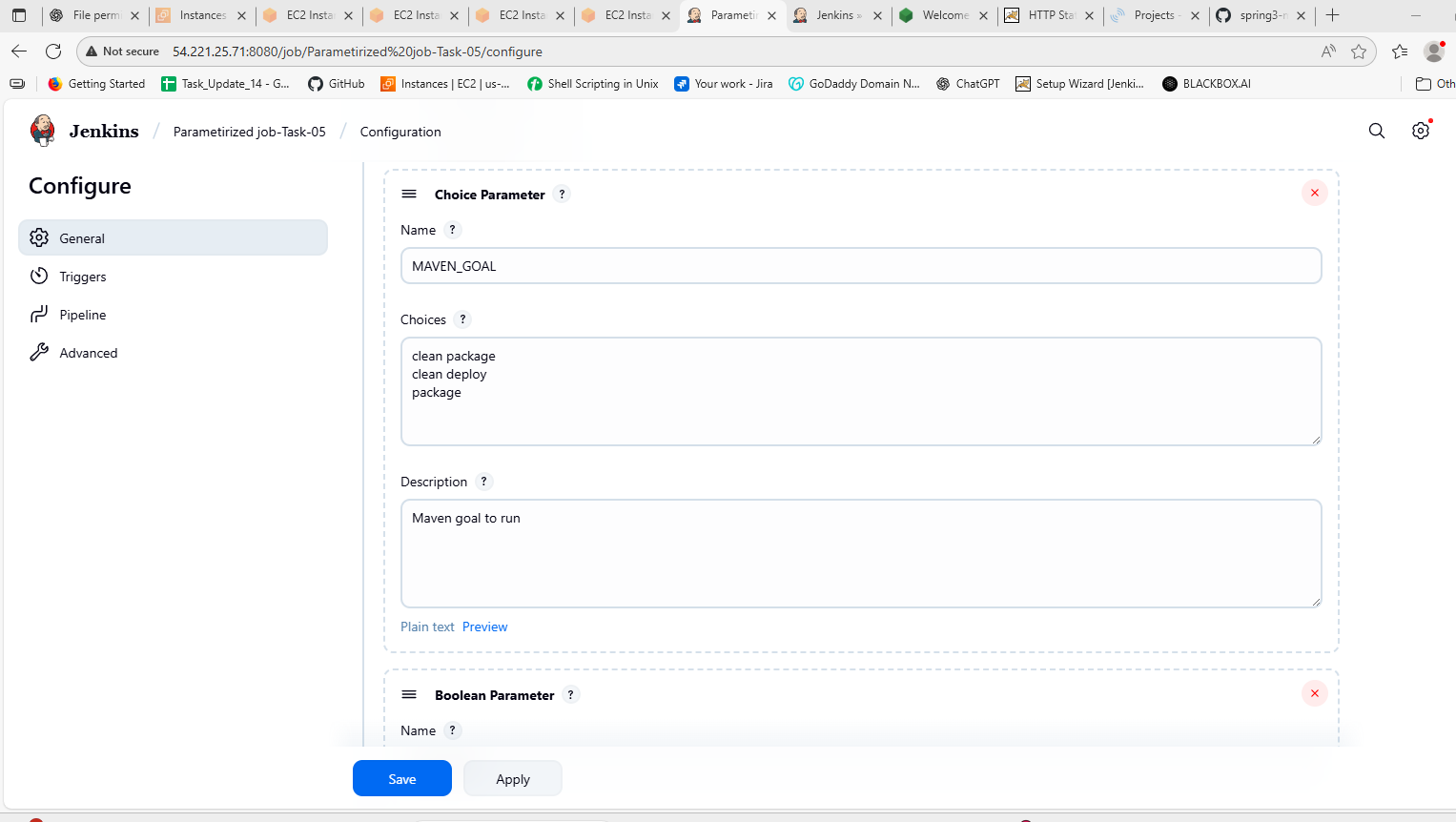
}

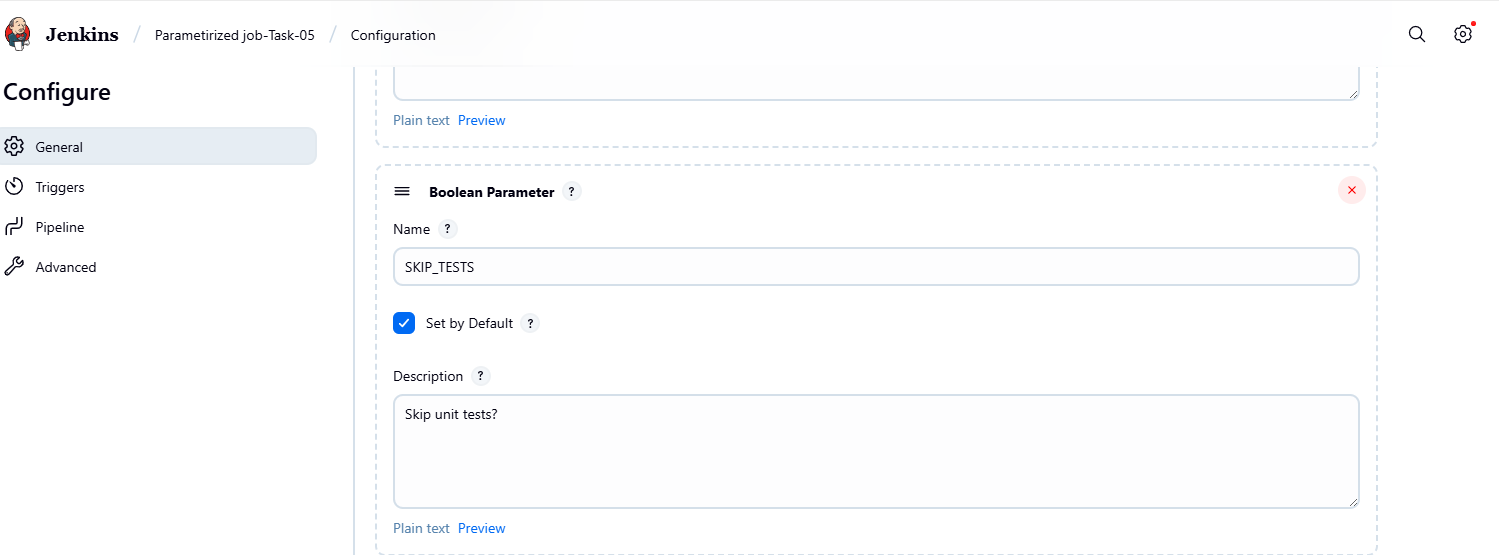
**5) Create a parametirized job in jenkins.**

<https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git>

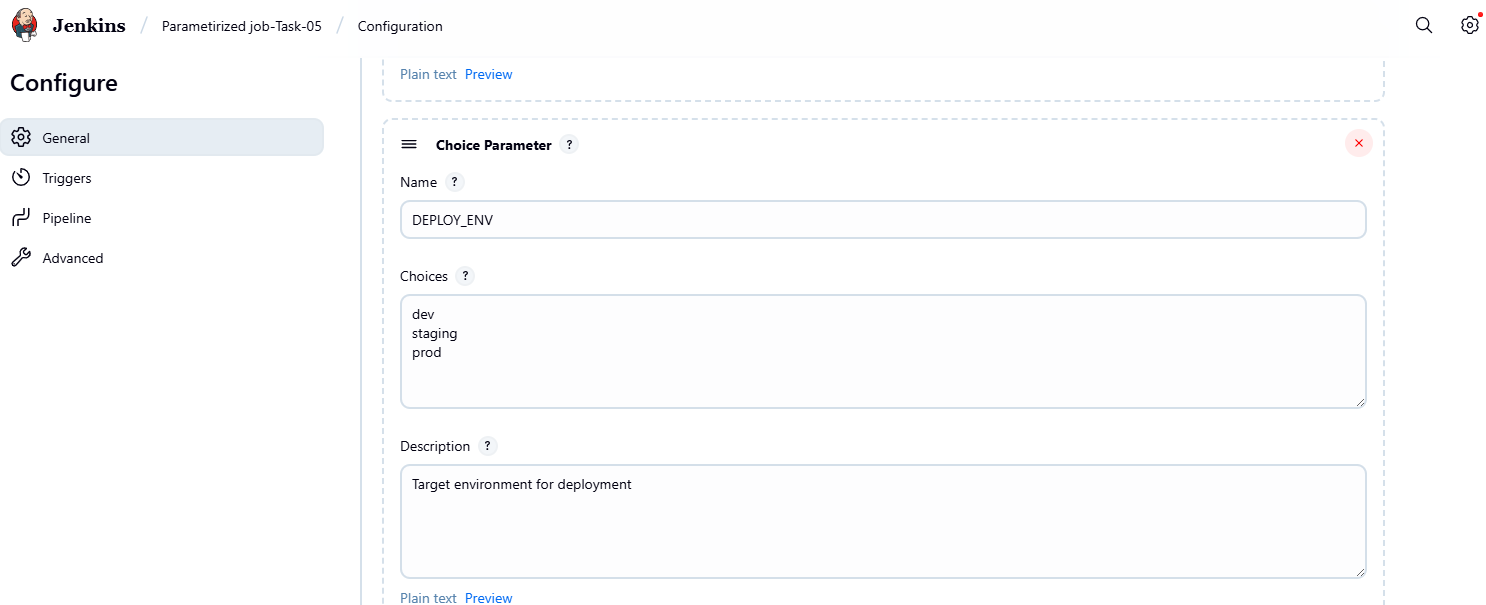
First we have to create a name : Paramiteriased job then click on This project is parameterized then add the different parameters given below

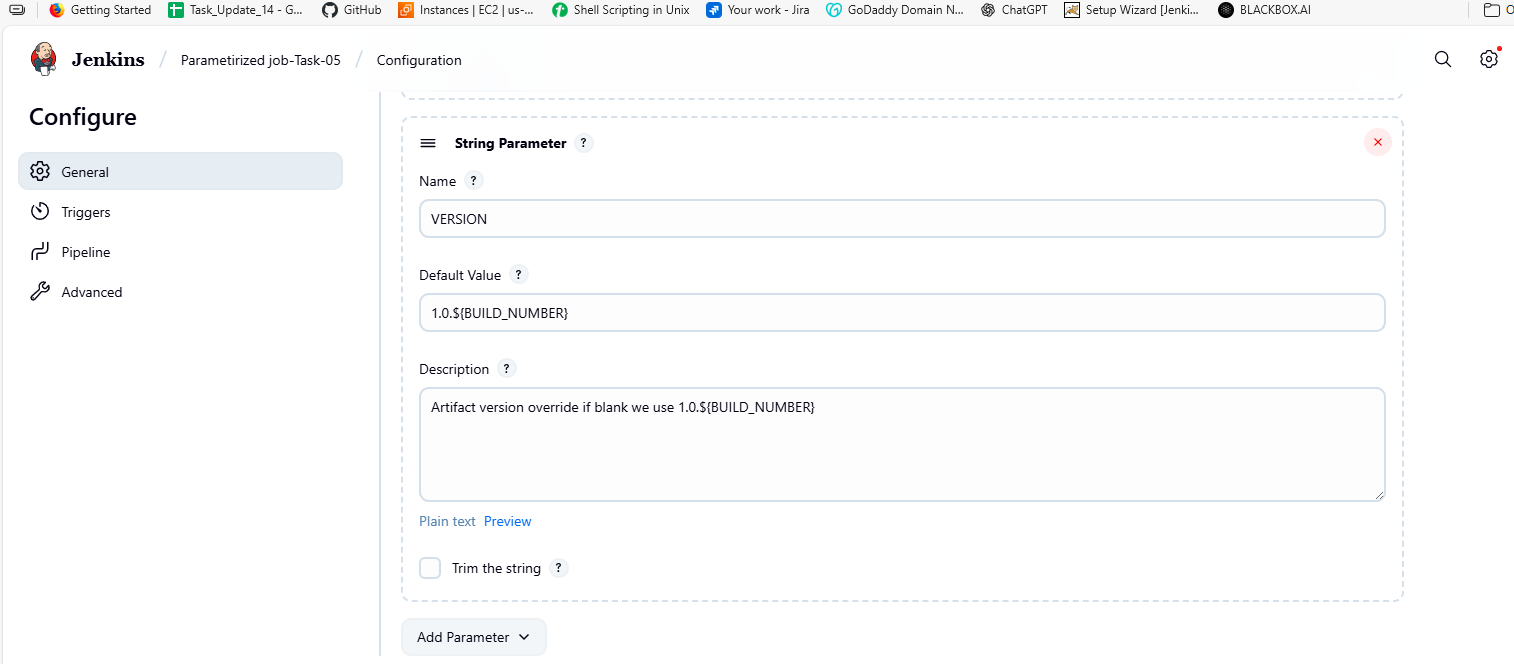












Then save it and go to pipeline and select scripted then paste the script given below

pipeline {

agent any

tools {

maven "MVN\_HOME"

}

environment {

NEXUS\_URL = "52.91.90.35:8081"

NEXUS\_REPOSITORY = "Abdul"

NEXUS\_CREDENTIAL\_ID = "My-Nexus"

TOMCAT\_USER = "deployer" // Tomcat manager username

TOMCAT\_PASSWORD = "deployer" // Tomcat manager password

TOMCAT\_HOST = "184.72.143.25" // Tomcat EC2 public IP

TOMCAT\_PORT = "8080"

SLACK\_CHANNEL = "#jenkins-integration"

SLACK\_CREDENTIAL\_ID = "slack\_notification"

}

stages {

stage("Clone code") {

steps {

git 'https://github.com/Abdulaziz920/spring3-mvc-maven-xml-hello-world-1.git'

}

}

stage("Maven build") {

steps {

sh 'mvn -B -Dmaven.test.failure.ignore=true clean install'

}

}

stage('Publish to Nexus') {

steps {

script {

def pom = readMavenPom file: 'pom.xml'

def artifactVersion = pom.version

def groupId = pom.groupId

def artifactId = pom.artifactId

def warFiles = findFiles(glob: "target/${artifactId}-${artifactVersion}.war")

if (warFiles.length == 0) {

error "WAR file not found: target/${artifactId}-${artifactVersion}.war"

}

def warFile = warFiles[0].path

echo "Uploading artifact: ${warFile} (version: ${artifactVersion}) to Nexus"

nexusArtifactUploader(

artifacts: [[

artifactId: artifactId,

classifier: '',

file: warFile,

type: 'war'

], [

artifactId: artifactId,

classifier: '',

file: 'pom.xml',

type: 'pom'

]],

credentialsId: NEXUS\_CREDENTIAL\_ID,

groupId: groupId,

version: artifactVersion,

repository: NEXUS\_REPOSITORY

)

}

}

}

stage("Deploy to Tomcat") {

steps {

script {

def pom = readMavenPom file: 'pom.xml'

def artifactVersion = pom.version

def artifactId = pom.artifactId

def warFile = "target/${artifactId}-${artifactVersion}.war"

echo "Deploying ${warFile} to Tomcat at ${TOMCAT\_HOST}:${TOMCAT\_PORT}"

sh """

curl -u ${TOMCAT\_USER}:${TOMCAT\_PASSWORD} \

-T ${warFile} \

"http://${TOMCAT\_HOST}:${TOMCAT\_PORT}/manager/text/deploy?path=/${artifactId}&update=true"

"""

}

}

}

}

post {

success {

slackSend(

channel: SLACK\_CHANNEL,

color: 'good',

message: ":white\_check\_mark: Pipeline '${env.JOB\_NAME} [${env.BUILD\_NUMBER}]' completed successfully! By Abdul Aziz <${env.BUILD\_URL}|Open Build>"

)

cleanWs()

}

failure {

slackSend(

channel: SLACK\_CHANNEL,

color: 'danger',

message: ":x: Pipeline '${env.JOB\_NAME} [${env.BUILD\_NUMBER}]' failed! <${env.BUILD\_URL}|Open Build>"

)

cleanWs()

}

always {

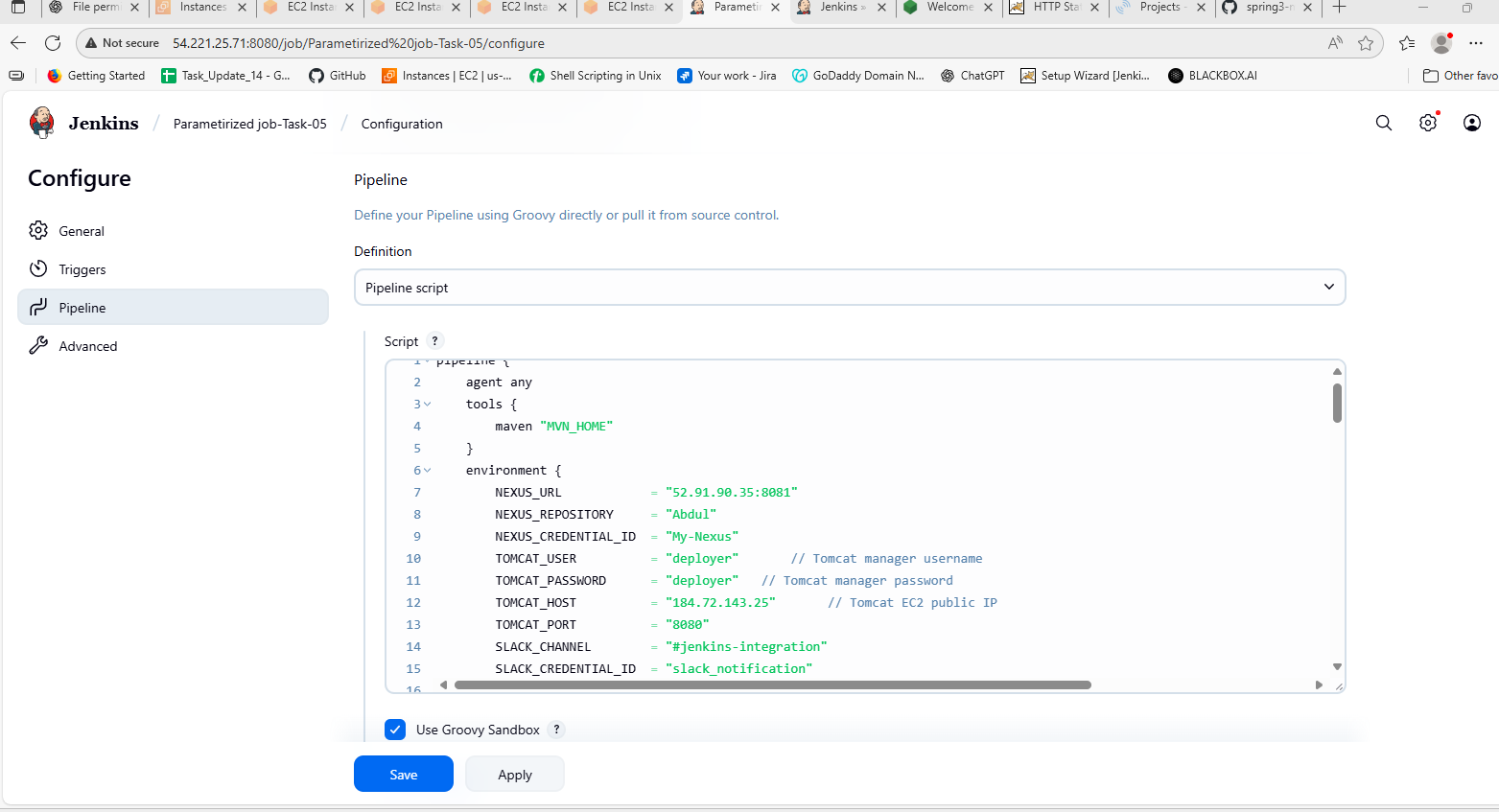
echo "Cleaning workspace..."

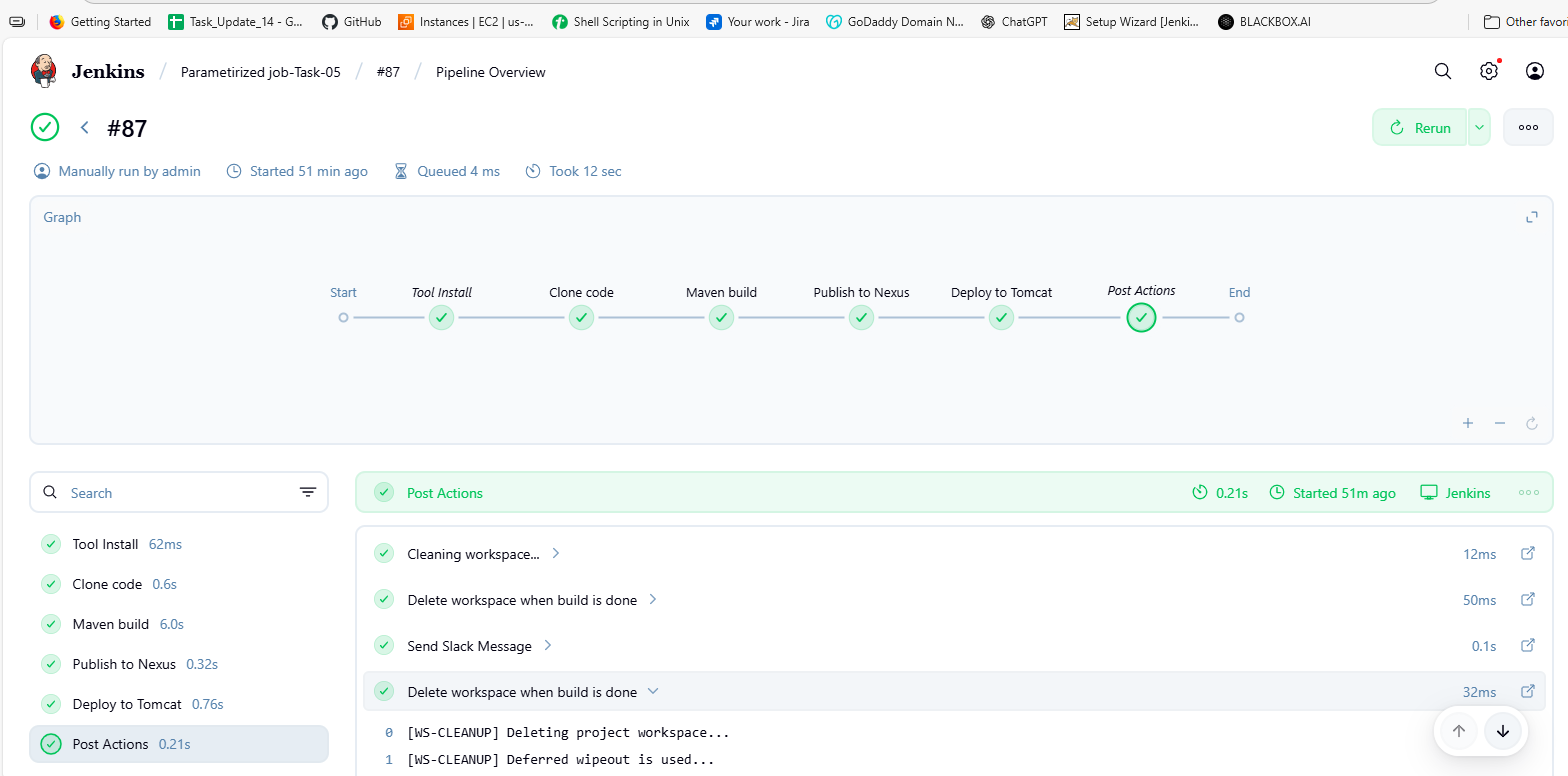
cleanWs()

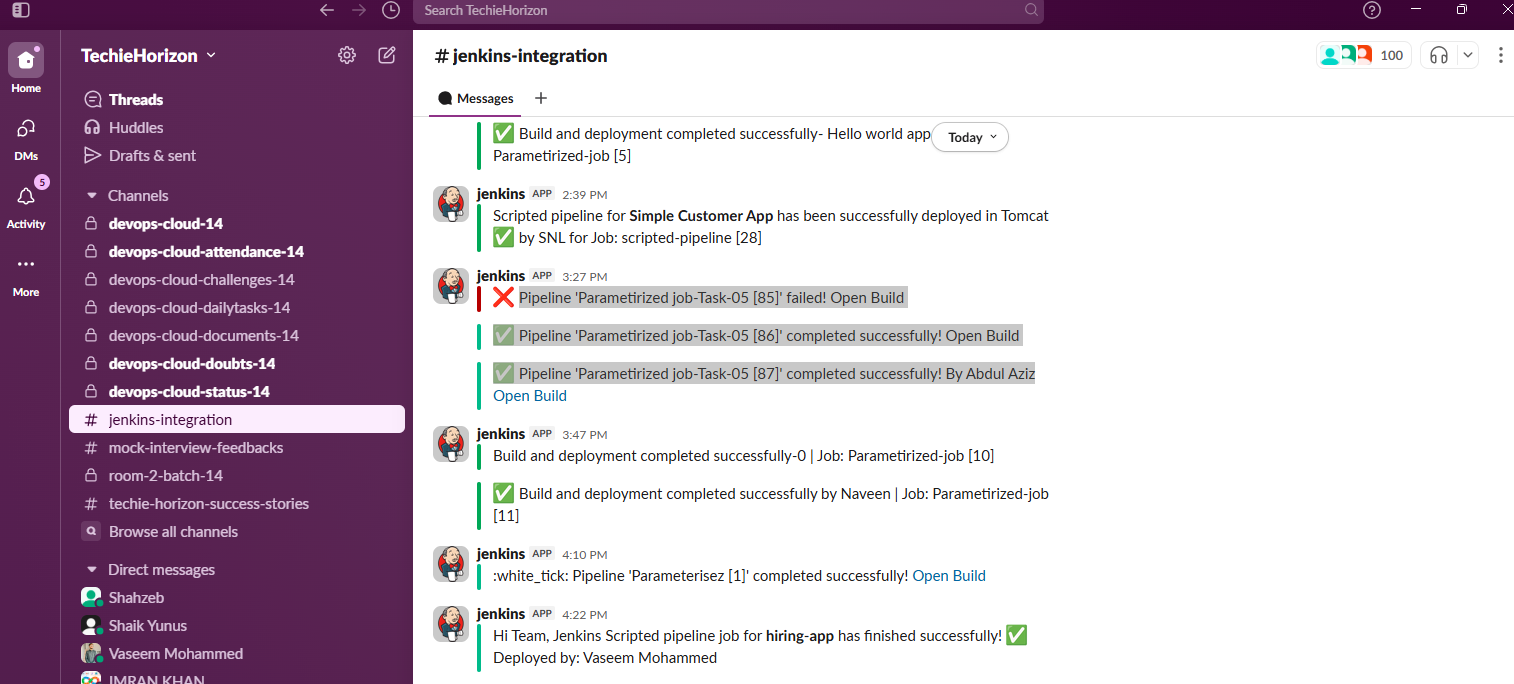
}

}

}

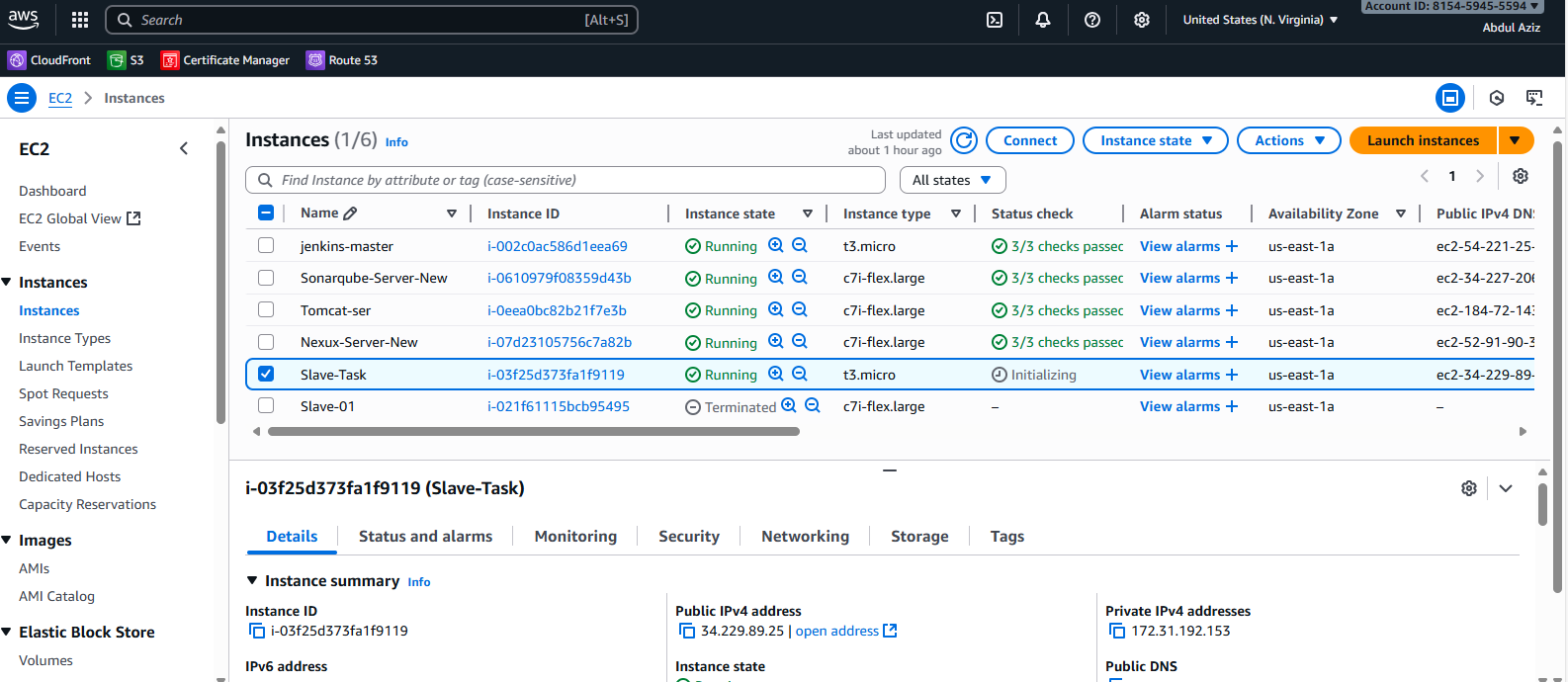






1. **Setup one slave machine for jenkins.**

First we have to create in slave machine and one master machine then install java in both and then.



Then configure everything in the slave and Jenkins given below

Slave Machine :

1. **sudo yum install -y git**
2. **sudo dnf install -y java-17-amazon-corretto**
3. **copy the key gen of slave1-ec2 → ssh-keygen**
4. **cd .ssh**
5. **cat id\_rsa.pub > authorized\_keys**
6. **chmod 700 authorized\_keys**
7. **In root → mkdir slave1-workspace**

Master Machine:

1. **Login to master machine.**
2. **Switch to root user.**
3. **Create Jenkins SSH directory:**

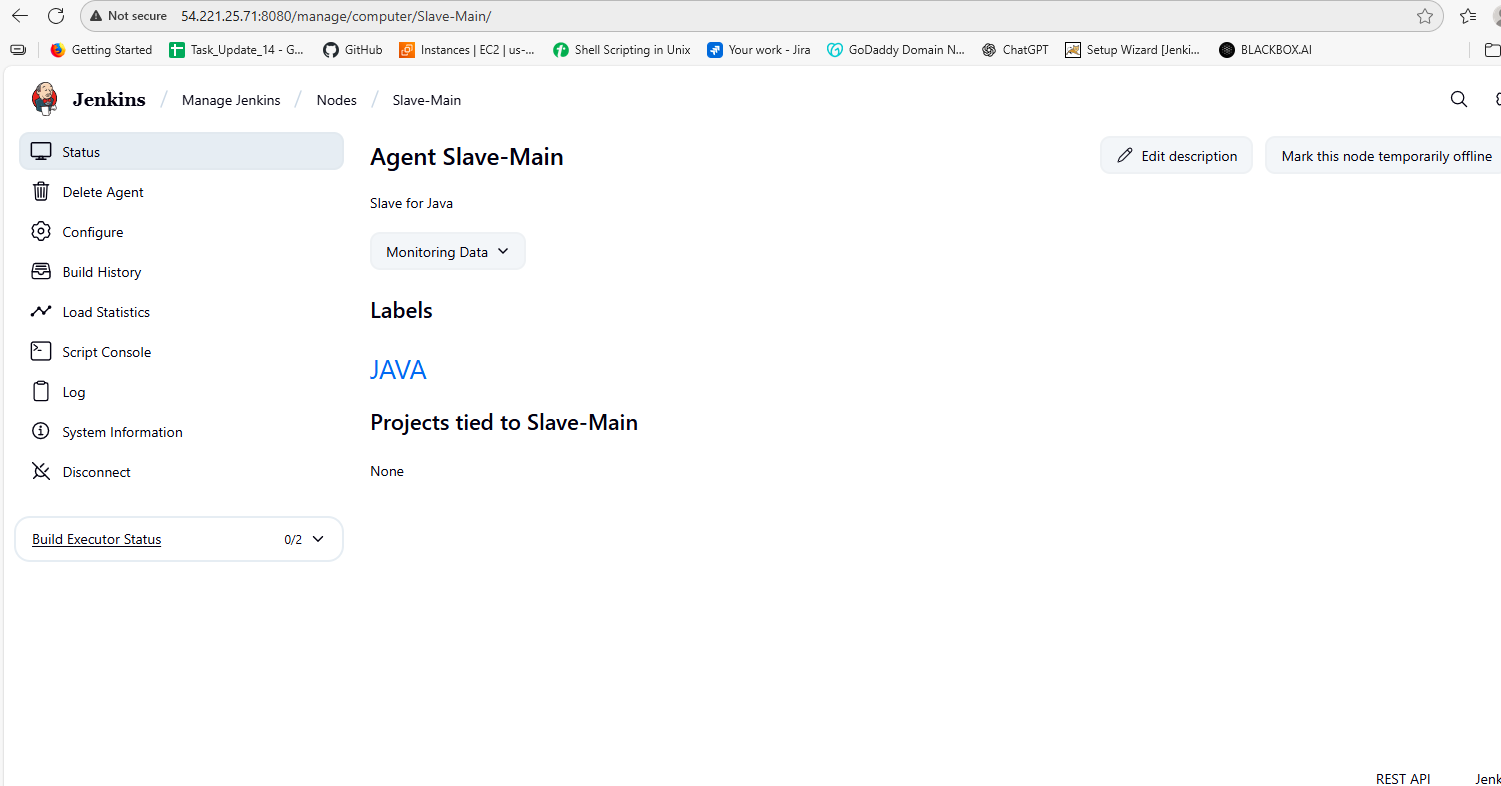
**mkdir -p /var/lib/jenkins/.ssh**

**cd /var/lib/jenkins/.ssh**

1. **ssh-keyscan -H SLAVE-NODE-PUBLIC-IP >> /var/lib/jenkins/.ssh/known\_hosts**
2. **chown jenkins:jenkins /var/lib/jenkins/.ssh/known\_hosts**
3. **chmod 644 /var/lib/jenkins/.ssh/known\_hosts**

After that we have to create a job on Jenkins As given below

1. **Manage Jenkins → Nodes → New Node**
2. **Name: slave1 → Type: Permanent Agent**
3. **Remote root directory: /home/ec2-user**
4. **This path is taken from slave machine**
5. **Slave ec2 → cat /etc/passwd**
6. **Create mkdir slave1-workspace(if already created then ignore)**
7. **In slave root → chown ec2-user:ec2-user slave1-workspace**
8. **chmod 777 slave1-workspace**
9. **Labels: slave1 java (as you like)**
10. **Usage: “Only build jobs with label expressions” (optional)**
11. **Launch method: Launch agents via SSH**
    * **Host: <NEW\_AGENT\_IP\_OR\_DNS>**
    * **Credentials: create/select “SSH Username with private key”**
      + **Username: ec2-user**
      + **Private key: paste content of /var/lib/jenkins/.ssh/id\_rsa from the master or pem key.**
    * **Host Key Verification Strategy: Known hosts file (recommended)**
12. **Save → Jenkins should connect and show Online.**



After that we have to run multiple jobs and the jobs start running on the slave machines

If the jobs are running that successful.