Jenkins-04

1. Setup Jenkins CICD pipeline using freestyle job using Docker containers using below code. <https://github.com/betawins/hiring-app.git> Stages:  
Git Clone  
SonarQube Integration  
Maven Compilation  
Nexus Artifactory  
Slack Notification  
Deploy On tomcat

TITLE:

Jenkins CICD pipeline using freestyle job using Docker containers

OBJECTIVE:

The objective of a Jenkins pipeline is to **automate the software delivery process** by defining stages such as code checkout, build, test, and deployment in a structured and repeatable manner.  
It helps achieve **Continuous Integration and Continuous Delivery (CI/CD)** by ensuring faster, reliable, and consistent application deployments using **Jenkins.**

PREREQUISITES:

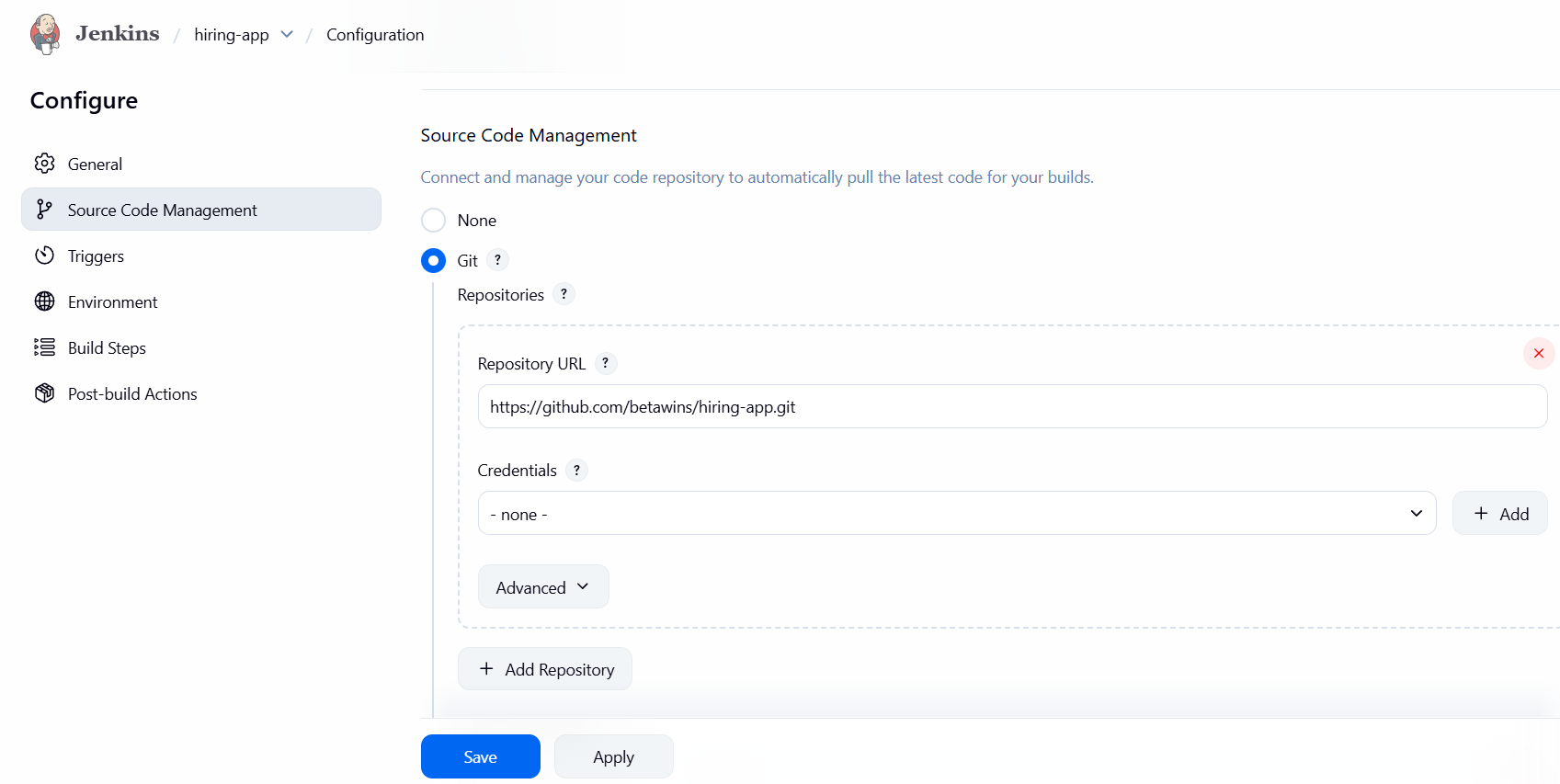
Before creating and running a Jenkins pipeline, the following prerequisites are required:

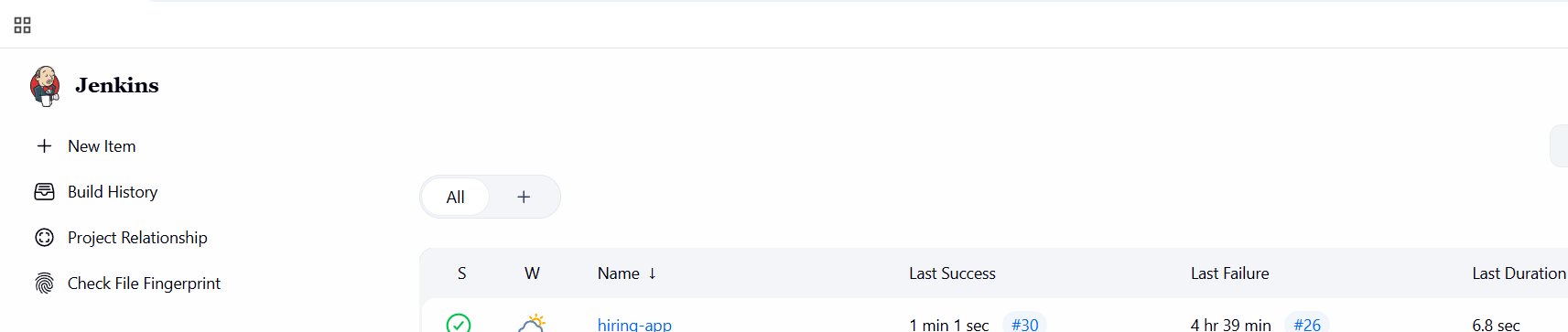
1. Jenkins server installed and running
2. Required plugins installed (Git, Pipeline, Credentials, etc.)
3. Source code repository (GitHub/GitLab/Bitbucket)
4. Build tools installed (e.g., Maven, Java, Node.js)
5. Necessary credentials configured in Jenkins
6. Access to deployment environment (server, container platform, or cloud)

STEP-BY-STEP-IMPLIMENATATION:

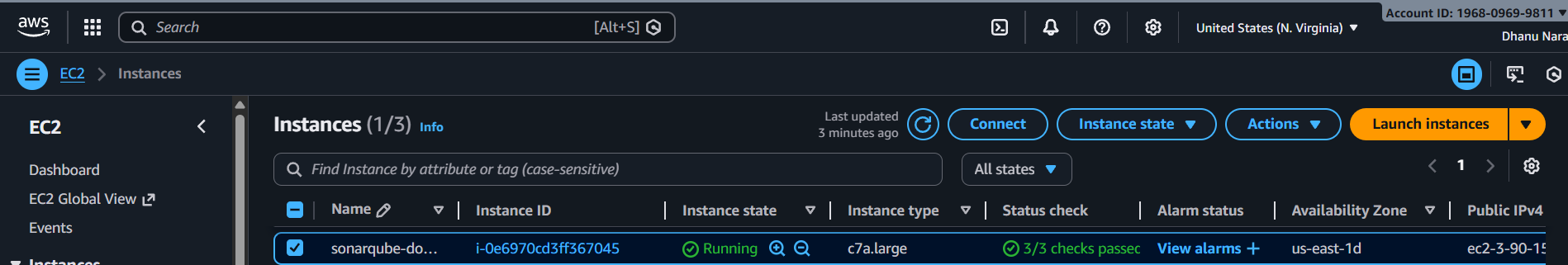
We need to launch an instance and get Jenkins access.

Create a job using github URL and clone the source code.





Now launch an instance with instance type m7large instance type and 20 GB volume



Yum install docker -y

Systemctl start docker

Systemctl enable docker

Systemctl status docker

docker pull sonarqube

docker images

docker run -itd -p 9003:9000 sonarqube

docker ps

{paste the “IP:9003” in the browser}



Now install sonar scanner in Jenkins server.

{cd /opt

wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip

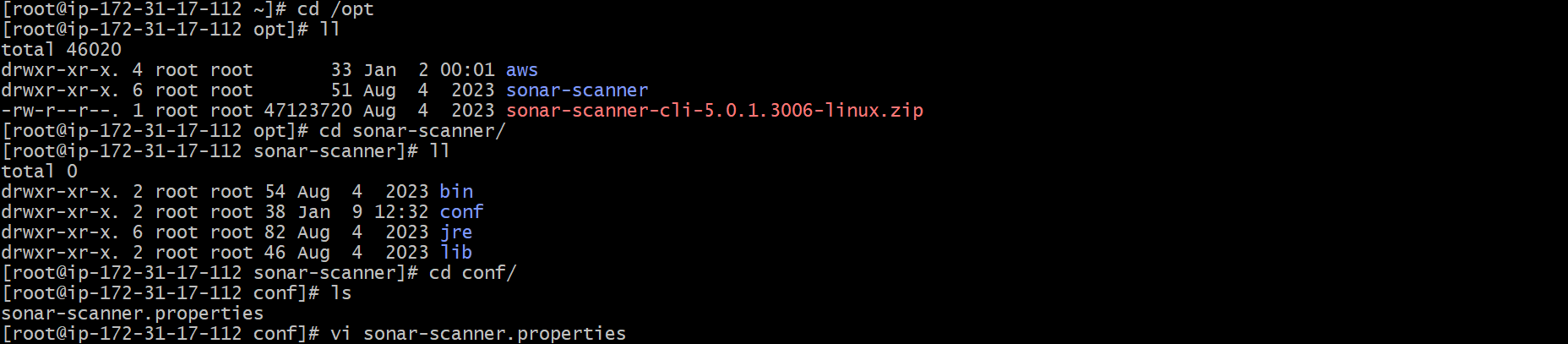
unzip sonar-scanner-cli-5.0.1.3006-linux.zip

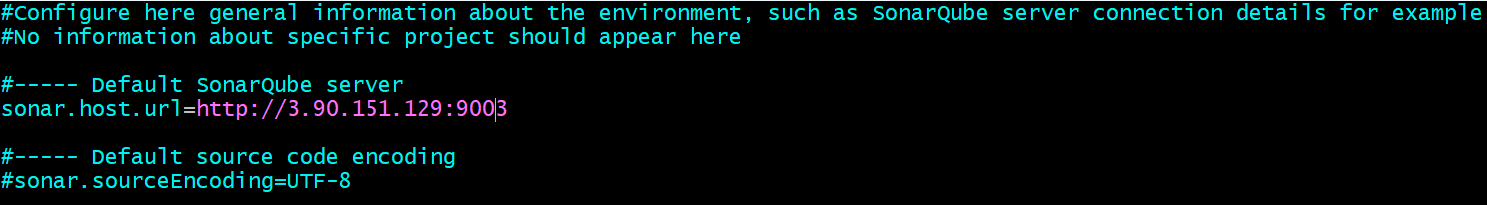
mv sonar-scanner-5.0.1.3006-linux sonar-scanner}.

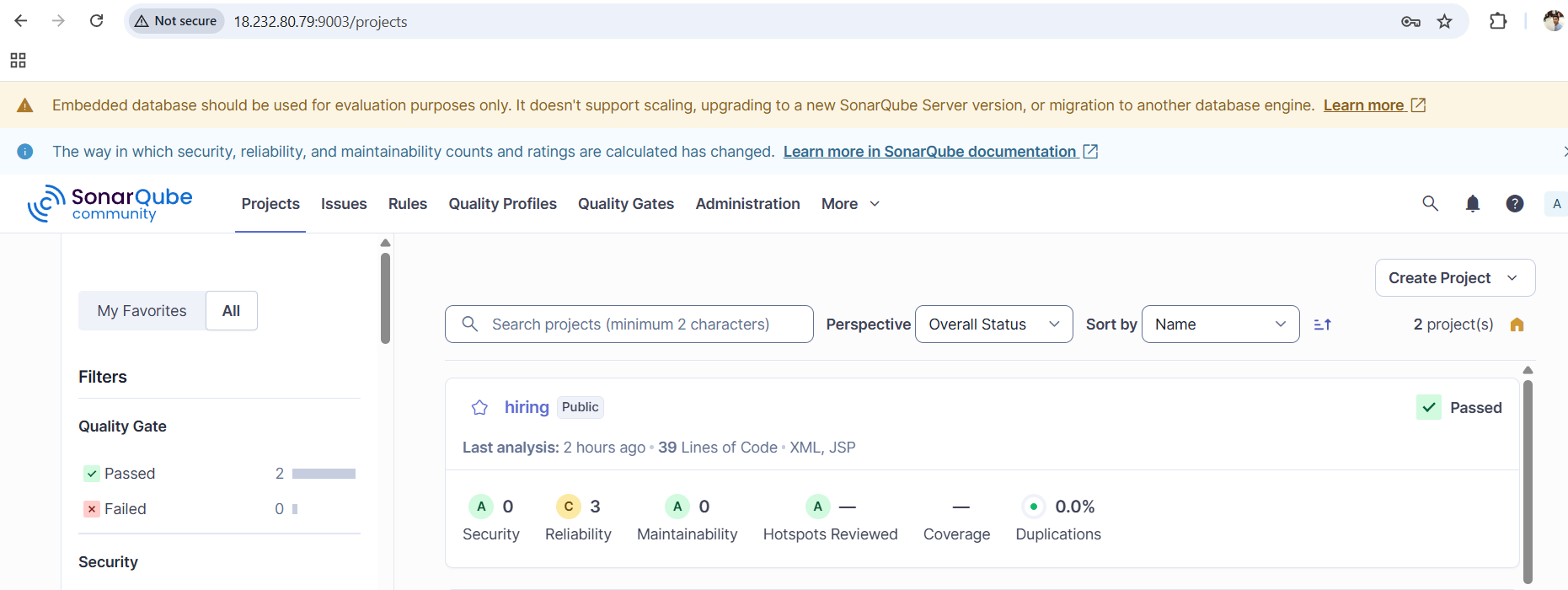
cd conf

vi sonar-scanner properties

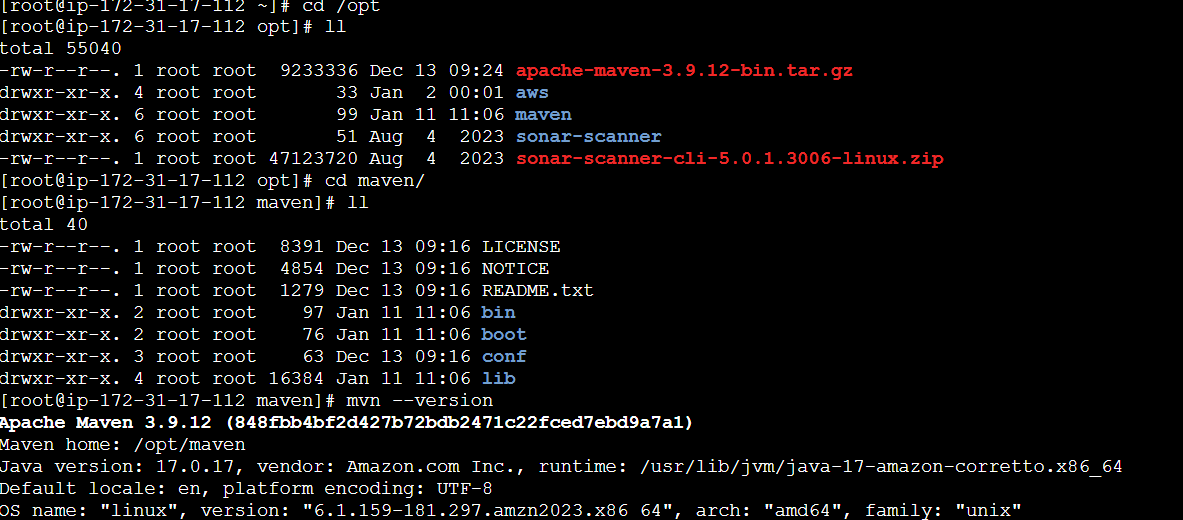
Now change to this“DockerIP:9003”.







Install maven in Jenkins server.

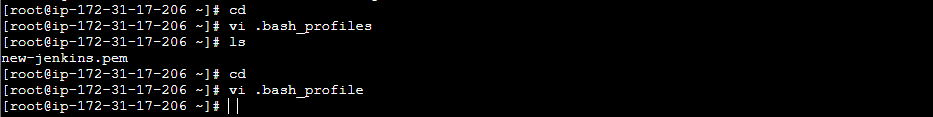


**Now we need to pass this path to global variable in order to use mvn –version at any location in cli**

**For that change directory to home directory**

**cd**

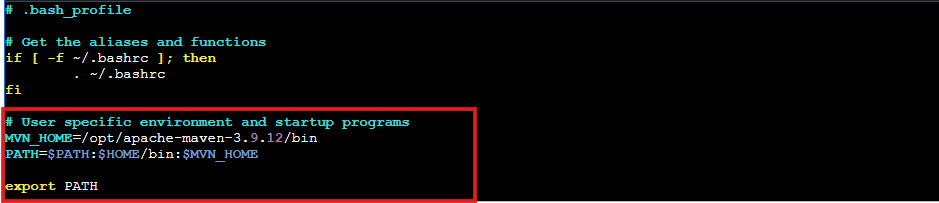
**vi .bash\_profiles 🡪 .bash\_profile : contains global variables**

****

MVN\_HOME=/opt/apache-maven-3.9.12/bin

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

export PATH

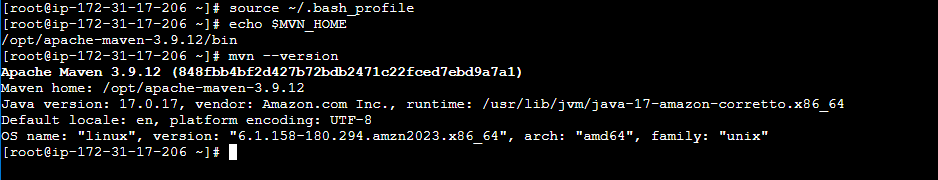
****

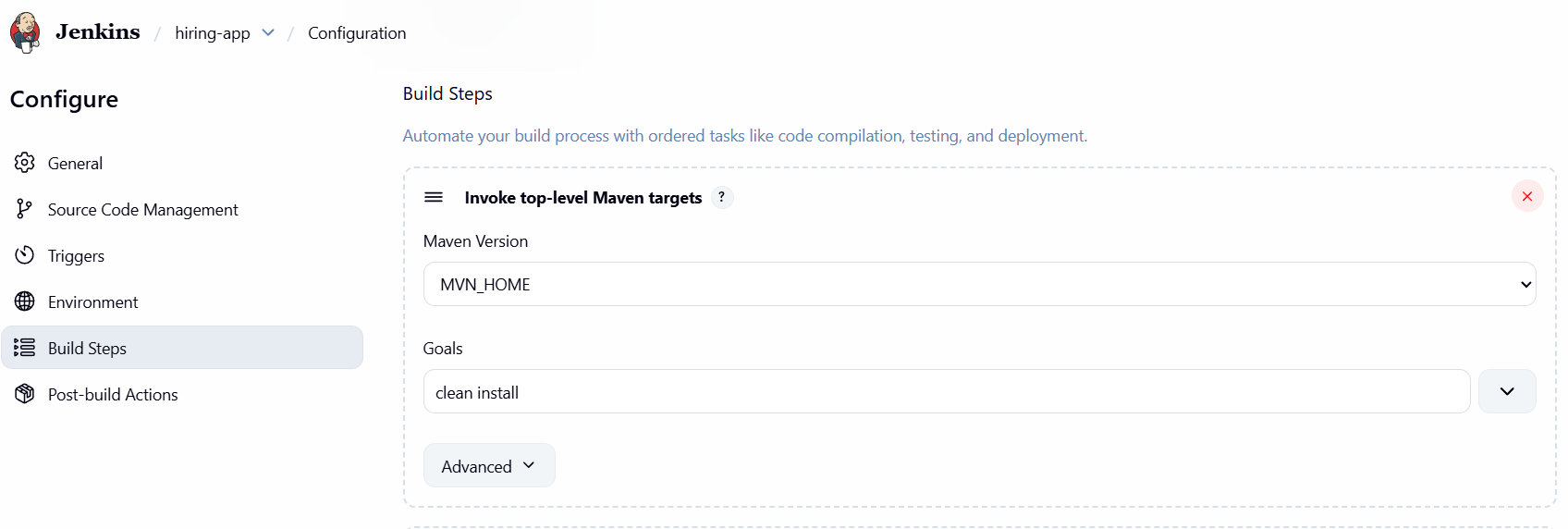
**Now we need to restart the .bash\_profile**

**source ~/.bash\_profile**

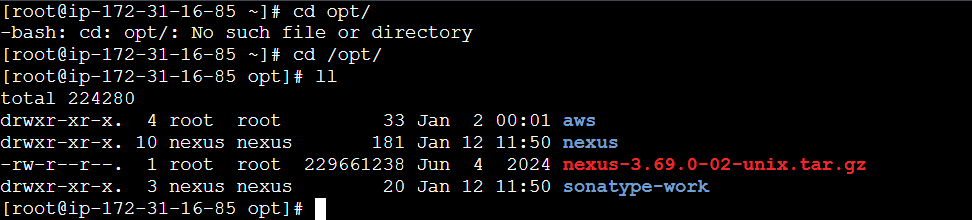
**echo $MVN\_HOME**

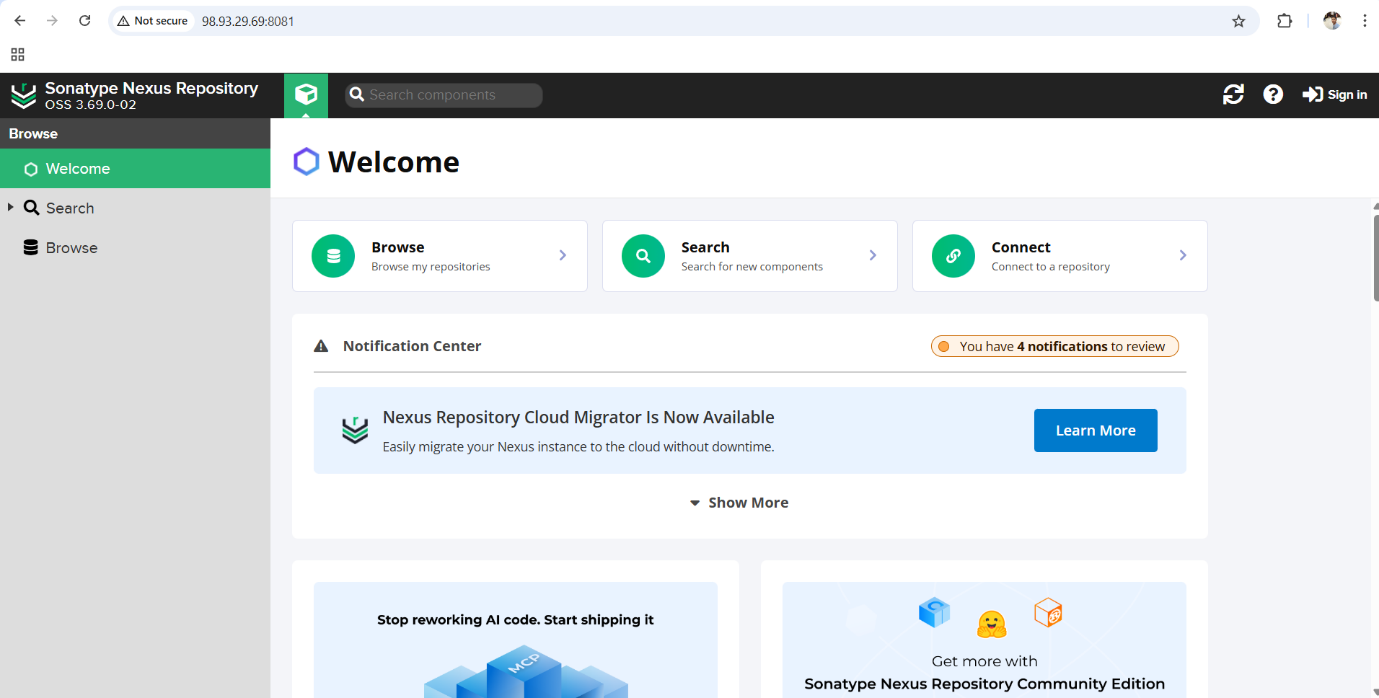
**mvn --version**

****

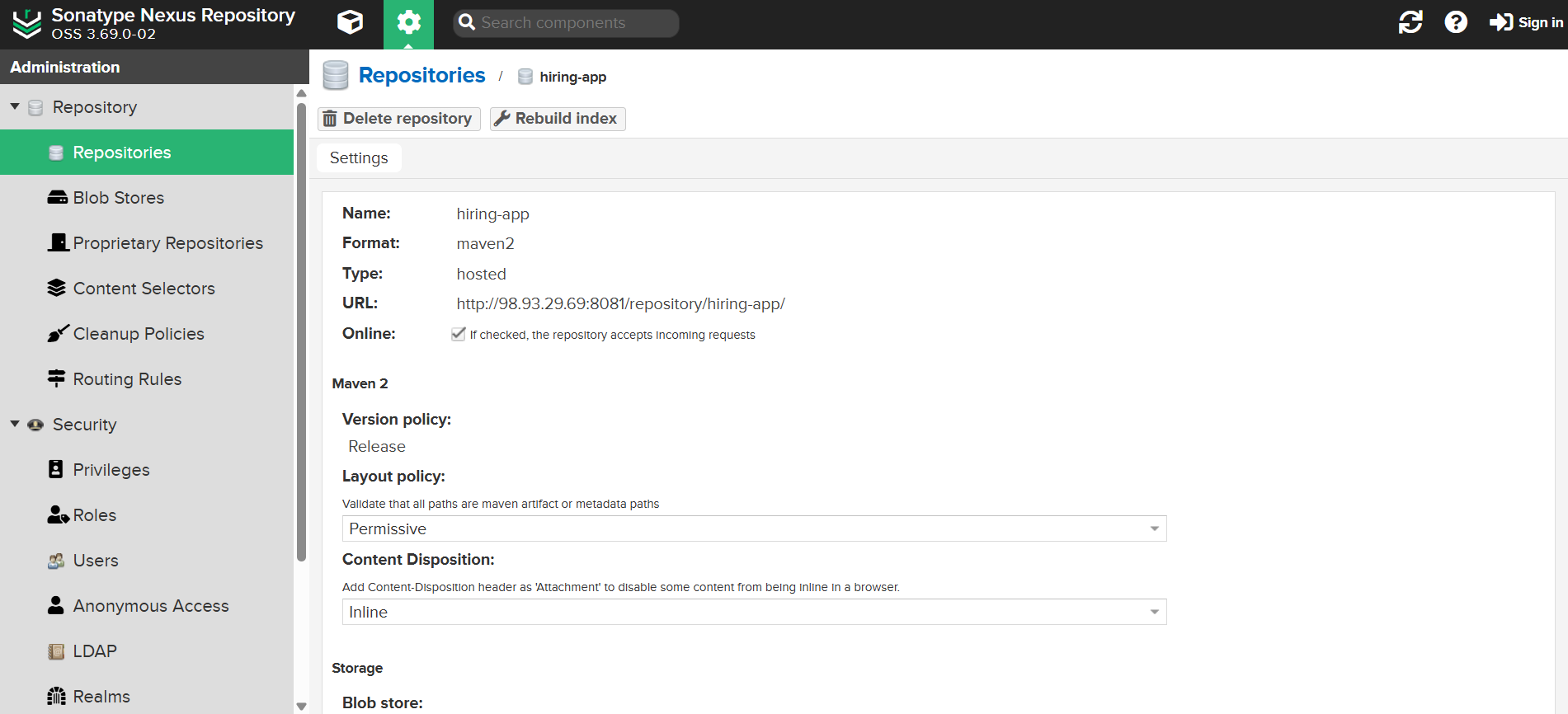


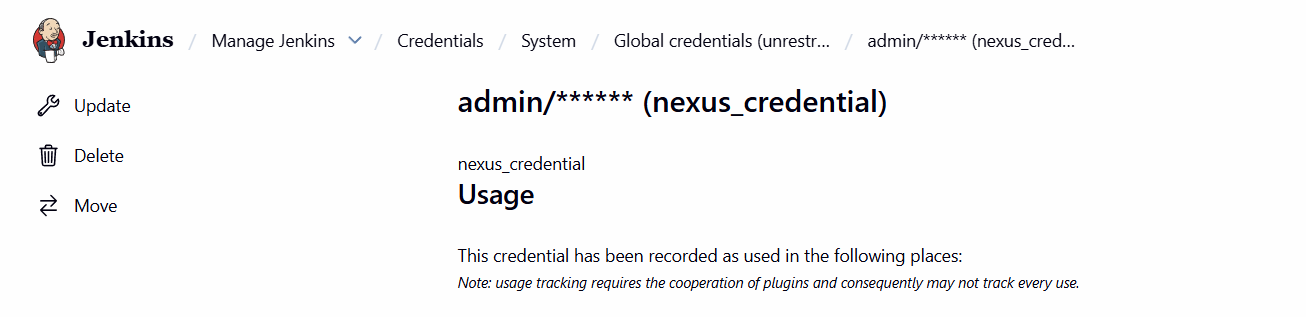
Launch new instance and install nexus in that.

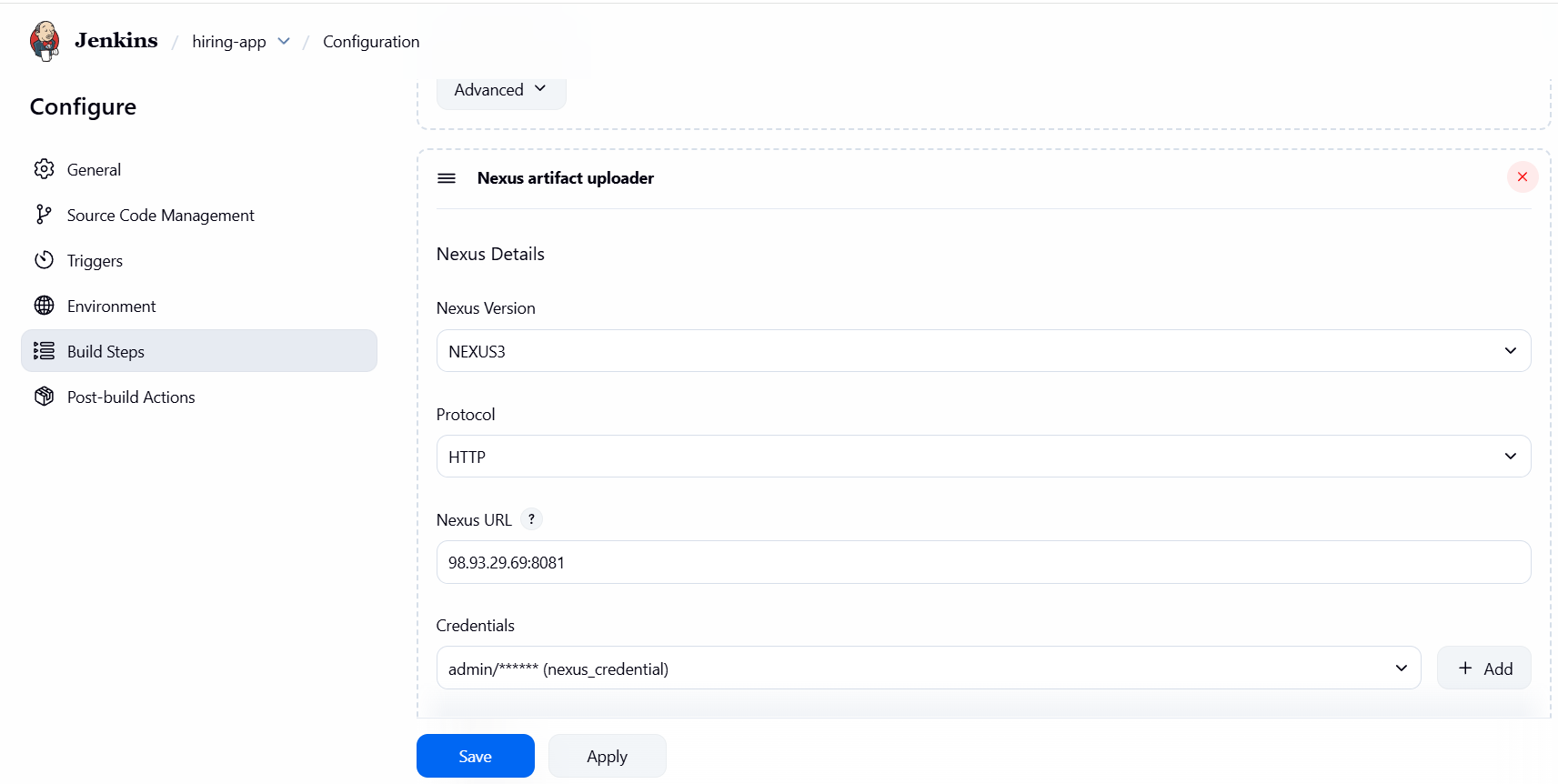




Create a repository in nexus.

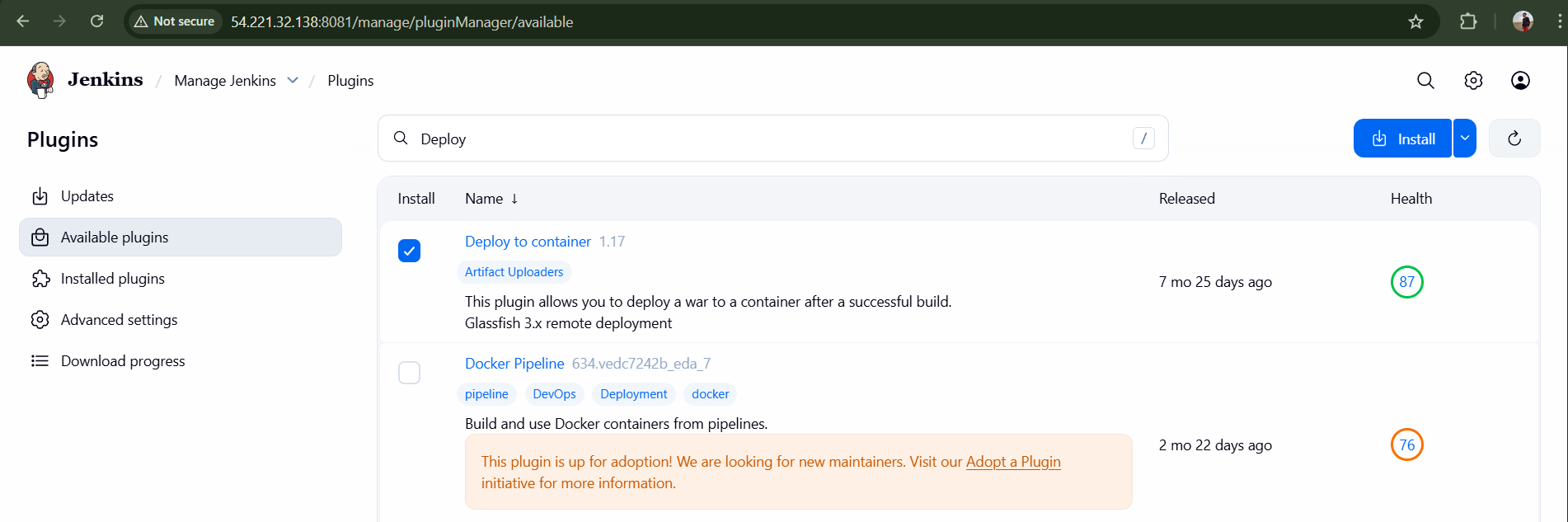


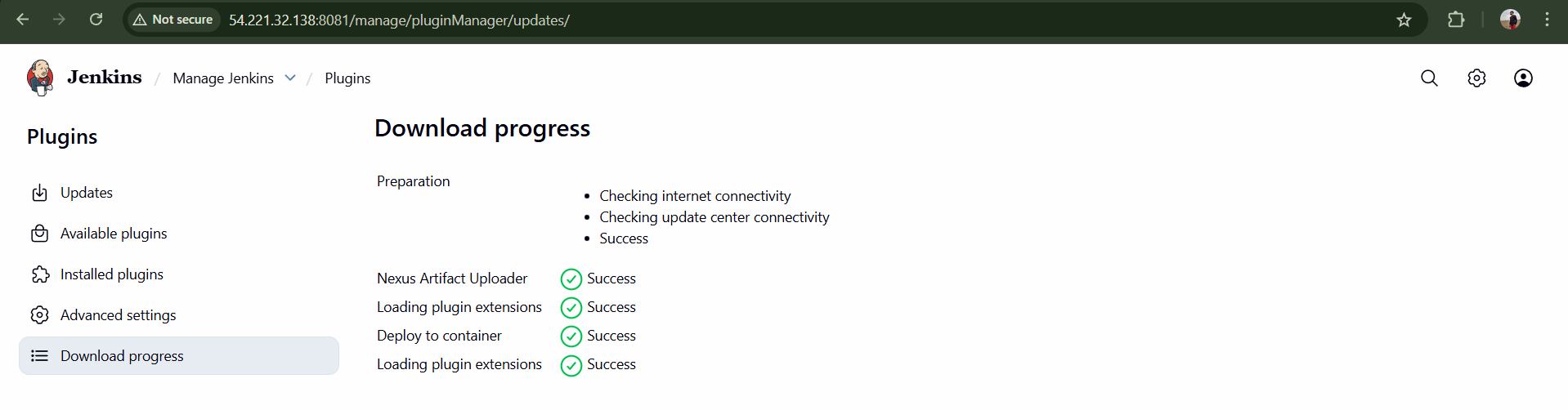




Launch new instance and install tomcat.

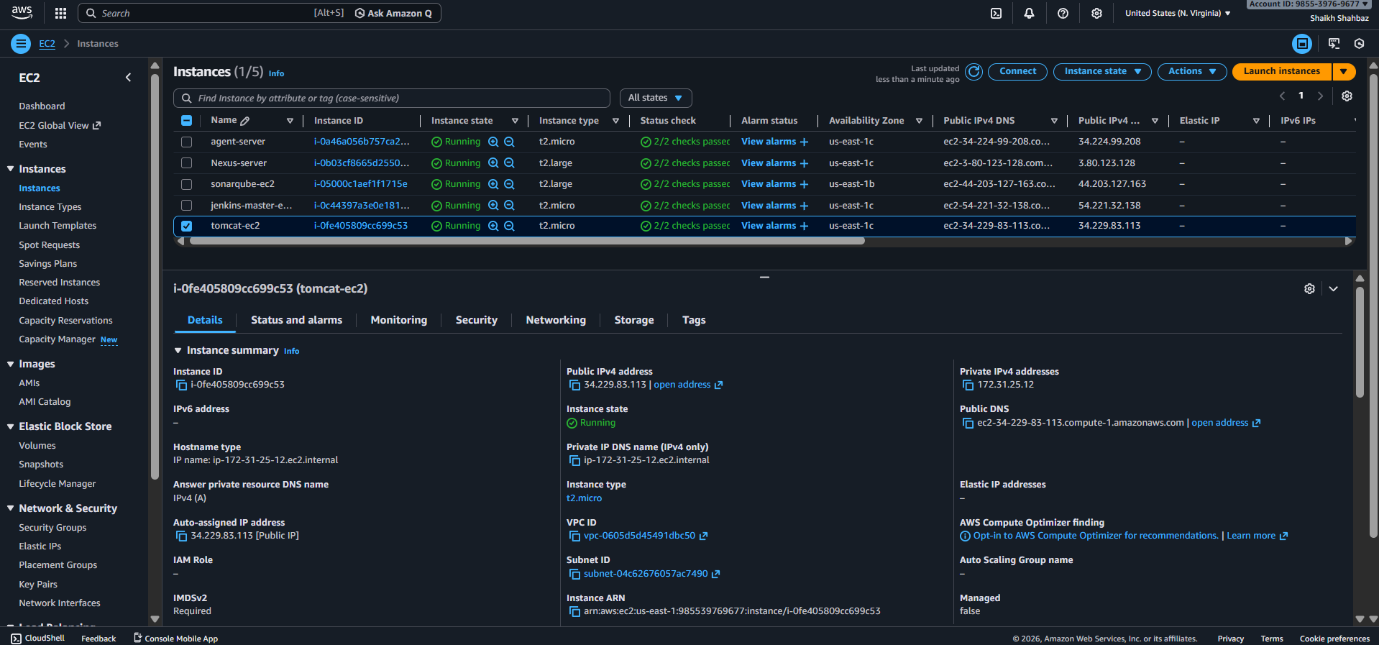
**Goto Jenkins GUI page 🡪 Manage Jenkins 🡪 Available Plugins 🡪 <Deploy to container>**

****

****

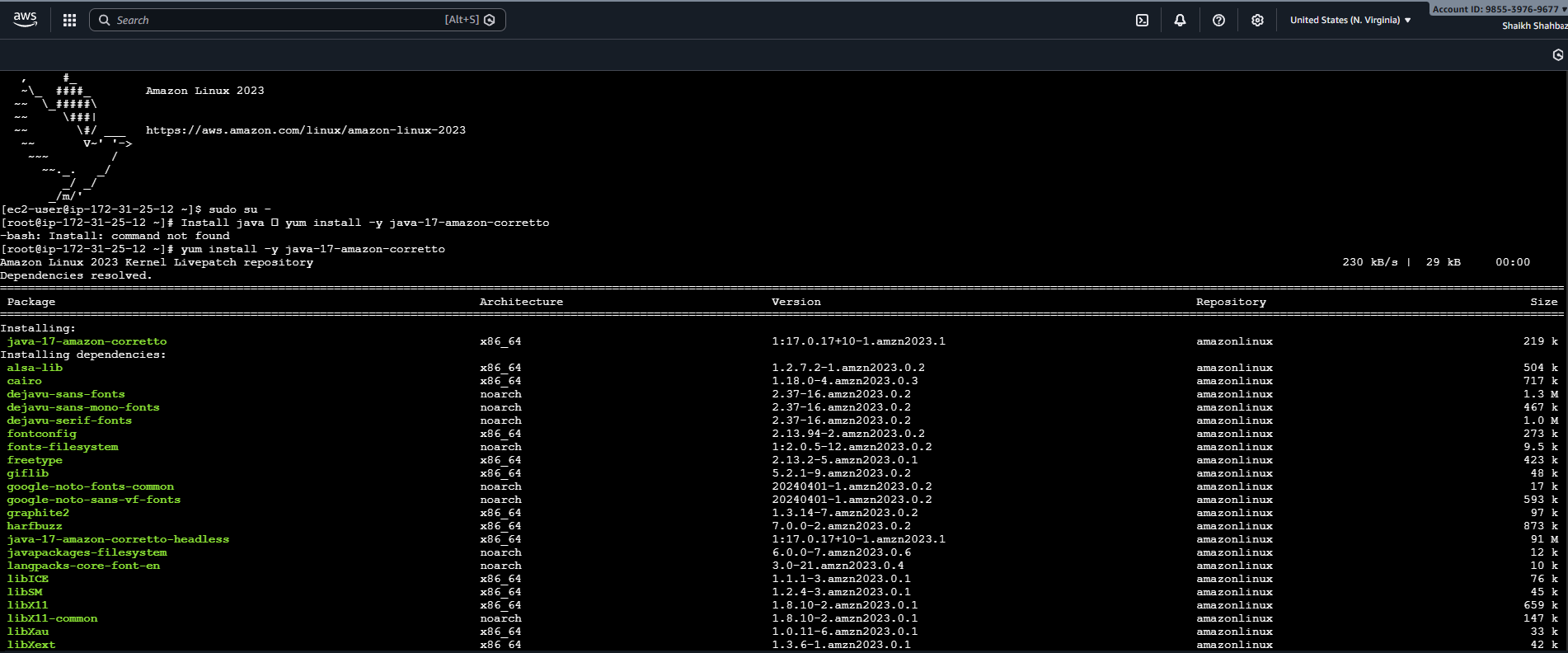
**Now Launch a new ec2 instance for tomcat deployment**

**Tomcat-ec2 <t2.micro>**

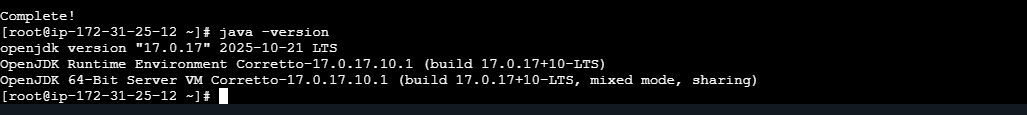
****

**Install java and install tomcat in it and start**

**Install java 🡪 yum install -y java-17-amazon-corretto**

****

**Java -version**

****

**cd /opt**

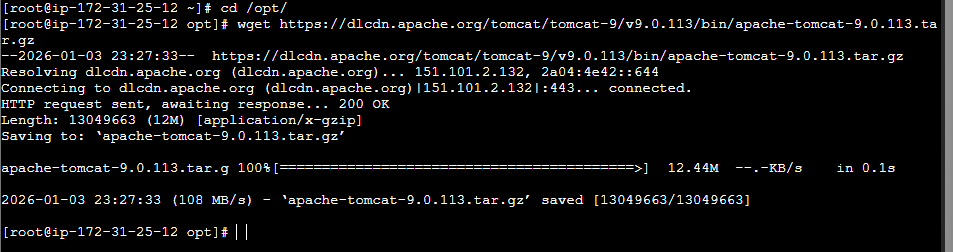
**wget** [**https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.113/bin/apache-tomcat-9.0.113.tar.gz**](https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.113/bin/apache-tomcat-9.0.113.tar.gz)

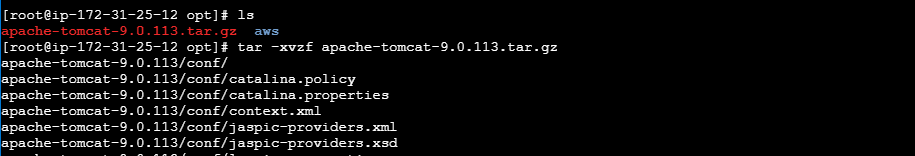
**tar -xvzf apache-tomcat-9.0.113.tar.gz**

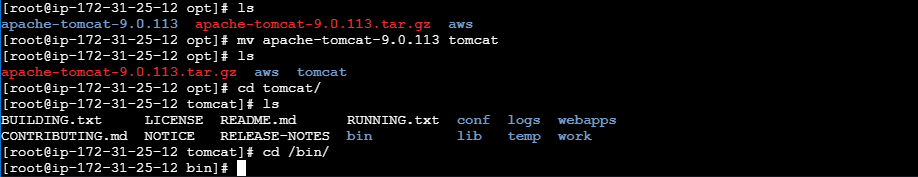
**mv apache-tomcat-9.0.113 tomcat**

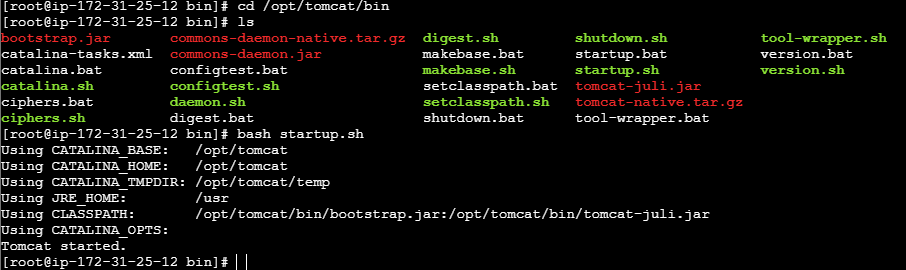
**cd /tomcat 🡪 cd /bin**

**bash startup.sh**

****

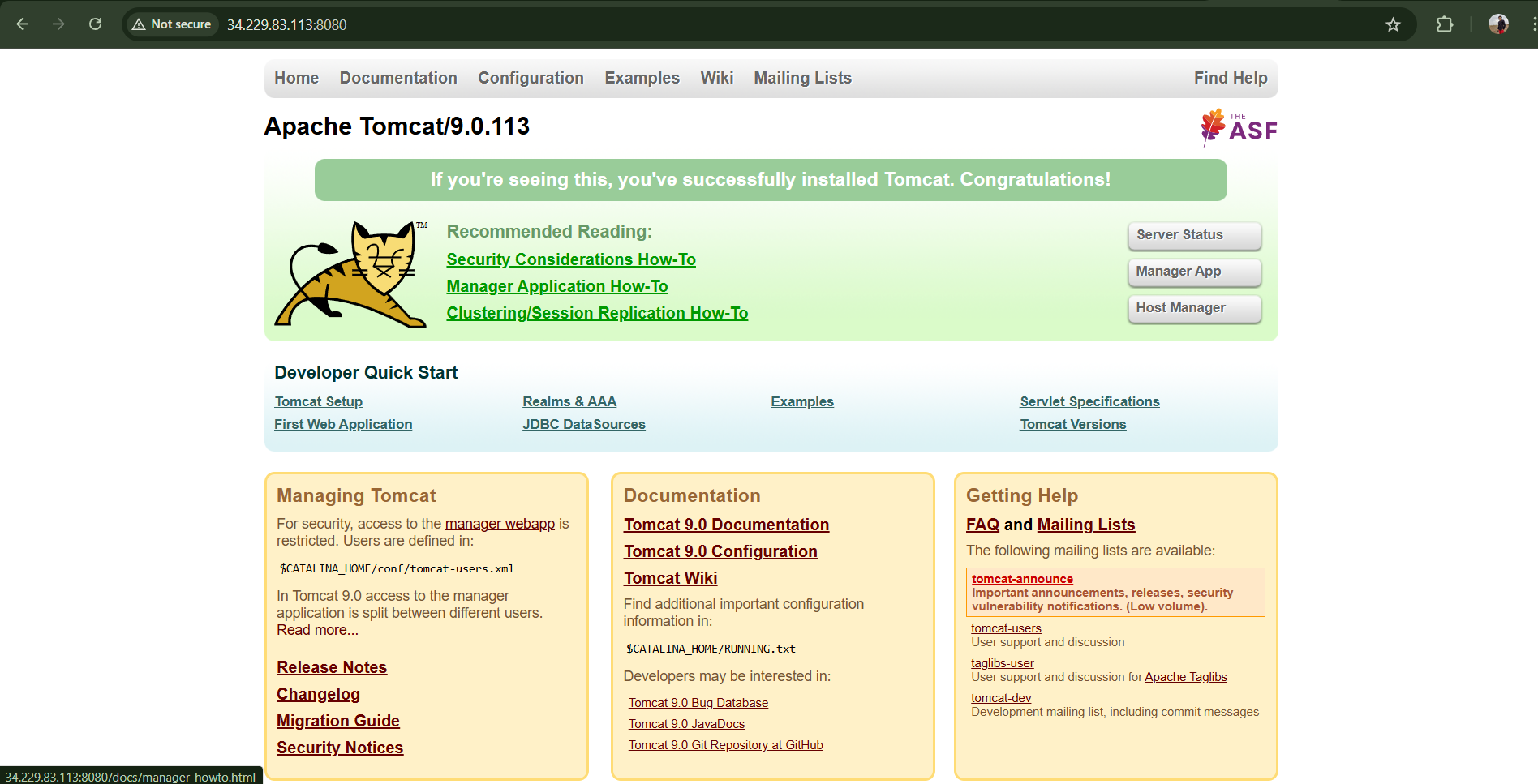
****

****

****

**Tomcat successfully started**

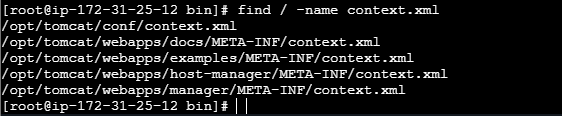
**Check on Browser with <Public-ip>:8080**

****

**Now create user in tomcat**

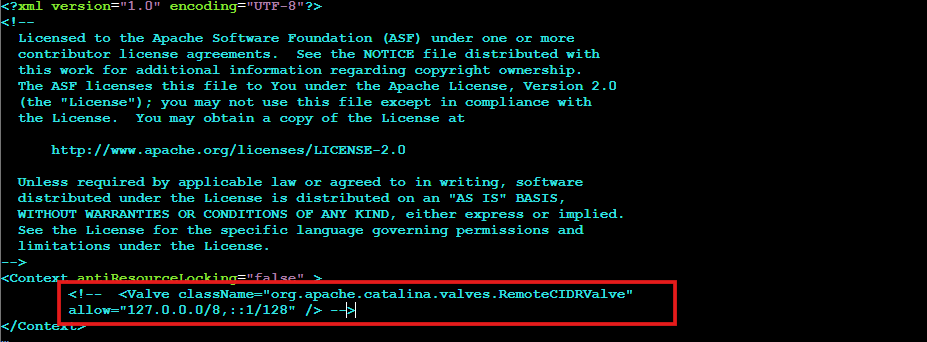
**Tacke file webapps and edit the 4-Files <uncomment>**

**find / -name context.xml**

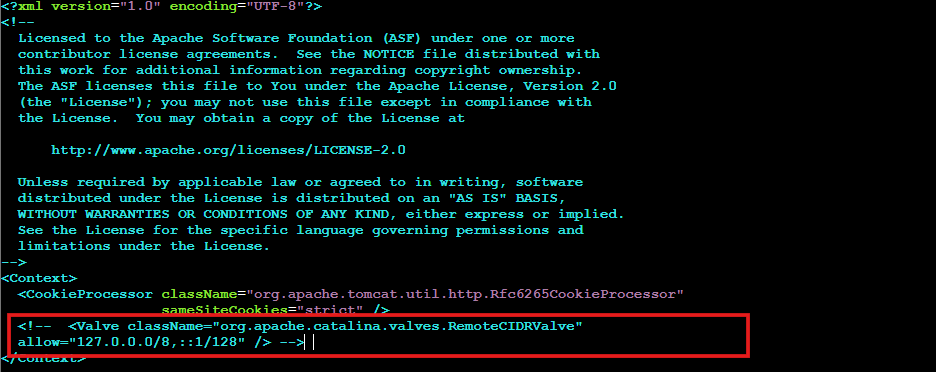
****

**Add comment <!-- & --> to above 4-files**

**vi /opt/tomcat/webapps/docs/META-INF/context.xml**

****

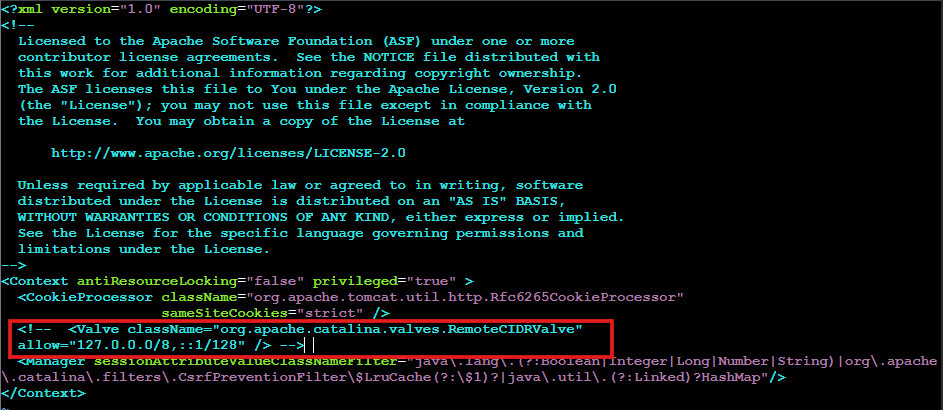
**vi /opt/tomcat/webapps/examples/META-INF/context.xml**

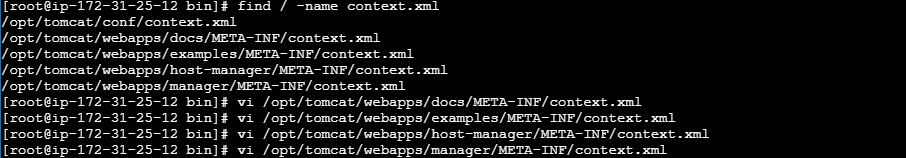
****

**vi /opt/tomcat/webapps/host-manager/META-INF/context.xml**

****

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

****

****

**Now go to conf in xml file and add the below details to add user to tomcat <tomcat-user.xml>**

**cd /opt/tomcat/conf/**

**vi tomcat-users.xml**

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

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

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

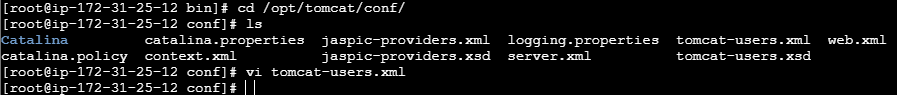
*<role rolename="manager-status"/>*

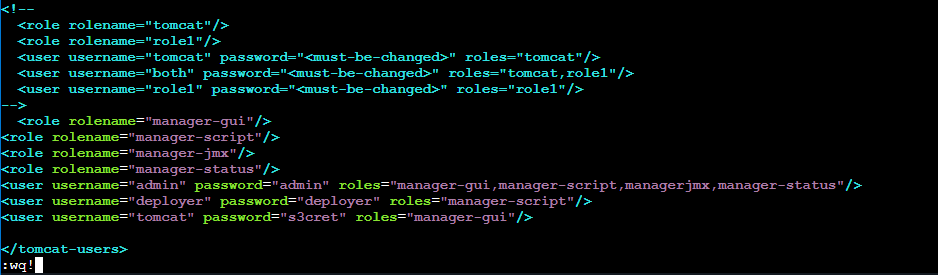
*<user username="admin" password="admin" roles="manager-gui,manager-script,managerjmx,manager-status"/>*

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

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

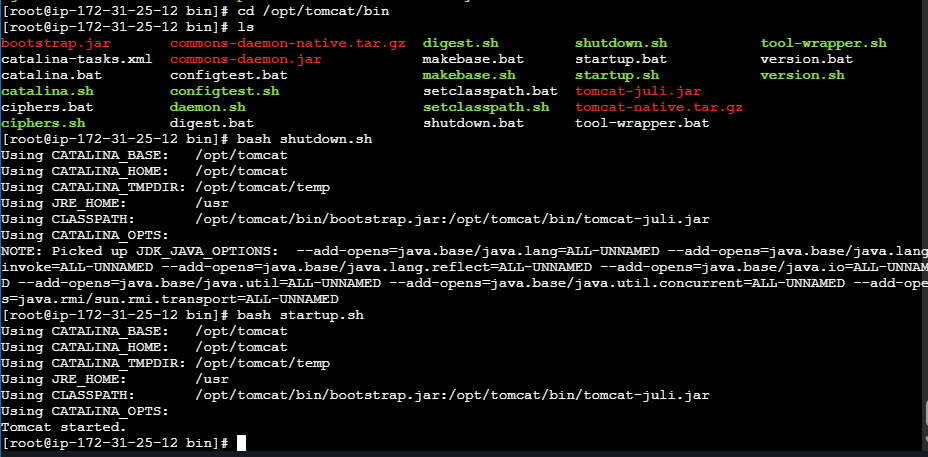
**After adding user to tomcat we need to shutdown and then start the tomcat server**

****

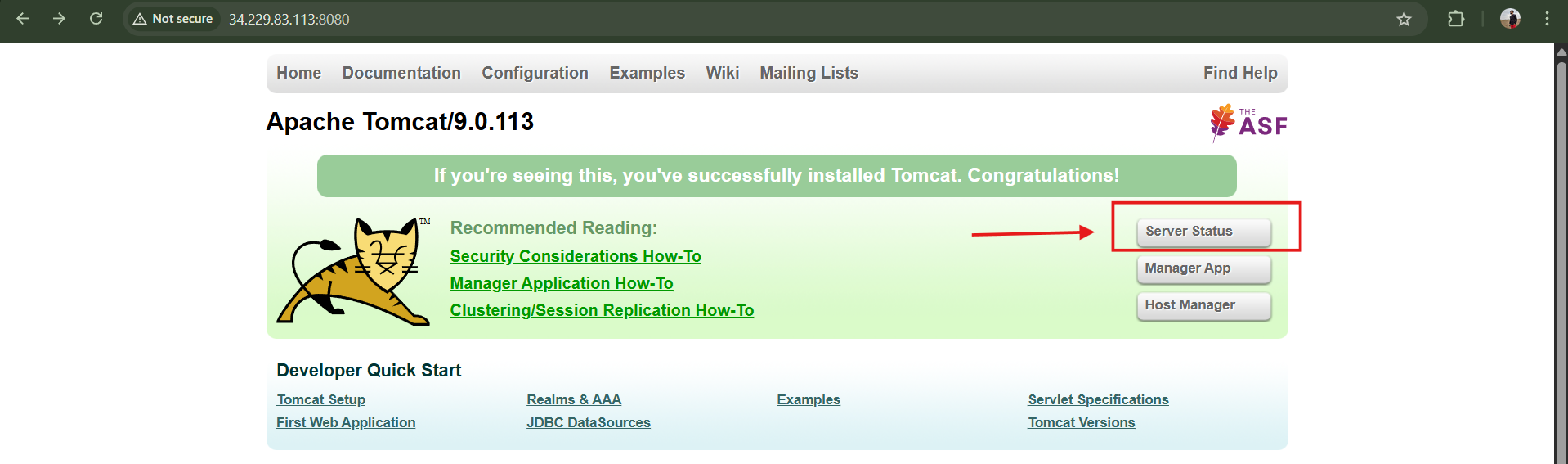
****

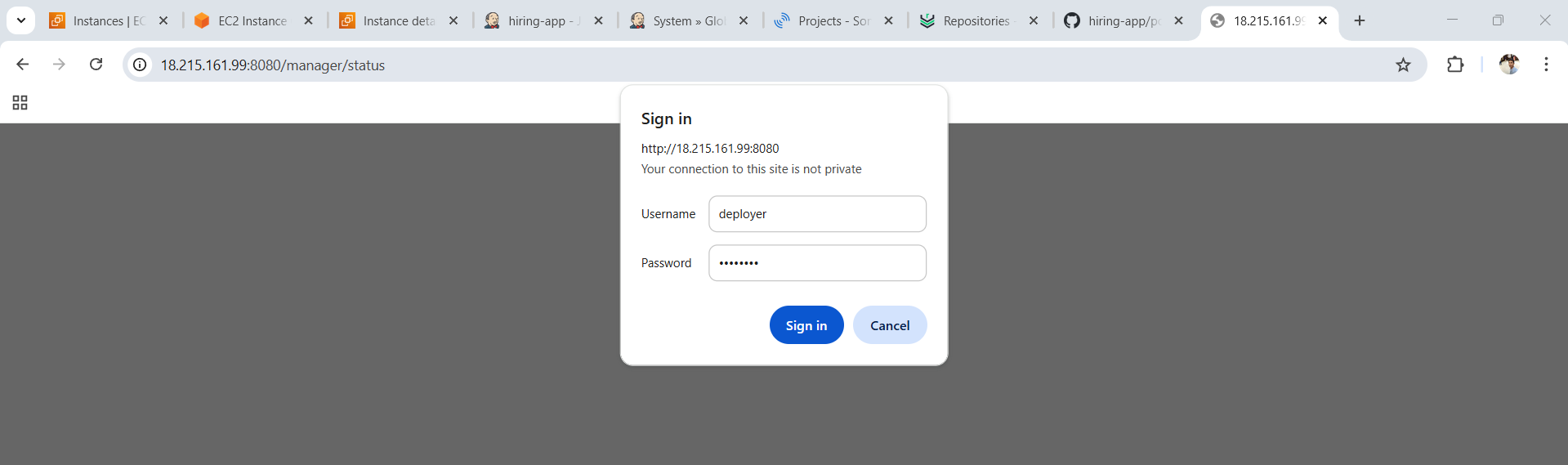
**Shutdown tomcat 🡪 bash shutdown.sh**

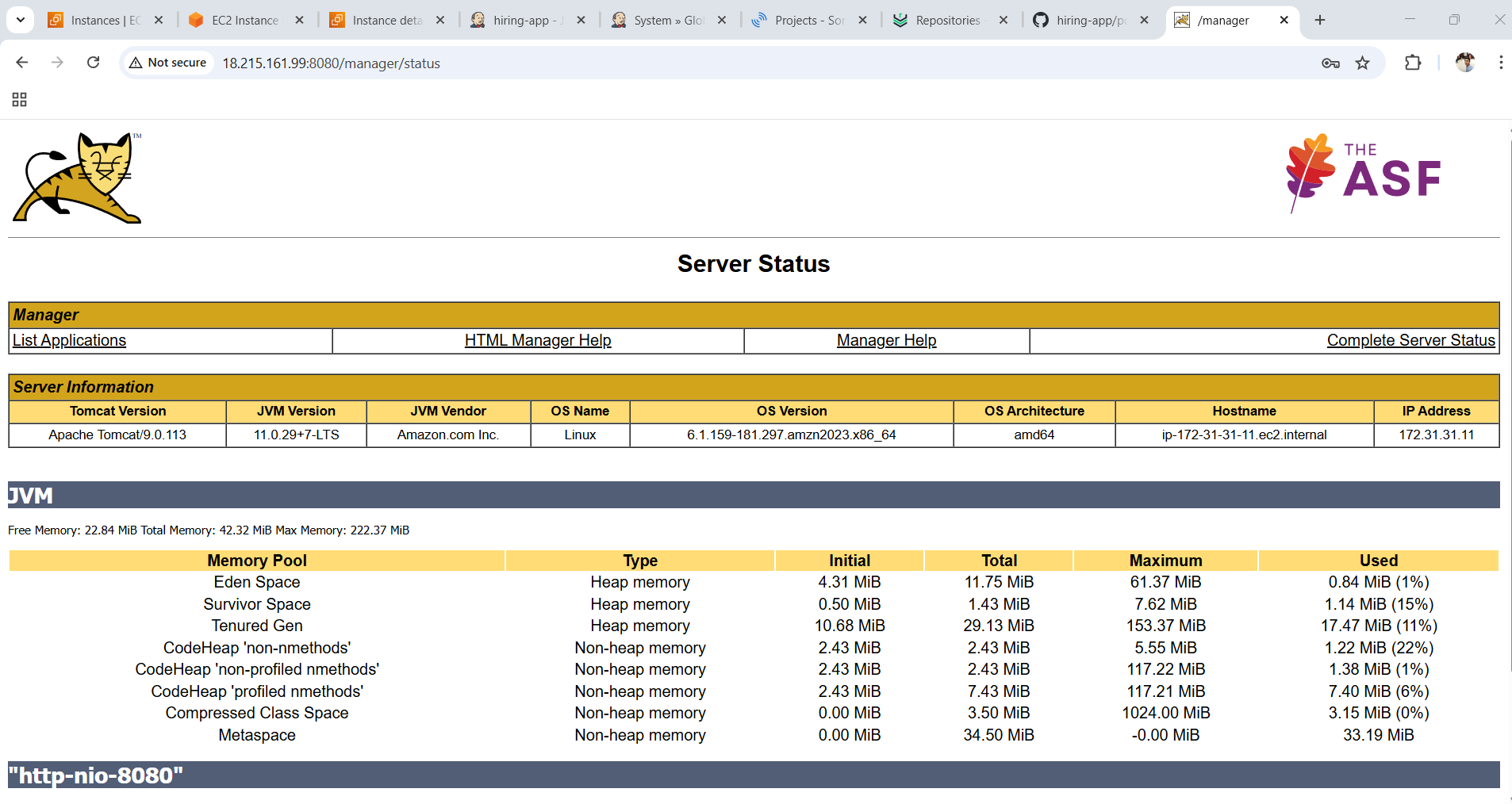
**Start tomcat 🡪 bash startup.sh**

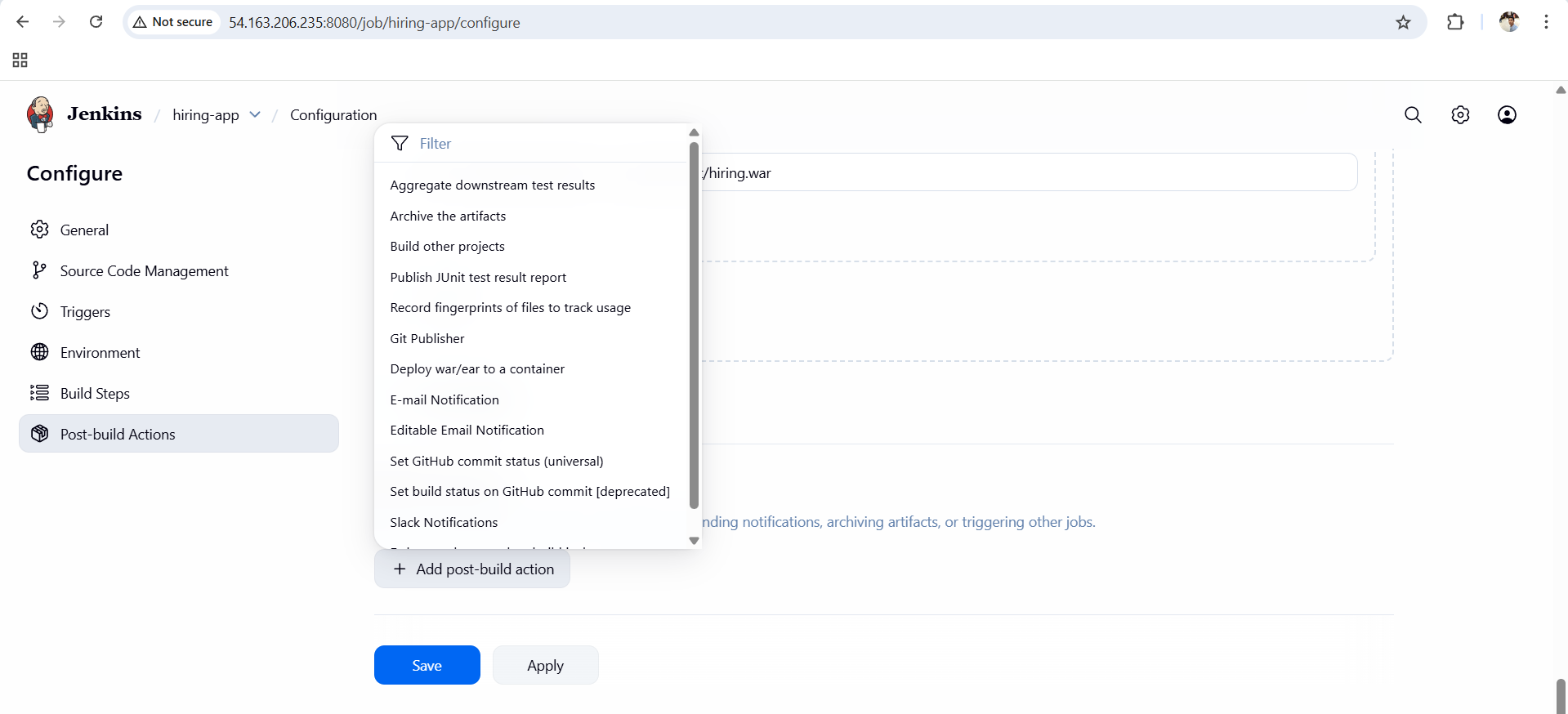
****

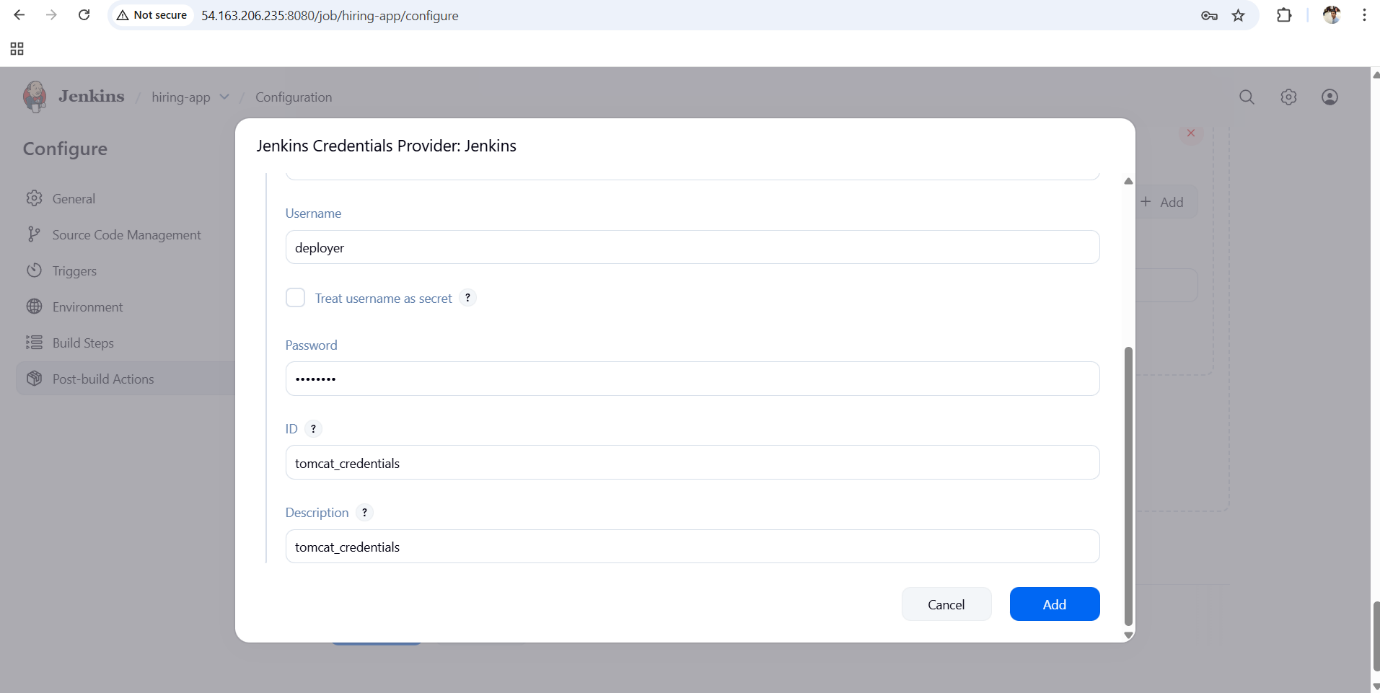
**Refresh the tomcat server page then click on server status**

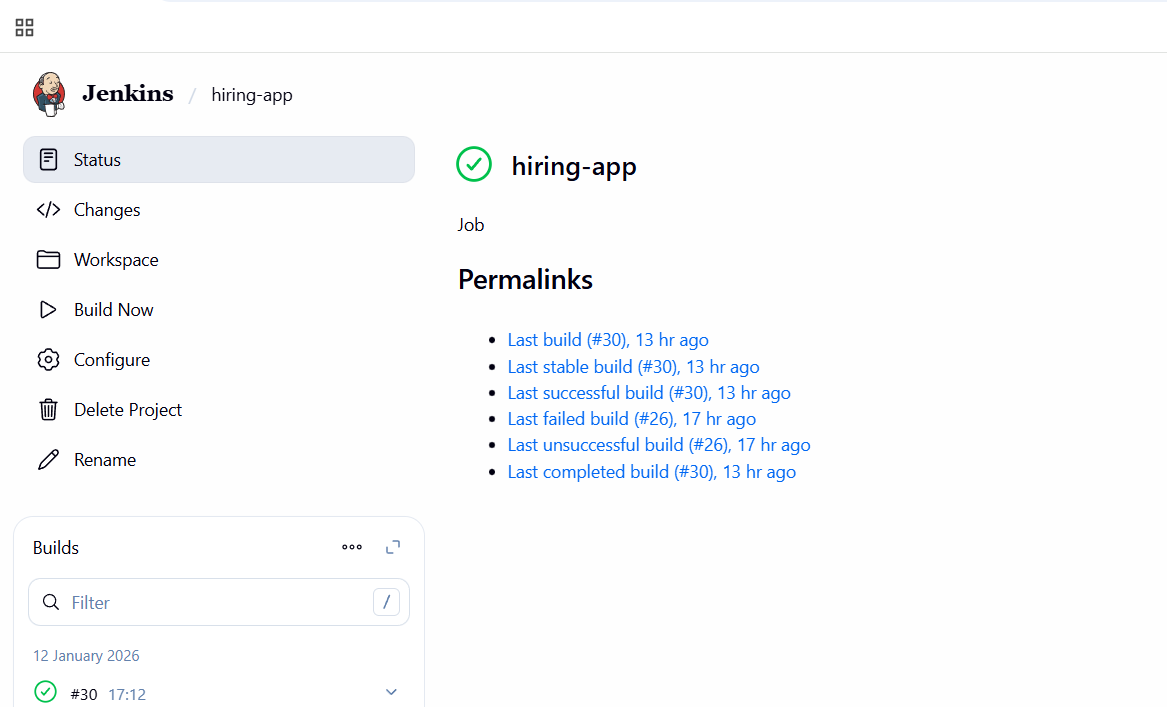
****











CONCLUSION:

Jenkins pipelines provide a **powerful and flexible way** to implement CI/CD practices by automating the entire application lifecycle.  
By using either **Declarative or Scripted pipelines**, teams can improve code quality, reduce manual errors, and deliver applications faster and more reliably. Jenkins pipelines are widely adopted in real-world DevOps environments for building scalable and maintainable automation workflows.

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:  
Git Clone  
SonarQube Integration  
Maven Compilation  
Nexus Artifactory  
Slack Notification  
Deploy On tomcat

TITLE:

Jenkins CICD pipeline using Declarative pipeline using feature-1.1 branch.

OBJECTIVE:

The objective of this project is to **set up an end-to-end CI/CD pipeline using Jenkins Declarative Pipeline** for the **feature-1.1 branch** of the application.  
The pipeline automates **source code checkout, static code analysis using SonarQube, Maven compilation, artifact storage in Nexus, deployment to Tomcat, and build notifications via Slack**, ensuring faster, reliable, and consistent application delivery.

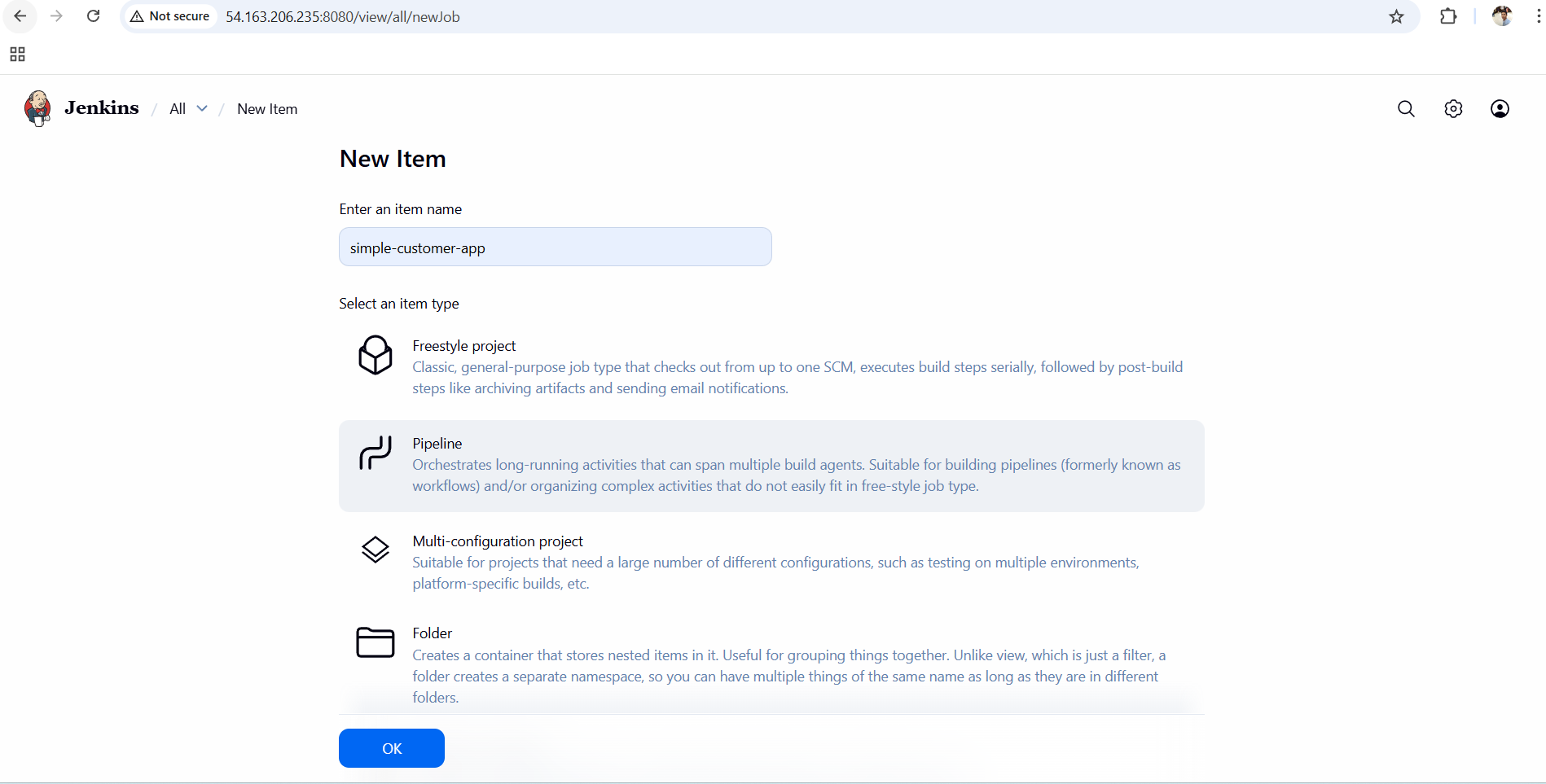
PREREQUISITES:

Before implementing this Jenkins CI/CD pipeline, the following prerequisites are required:

1. Jenkins installed and running
2. GitHub repository with **feature-1.1 branch** available
3. SonarQube server configured and accessible from Jenkins
4. SonarQube token added in Jenkins credentials
5. Maven and Java installed on Jenkins server
6. Nexus Repository Manager configured with hosted repository
7. Nexus credentials stored in Jenkins
8. Apache Tomcat server installed and running
9. Tomcat Manager access or deployment permissions configured
10. Slack workspace with incoming webhook or Slack Jenkins plugin configured

STEP-BY-STEP-IMPLIMENTATION:

Create a job of pipeline based.



Give this pipeline and run the build.

pipeline {

agent any

environment {

// Maven

MVN\_HOME = "/opt/maven/bin/mvn"

// Nexus

NEXUS\_URL = "http://98.93.29.69:8081/repository/simple-customer-app/"

GROUP\_ID = "in.javahome"

ARTIFACT = "hiring"

VERSION = "0.1"

// Tomcat

TOMCAT\_URL = "http://172.31.31.11:8080"

CONTEXT = "hiring"

}

stages {

stage('Clean Workspace') {

steps {

deleteDir()

}

}

stage('Git Clone') {

steps {

git branch: 'main',

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

}

}

stage('Build WAR') {

steps {

sh '''

${MVN\_HOME} clean package -DskipTests

'''

}

}

stage('Upload to Nexus (Release)') {

steps {

withCredentials([usernamePassword(

credentialsId: 'nexus\_credential',

usernameVariable: 'NEXUS\_USER',

passwordVariable: 'NEXUS\_PASS'

)]) {

sh '''

cat > settings.xml <<EOF

<settings>

<servers>

<server>

<id>simple-customer-app</id>

<username>${NEXUS\_USER}</username>

<password>${NEXUS\_PASS}</password>

</server>

</servers>

</settings>

EOF

${MVN\_HOME} deploy:deploy-file \

-s settings.xml \

-DgroupId=${GROUP\_ID} \

-DartifactId=${ARTIFACT} \

-Dversion=${VERSION} \

-Dpackaging=war \

-Dfile=target/${ARTIFACT}.war \

-DrepositoryId=simple-customer-app \

-Durl=${NEXUS\_URL}

'''

}

}

}

stage('Deploy to Tomcat (Manager API)') {

steps {

withCredentials([usernamePassword(

credentialsId: 'tomcat\_credentials', //

usernameVariable: 'TC\_USER',

passwordVariable: 'TC\_PASS'

)]) {

sh '''

echo "Undeploying existing application (if any)..."

curl -u ${TC\_USER}:${TC\_PASS} \

"${TOMCAT\_URL}/manager/text/undeploy?path=/${CONTEXT}" || true

echo "Deploying new WAR via Tomcat Manager API..."

curl -u ${TC\_USER}:${TC\_PASS} \

-T target/${ARTIFACT}.war \

"${TOMCAT\_URL}/manager/text/deploy?path=/${CONTEXT}&update=true"

'''

}

}

}

}

post {

success {

echo " FULL CI/CD PIPELINE SUCCESS (Git → Maven → Nexus → Tomcat)"

}

failure {

echo " PIPELINE FAILED – Check logs"

}

}

}



CONCLUSION:

By implementing this Jenkins Declarative CI/CD pipeline, the application delivery process is **fully automated from code commit to deployment**.  
The integration of **SonarQube** improves code quality, **Maven** ensures reliable builds, **Nexus** provides centralized artifact management, **Tomcat** enables automated deployment, and **Slack notifications** offer real-time build status updates.  
This pipeline demonstrates a **real-world DevOps workflow** that improves deployment speed, reduces manual errors, and enhances overall software quality.

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:  
Git Clone  
SonarQube Integration  
Maven Compilation  
Nexus Artifactory  
Slack Notification  
Deploy On tomcat

TITLE:

Jenkins CICD pipeline using Scripted pipeline using feature- 1.1 branch.

OBJECTIVE:

The objective is to implement a **Scripted Jenkins CI/CD pipeline** for the **feature-1.1 branch** that automates the complete software delivery lifecycle.  
The pipeline performs **source code checkout, static code analysis using SonarQube, Maven compilation, artifact publishing to Nexus, application deployment on Tomcat, and build notifications via Slack**, ensuring faster, reliable, and quality-driven deployments.

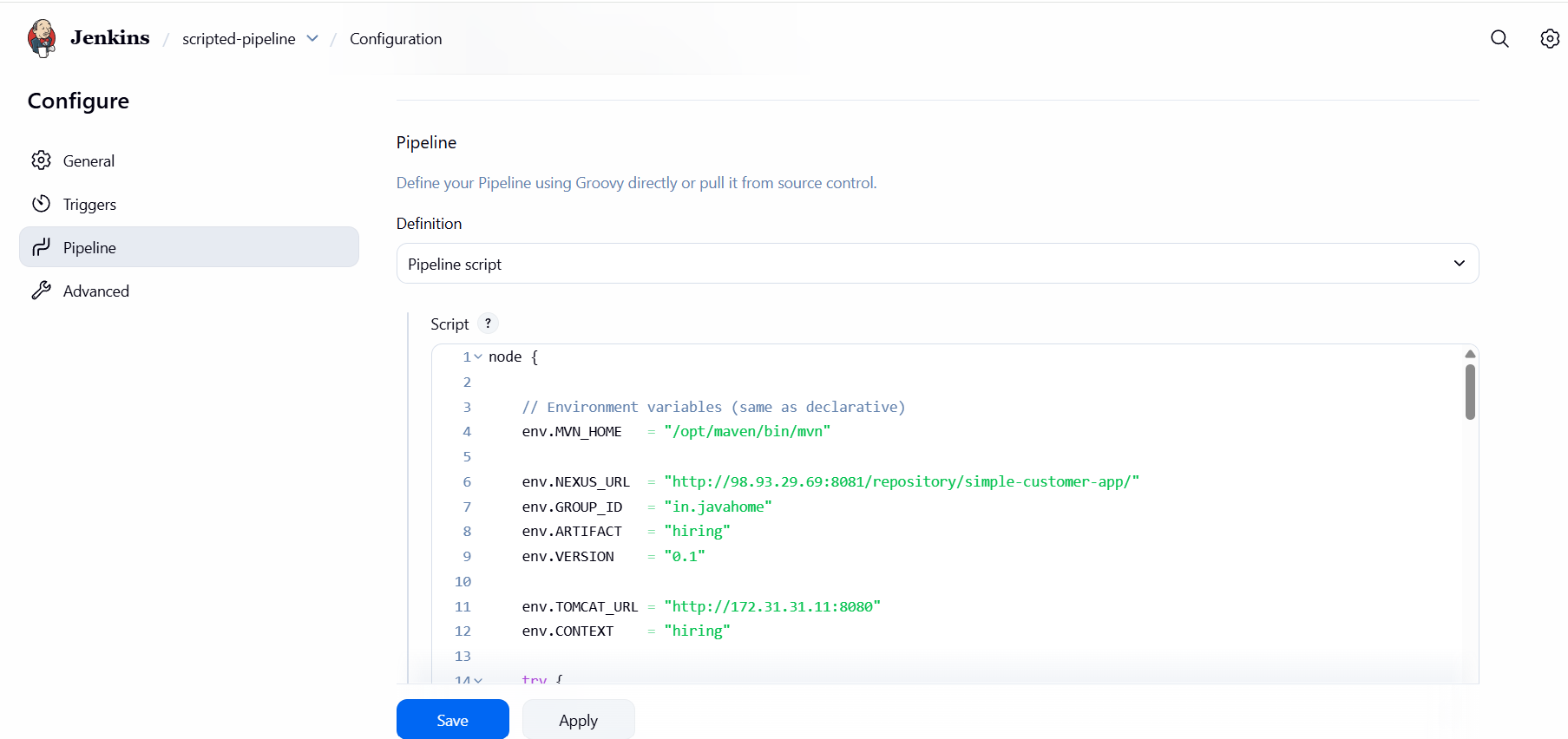
PREREQUISITES:

Before setting up this pipeline, ensure the following prerequisites are met:

1. Jenkins server installed and running
2. Git plugin and Pipeline plugin installed in Jenkins
3. GitHub repository with **feature-1.1** branch available
4. SonarQube server up and accessible from Jenkins
5. SonarQube token configured in Jenkins credentials
6. Maven and Java installed on Jenkins server
7. Nexus Repository Manager configured (hosted Maven repo)
8. Nexus credentials stored in Jenkins
9. Apache Tomcat installed and running
10. Tomcat Manager configured for deployment
11. Slack Jenkins plugin or Webhook configured

STEP-BY-STEP-IMPLIMENTATION:

Go to configure jobs and give the pipeline and build the job.



node {

// Environment variables (same as declarative)

env.MVN\_HOME = "/opt/maven/bin/mvn"

env.NEXUS\_URL = "http://98.93.29.69:8081/repository/simple-customer-app/"

env.GROUP\_ID = "in.javahome"

env.ARTIFACT = "hiring"

env.VERSION = "0.1"

env.TOMCAT\_URL = "http://172.31.31.11:8080"

env.CONTEXT = "hiring"

try {

stage('Clean Workspace') {

deleteDir()

}

stage('Git Clone') {

git branch: 'main',

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

}

stage('Build WAR') {

sh """

${env.MVN\_HOME} clean package -DskipTests

"""

}

stage('Upload to Nexus (Release)') {

withCredentials([usernamePassword(

credentialsId: 'nexus\_credential',

usernameVariable: 'NEXUS\_USER',

passwordVariable: 'NEXUS\_PASS'

)]) {

sh """

cat > settings.xml <<EOF

<settings>

<servers>

<server>

<id>simple-customer-app</id>

<username>\${NEXUS\_USER}</username>

<password>\${NEXUS\_PASS}</password>

</server>

</servers>

</settings>

EOF

${env.MVN\_HOME} deploy:deploy-file \

-s settings.xml \

-DgroupId=${env.GROUP\_ID} \

-DartifactId=${env.ARTIFACT} \

-Dversion=${env.VERSION} \

-Dpackaging=war \

-Dfile=target/${env.ARTIFACT}.war \

-DrepositoryId=simple-customer-app \

-Durl=${env.NEXUS\_URL}

"""

}

}

stage('Deploy to Tomcat (Manager API)') {

withCredentials([usernamePassword(

credentialsId: 'tomcat\_credentials',

usernameVariable: 'TC\_USER',

passwordVariable: 'TC\_PASS'

)]) {

sh """

echo "Undeploying existing application (if any)..."

curl -u \${TC\_USER}:\${TC\_PASS} \

"${env.TOMCAT\_URL}/manager/text/undeploy?path=/${env.CONTEXT}" || true

echo "Deploying new WAR via Tomcat Manager API..."

curl -u \${TC\_USER}:\${TC\_PASS} \

-T target/${env.ARTIFACT}.war \

"${env.TOMCAT\_URL}/manager/text/deploy?path=/${env.CONTEXT}&update=true"

"""

}

}

echo " FULL CI/CD PIPELINE SUCCESS (Git → Maven → Nexus → Tomcat)"

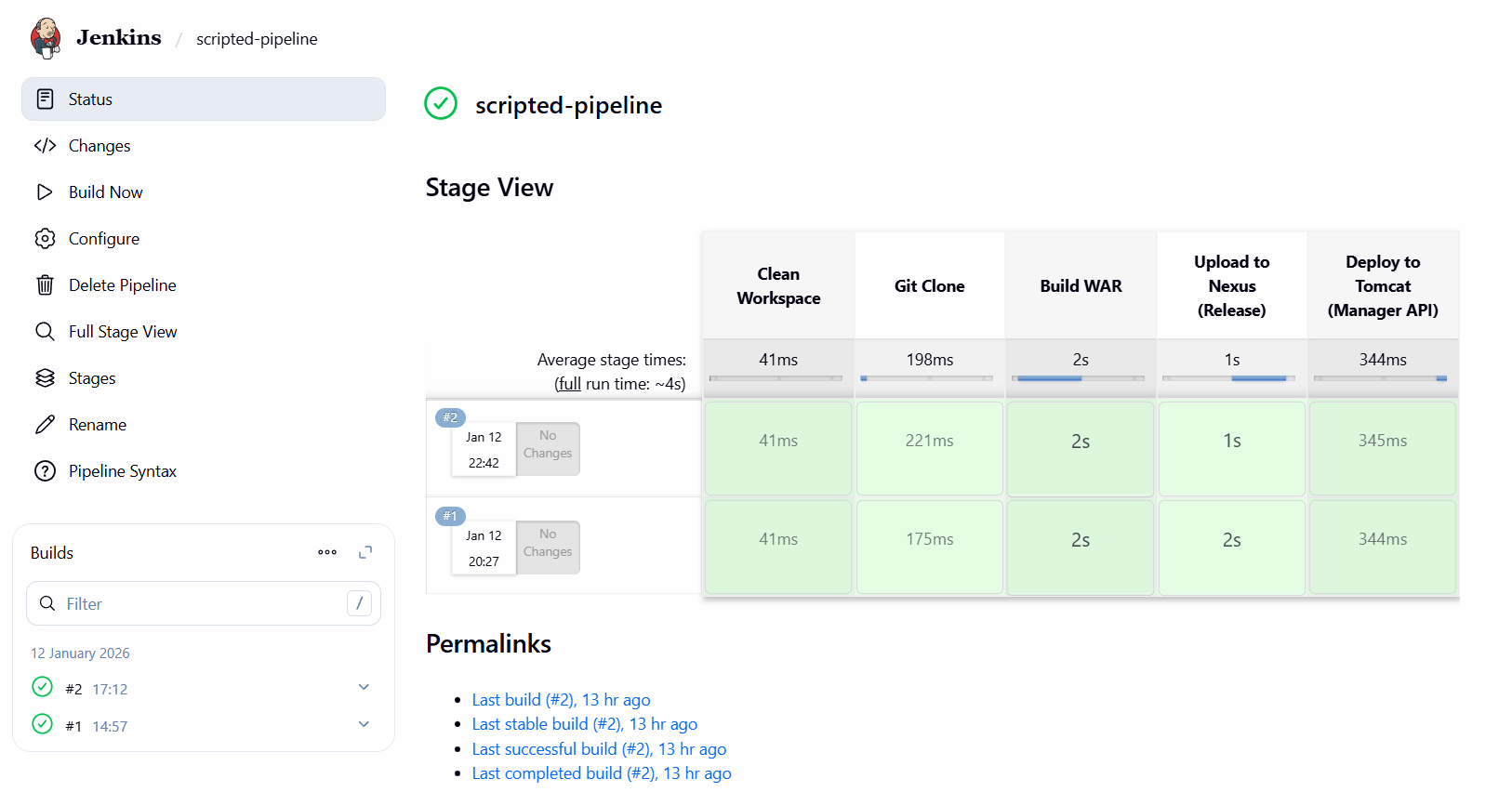
} catch (err) {

echo " PIPELINE FAILED – Check logs"

throw err

}

}



CONCLUSION:

This Scripted Jenkins CI/CD pipeline successfully automates the **end-to-end application delivery process** for the **feature-1.1 branch**.  
By integrating **SonarQube for quality checks**, **Maven for build automation**, **Nexus for artifact management**, **Tomcat for deployment**, and **Slack for notifications**, the pipeline ensures **high-quality, reliable, and fast software delivery**.  
This setup reflects a **real-world enterprise DevOps workflow** and demonstrates effective CI/CD implementation using Jenkins Scripted pipelines.

4. Write sample skeleton of pipelines.

**🧩 Sample Scripted Pipeline – Skeleton**

**Note:** This is a **skeleton only** (structure), not a full implementation.

node {

stage('Git Clone') {

// Clone feature-1.1 branch from GitHub repository

}

stage('SonarQube Integration') {

// Perform static code analysis using SonarQube

}

stage('Maven Compilation') {

// Compile and package the application using Maven

}

stage('Nexus Artifactory') {

// Upload build artifact to Nexus repository

}

stage('Deploy on Tomcat') {

// Deploy application to Apache Tomcat server

}

stage('Slack Notification') {

// Send build status notification to Slack

}

}

5. Create a parameterized job in Jenkins. <https://github.com/betawins/spring3-mvc-maven-xml-hello-world-1.git>.

TITLE:

Parametrized job in Jenkins

OBJECTIVE:

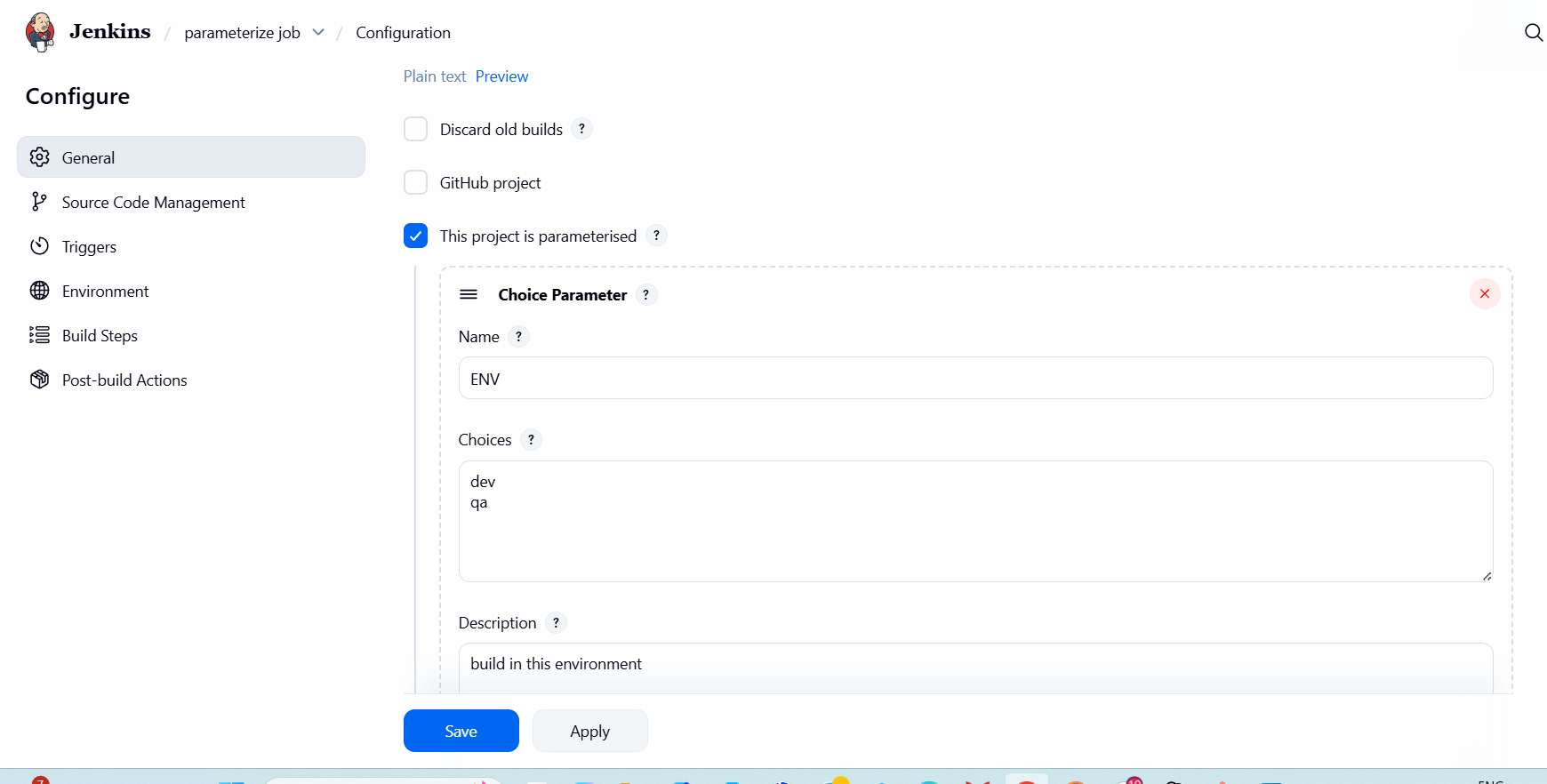
To implement a **Scripted Jenkins CI/CD pipeline** for the **feature-1.1 branch** that automates source code checkout, code quality analysis, build and packaging, artifact storage, deployment to Tomcat, and build status notification using **Jenkins**.

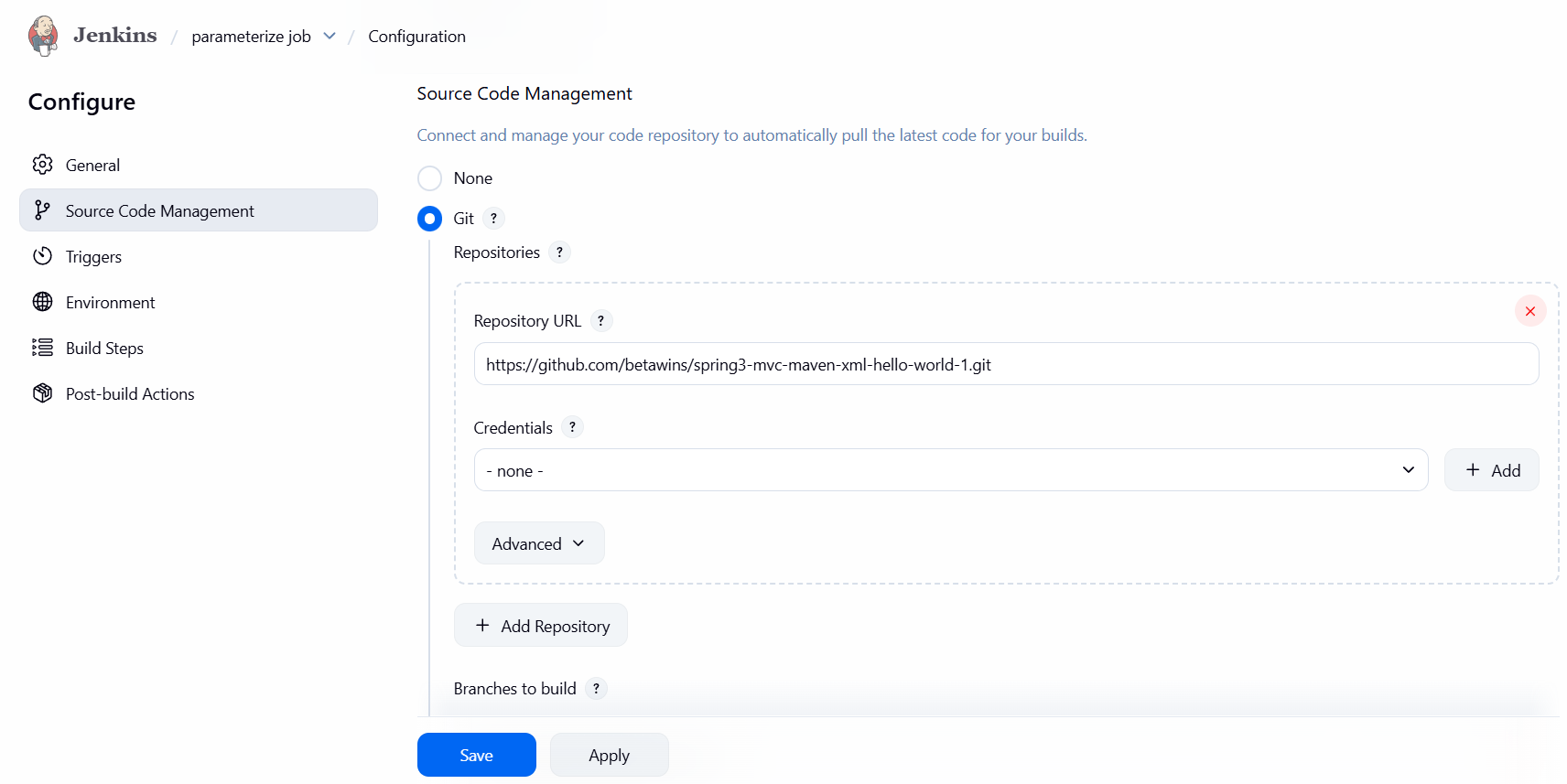
PREREQUISITES:

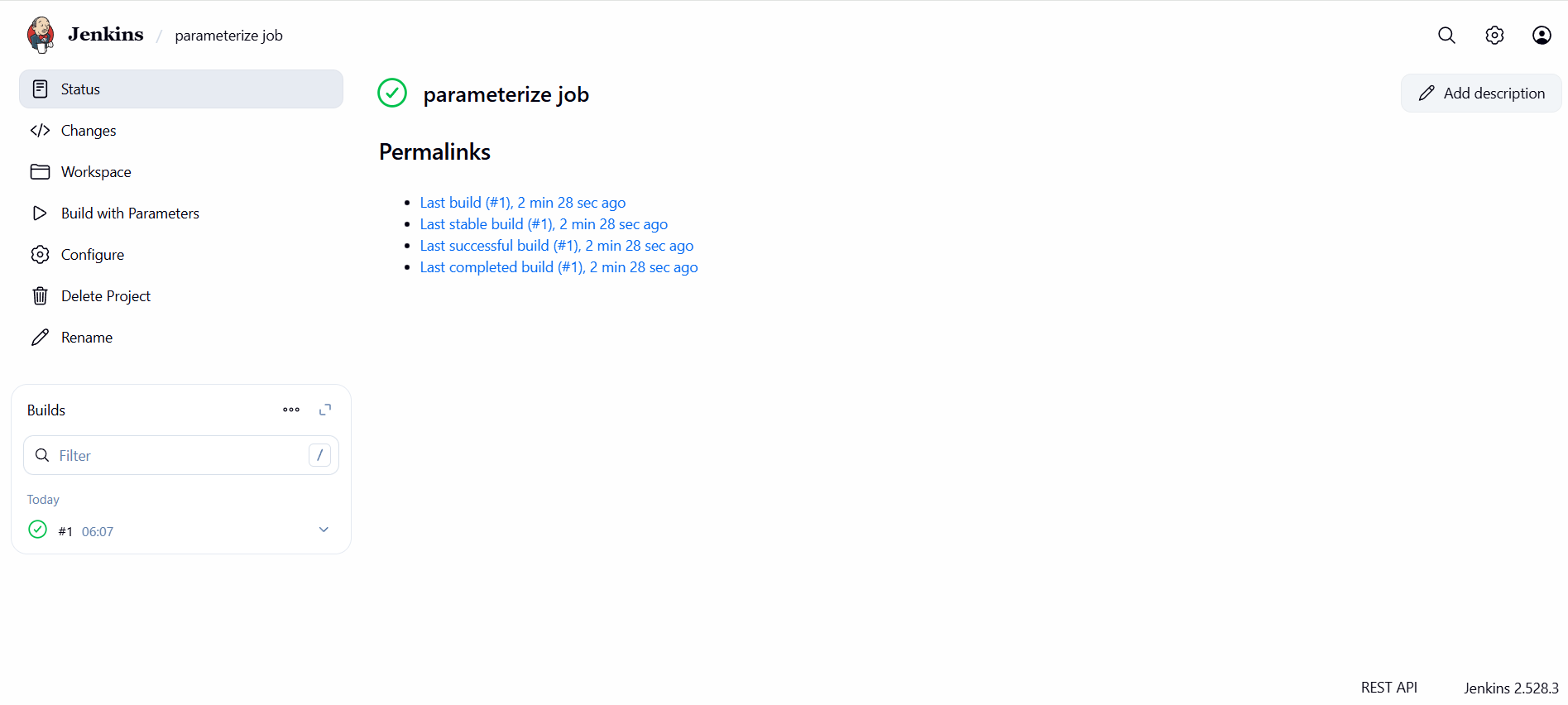
* Jenkins installed and running
* GitHub repository with feature-1.1 branch
* Jenkins Pipeline and Git plugins installed
* SonarQube server and token configured
* Maven and Java installed on Jenkins server
* Nexus Repository Manager configured
* Nexus credentials added in Jenkins
* Apache Tomcat installed and accessible
* Tomcat Manager configured for deployment
* Slack integration configured in Jenkins.

STEP-BY-STEP-IMPLIMENTATION:

Go to Jenkins dashboard and create a job .







CONCLUSION:

The Scripted Jenkins CI/CD pipeline automates the complete software delivery process for the feature-1.1 branch. By integrating SonarQube, Maven, Nexus, Tomcat, and Slack, the pipeline improves code quality, reduces manual intervention, and ensures faster and reliable application deployment.

6. Setup one slave machine for Jenkins.

TITLE:

Slave machine in Jenkins master.

OBJECTIVE:

To configure slave (agent) machines in Jenkins so that build and deployment jobs are executed on remote nodes, improving performance, scalability, and efficient resource utilization.

PREREQUISITES:

Jenkins master server must be installed and running.

Slave machine must be reachable from Jenkins master over the network.

Compatible Java version must be installed on the slave machine.

SSH service must be enabled and running on the slave machine.

Passwordless SSH (public–private key authentication) must be configured.

Jenkins user must have required permissions on the slave machine.

Required build tools (Git, Maven, Docker, etc.) must be installed on the slave

STEP-BY-STEP-IMPLIMENTATION:

Steps to be done on slave

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

Login to slave machine..

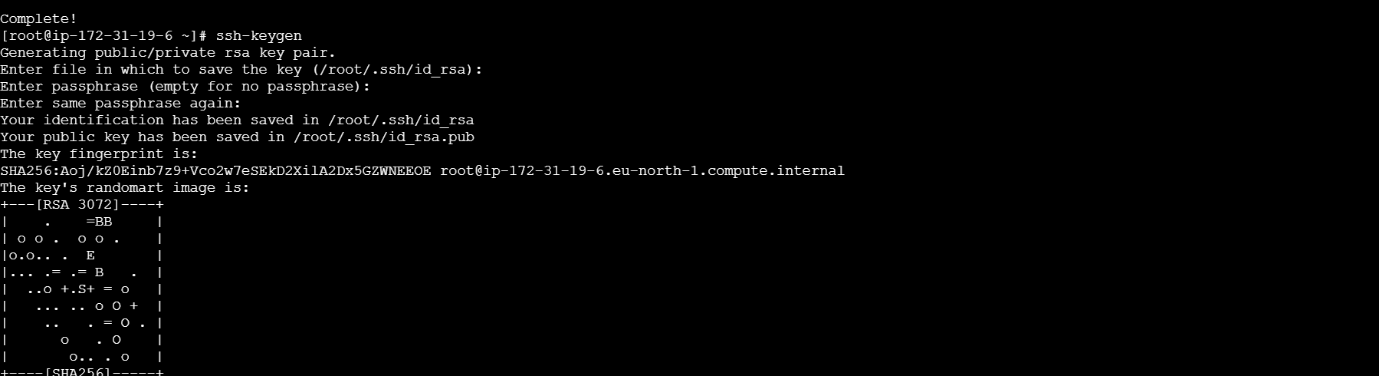
switch to root user

install jdk 11 (amazon-linux-extras install java-openjdk11)

Create ssh-keygen

cat id\_rsa.pub > authorized\_keys

chmod 700 authorized\_keys





Steps to be done on master machine:

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

Login to master machine

switch to root user.

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

cd /var/lib/jenkins/.ssh

ssh-keyscan -H SLAVE-NODE-IP-OR-HOSTNAME >>/var/lib/jenkins/.ssh/known\_hosts

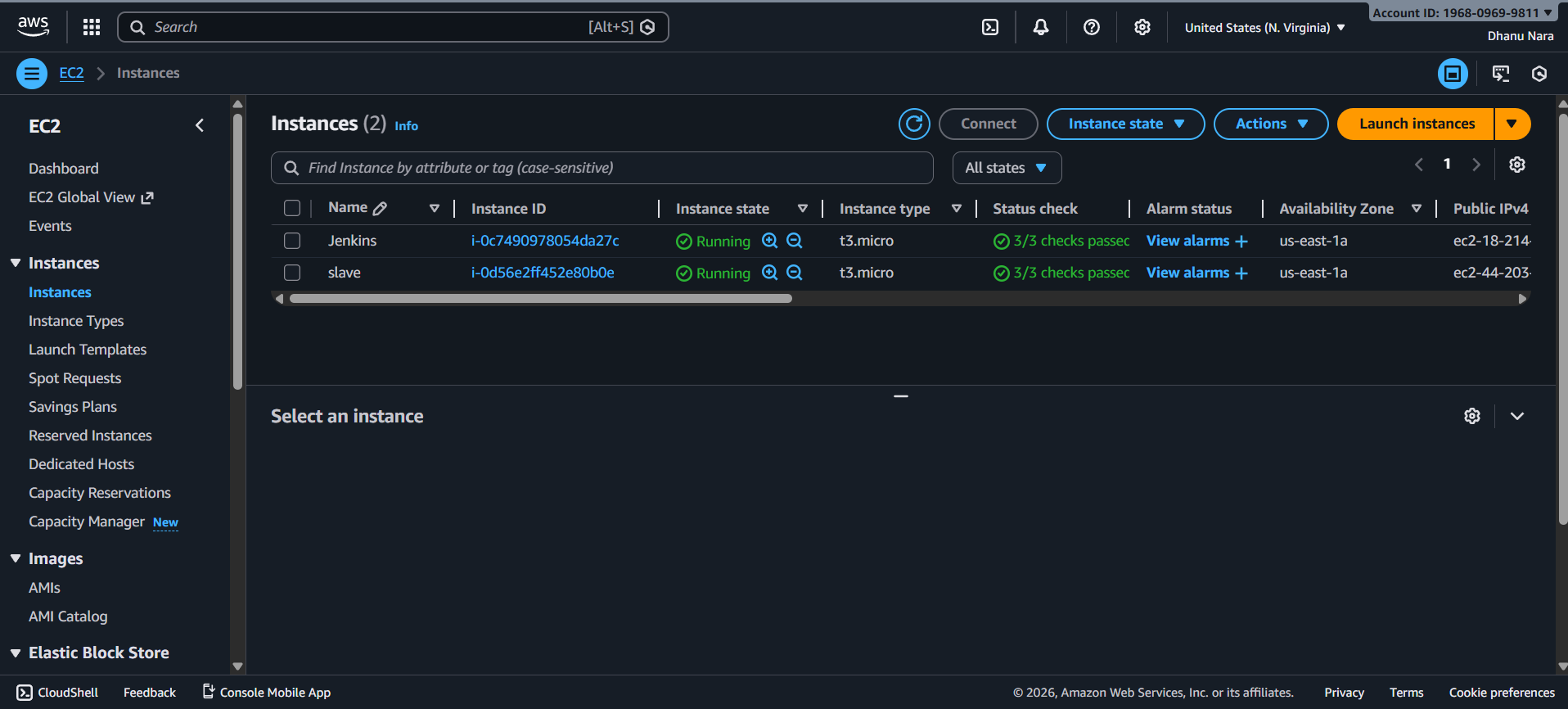
# ssh-keyscan -H 172.31.38.42 >>/var/lib/jenkins/.ssh/known\_hosts

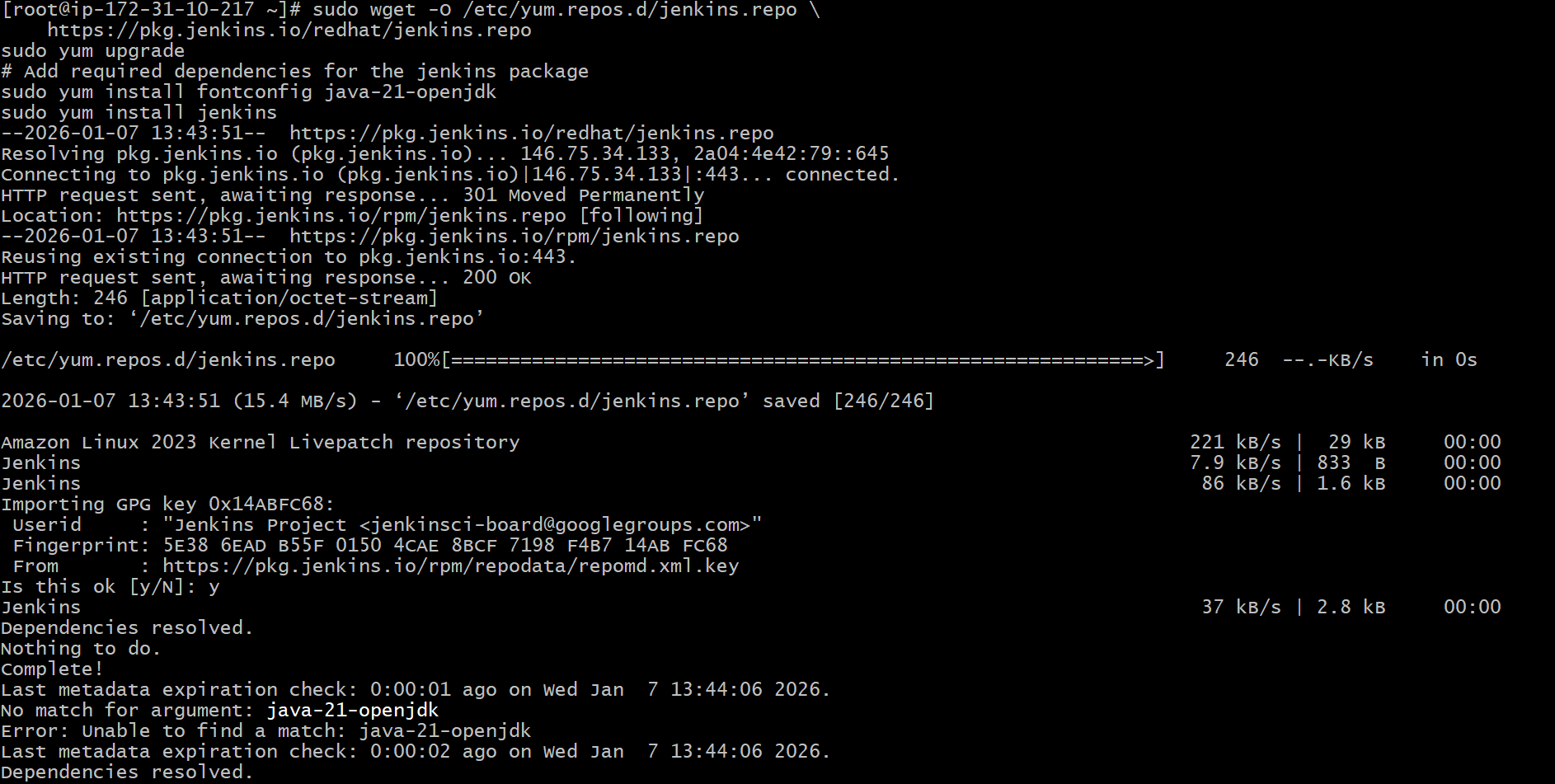
chown jenkins:jenkins known\_hosts

#we need to change the owner as we ran ssh-keyscan command using “root” user.

# default user of Jenkins will be “jenkins”

chmod 700 known\_hosts.





Go to manage jenkins

open manage node

under launch method select "Launch via SSh"

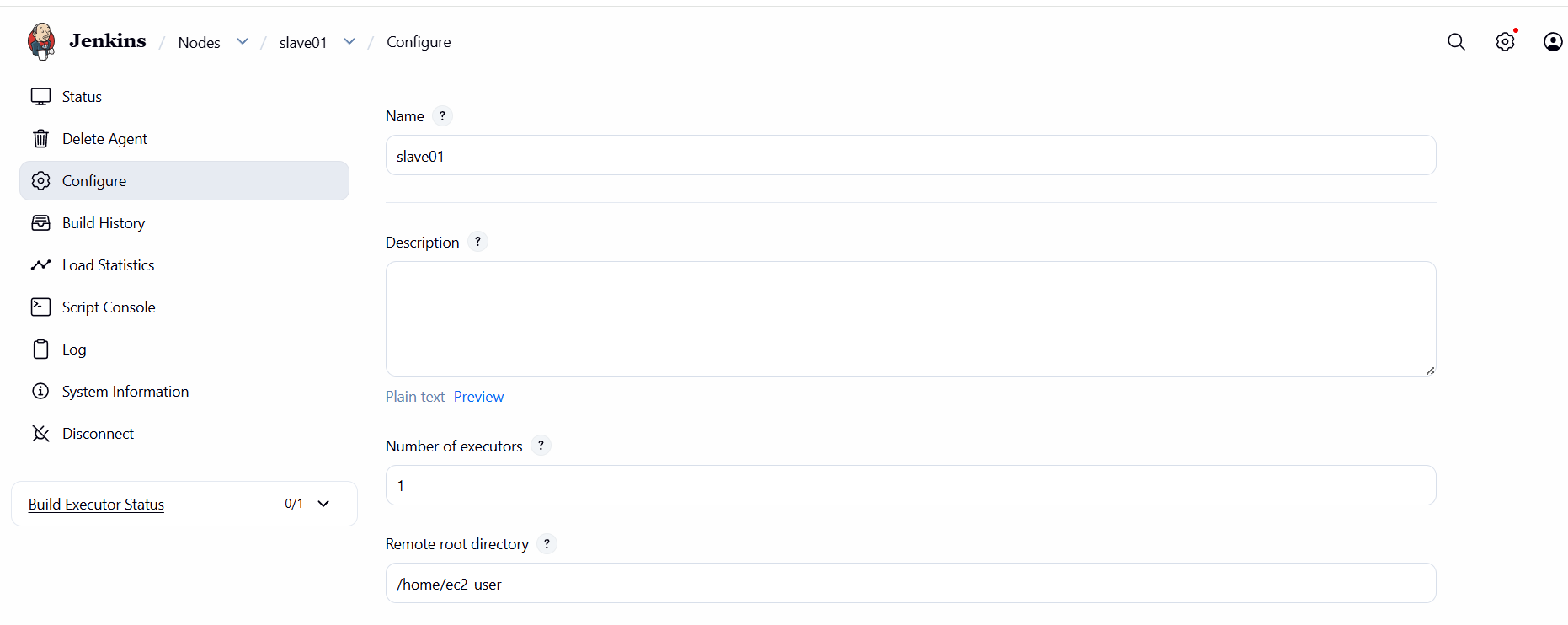
Host "public ip of slave"

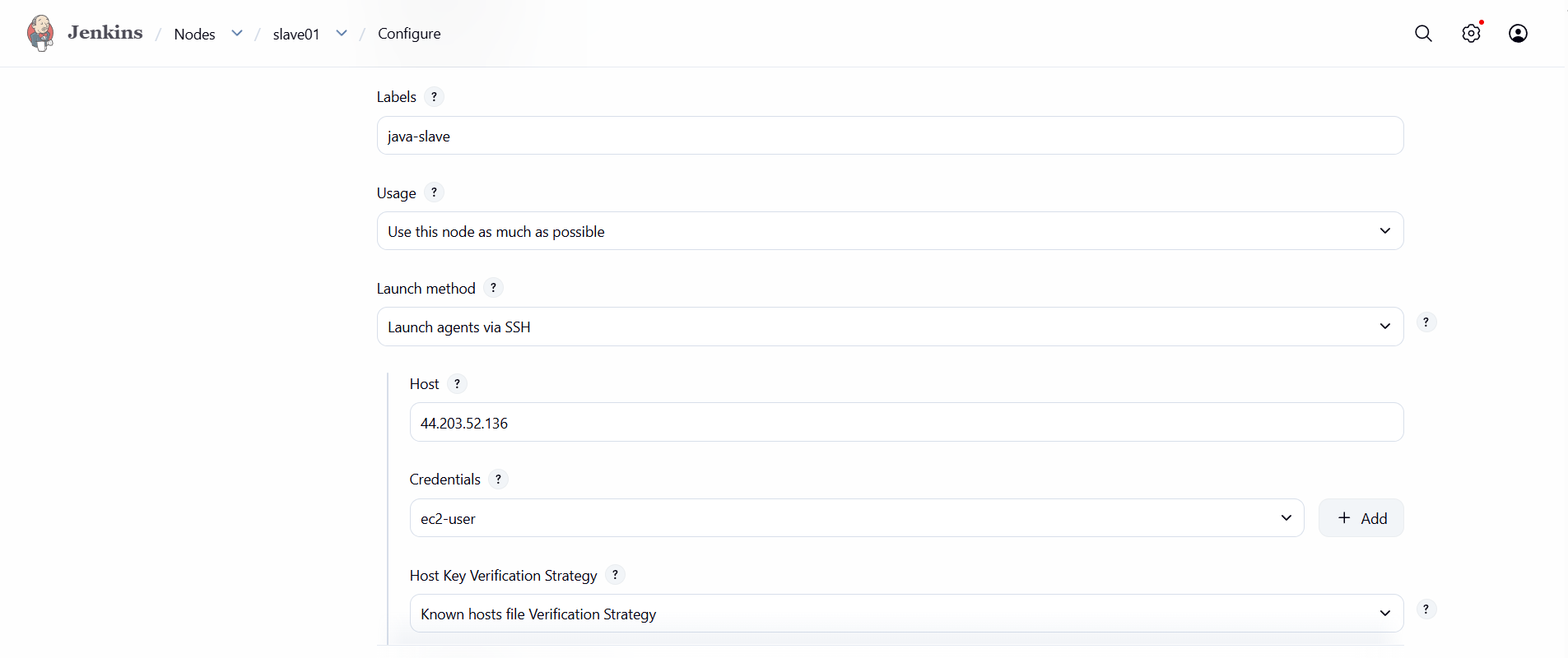
creditals: select user with private key

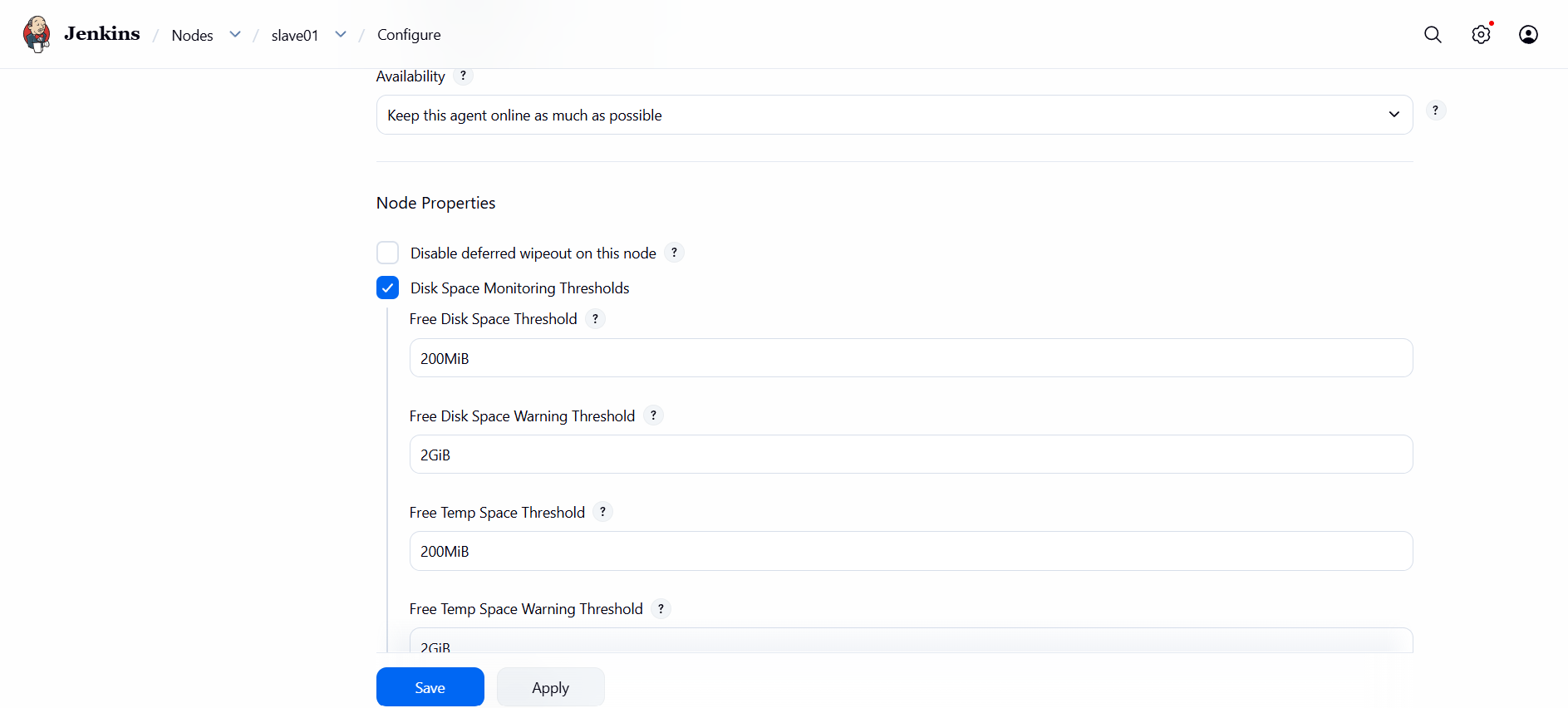
enter ec2-user as name and paste the pem key.

host key verification startegy to be selected

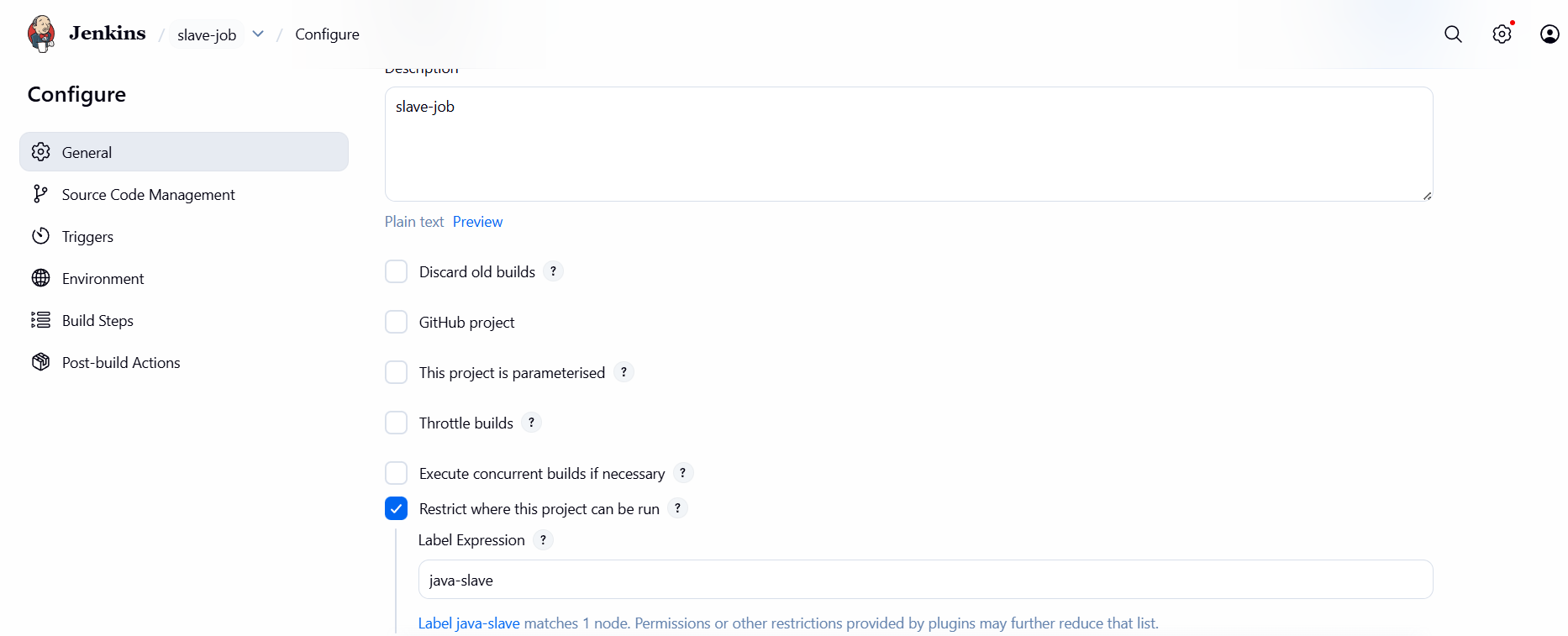
save..







Created one job by giving label java slave.





CONCLUSION:

Configuring slave machines in Jenkins enables distributed builds, reduces load on the master, and allows scalable, faster CI/CD execution across multiple environments.