PROJECT-3

Course: DevOps

Name: Vanguri Raja Vamsy

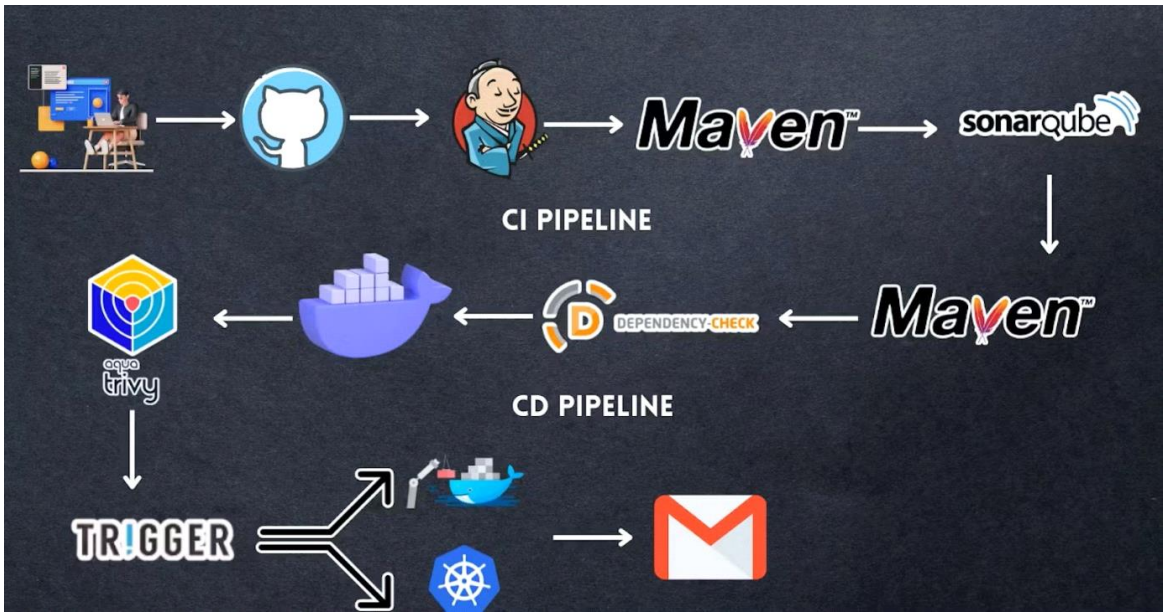
Batch No: 115 (9am-10am)

Mail id: rajavamsyvanguri@gmail.com

Trainer Name: Mr. Madhukar Reddy

Date: 22-12-23

Title: DEVSECOPS PROJECT: COMPLETE CI-CD (3 TIER APP)-DOCKERJENKINS.



ABSTRACT:

This project proposes a DevSecOps framework, seamlessly blending security into the software development and delivery pipeline. Focused on automating security testing, continuous monitoring, and threat detection, the initiative aims to proactively enhance security while accelerating time-to-market. Embracing Infrastructure as Code (IaC) security and fostering a collaborative culture, the project emphasizes shared responsibility among development, security, and operations teams. By automating incident response and promoting a unified approach, organizations can achieve a balanced synergy between speed and security, ensuring resilient software in the face of evolving cyber threats. This project provides a practical guide for implementing DevSecOps principles in the development lifecycle.

Steps: -

Step 1 — AWS

Step 2 — Server Connections and Installations.

Step 3 —Jenkins

Step 4 — Sonarqube

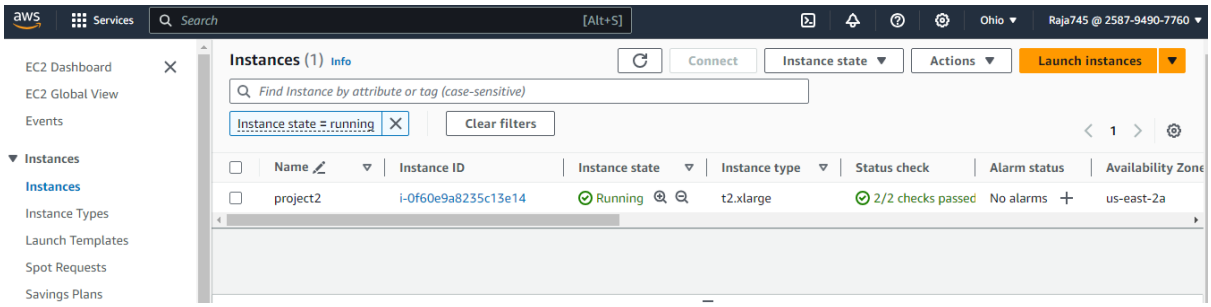
Step 5 — Install Nexus artifact.

Step 6 — Install tomcat and deploy the project in it.

Step 7 —DP-Check

Step 8 —Docker

STEP-1-AWS:



By using the AWS EC2, instance is created.

- Creating a t2.xlarge instance with the security group which is added the all traffic for inbound rules. I will allow all ports.
- Or we can specify the ports that which we need to allow.

STEP-2-SERVER CONECTIONS AND INSTALLATIONS:

- For the installation of Jenkins, we can install Jenkins through the terminal by implementing the following steps.
- We implement these commands by connecting through the console.

Installation of Jenkins:

- Execute the below command
  vi jenkins.sh
- Write the code in it.

```
#!/bin/bash

sudo apt update -y #sudo apt upgrade -y

wget -O -https://packages.adoptium.net/artifactory/api/gpg/key/public | tee

/etc/apt/keyrings/adoptium.asc

echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc]
https://packages.adoptium.net/artifactory/deb $(awk -F=
'/^VERSION_CODENAME/{print$2}' /etc/os-release) main" | tee

/etc/apt/sources.list.d/adoptium.list sudo apt update -y

sudo apt install temurin-17-jdk -y sudo apt install maven -y

/usr/bin/java --version

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null echo deb [signed-
by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update -y

sudo apt-get install jenkins -y

sudo systemctl start jenkins

sudo systemctl status Jenkins
```

- After writing code implement these commands to start the Jenkins.

  sudo chmod 777 jenkins.sh

  ./jenkins.sh # this will installl Jenkins

- Now copy the the Ip address and go through the Jenkins portal the interface will like shown below.

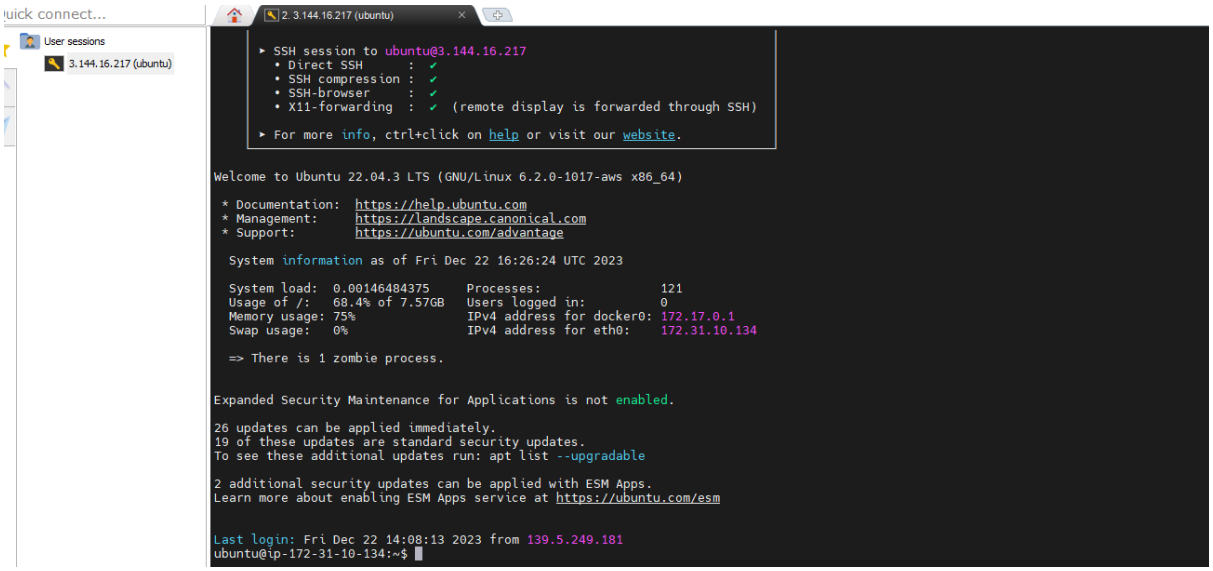
Installation of Docker:

- For the installation of the docker the following commands will be followed.

  sudo apt-get update.

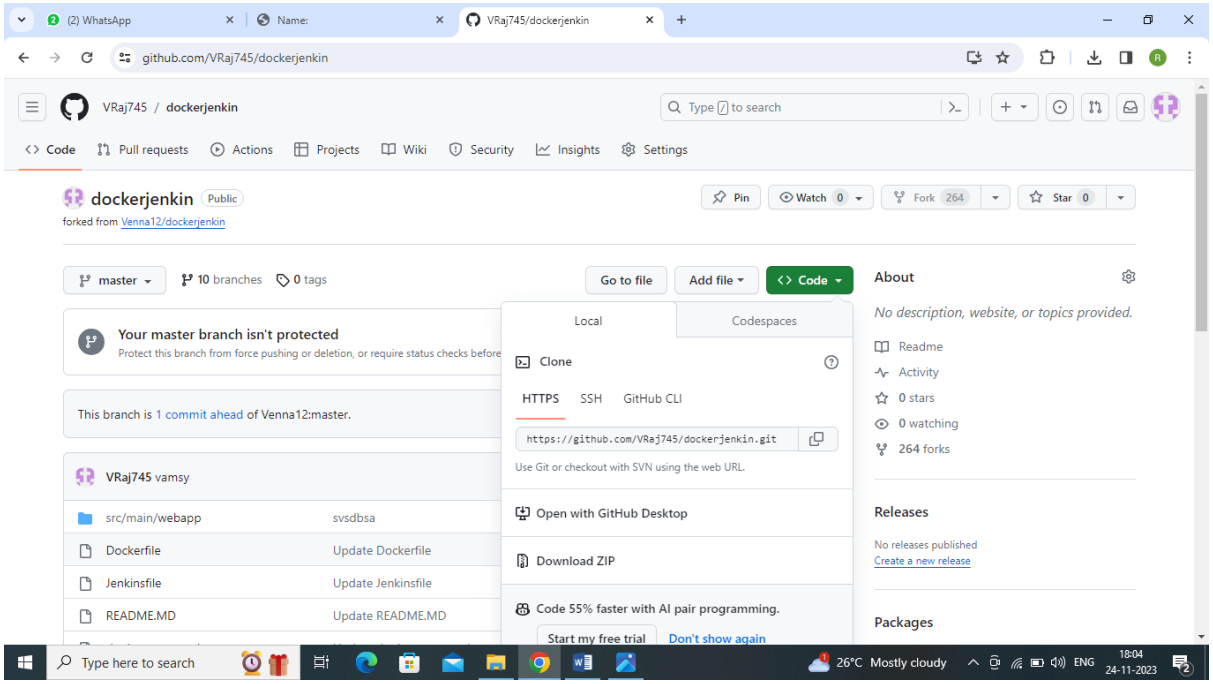
  sudo apt-get install docker.io -y

- Now after creation of the docker we need to create a sonar container.
- The default port number of the SonarQube is 9000.
  docker run -d --name sonar -p 9000:9000 sonarqube:lts-community.
- To check whether the container is created or not the command is.
  docker ps or docker ps -a



Above fig shows that the connection of instance in mobaxterm, which is used to install the jenkins, tomcat and required packages and more.

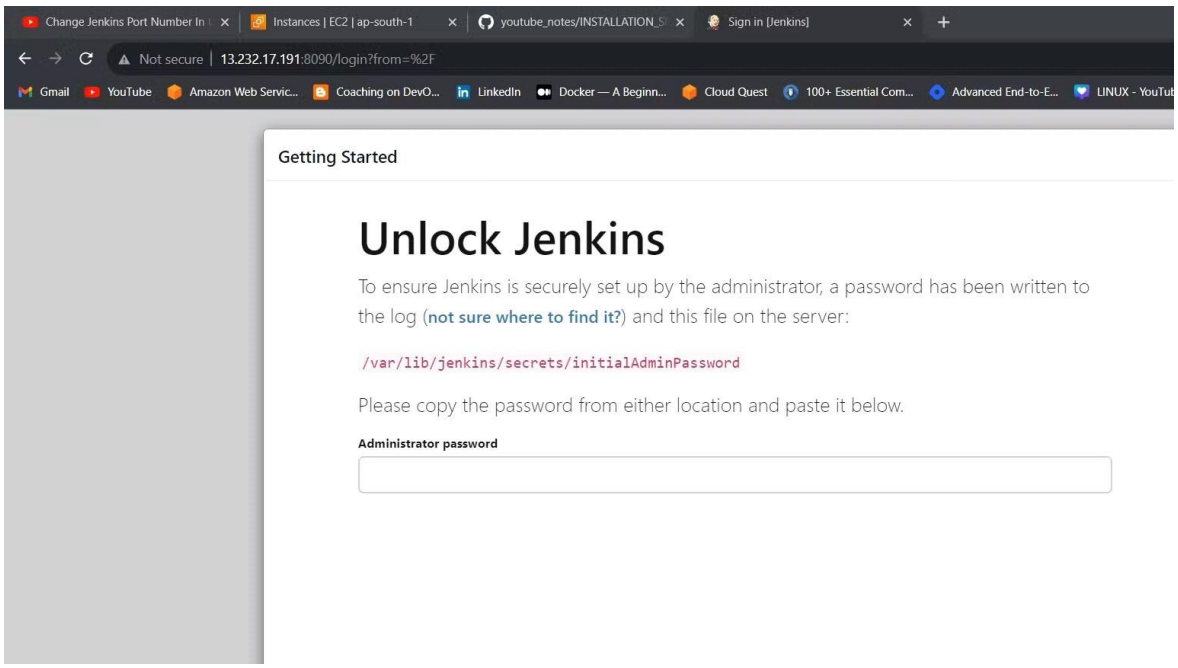
GITHUB:



Above fig shows that the git repository that which is used to deploy the webapp in the tomcat server. It is forked by the https://github.com/Venna12/dockerjenkin.git.

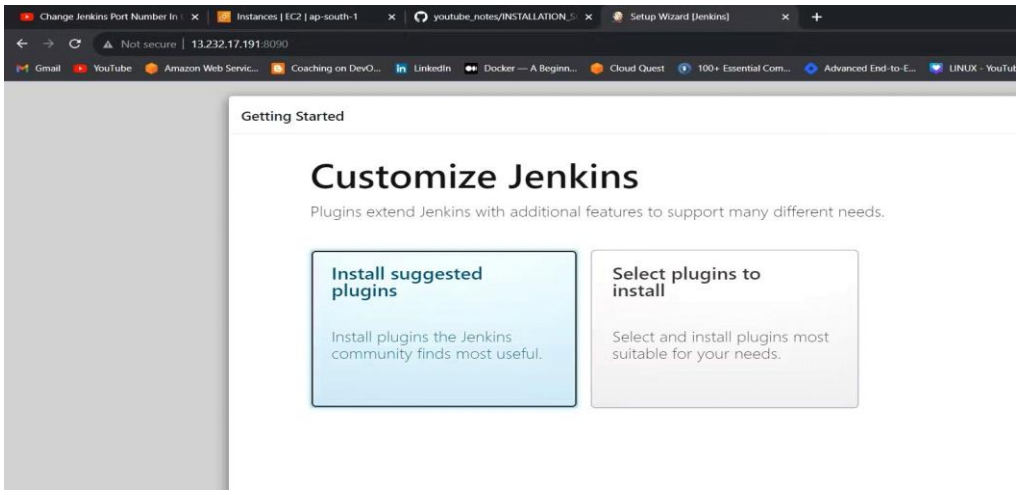
STEP-3-JENKINS:

Now copy the the Ip address and go through the Jenkins portal the interface will like shownbelow

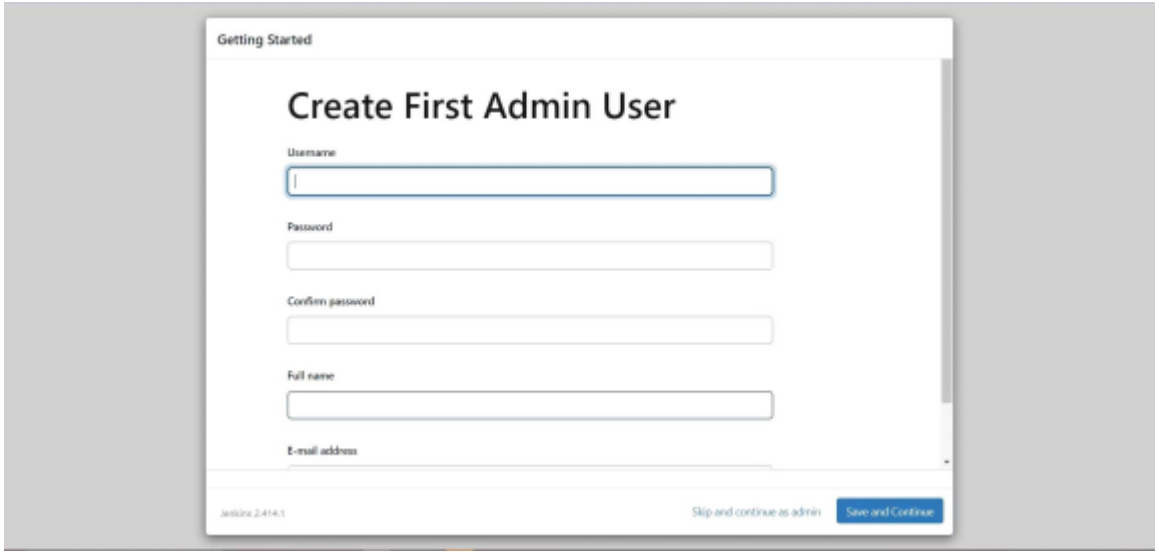


- Unlock the Jekins by using command through the given path then we will get the passwd

  cat /var/lib/Jenkins/secrets/initialAdminPassword
- Install the suggested Plugins.



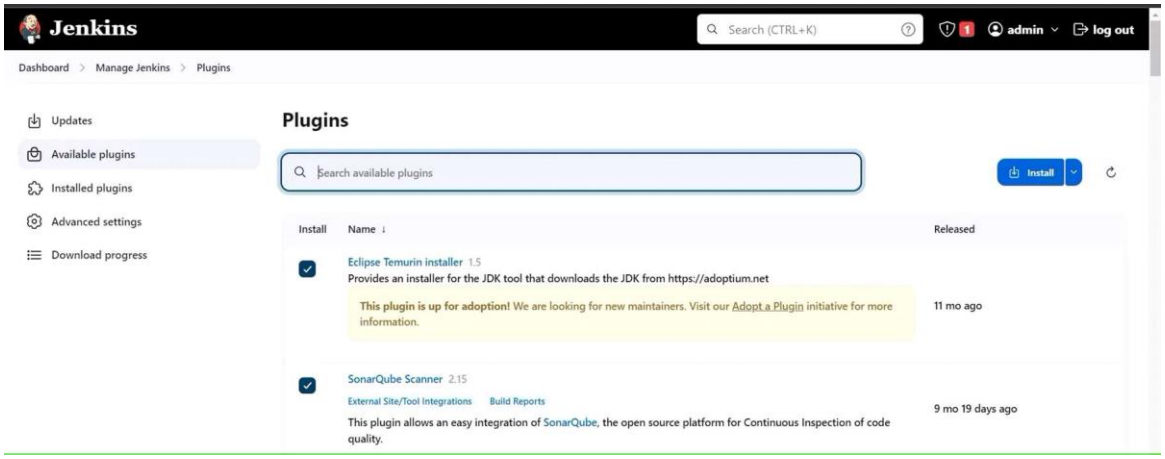
- After installing the plugins, we need to create the credentials for Jenkins.



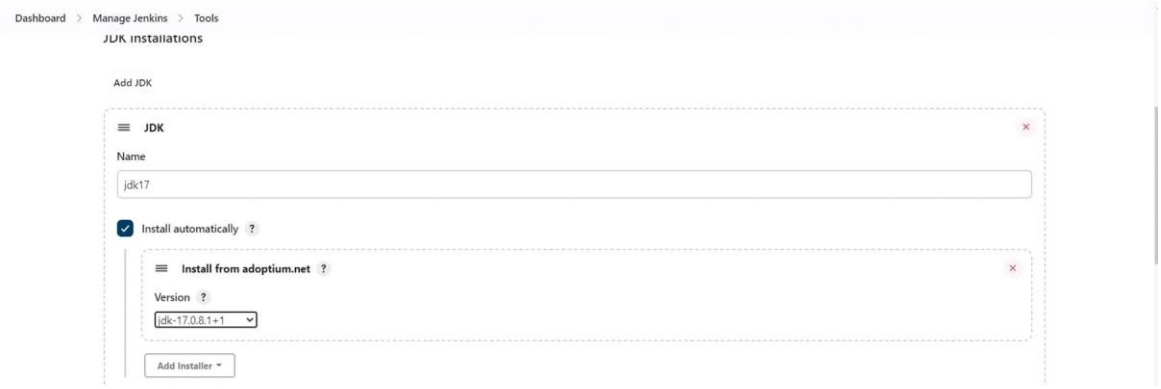
This is the Sign in page for Jenkins.

Required Plugins in The Jenkins:

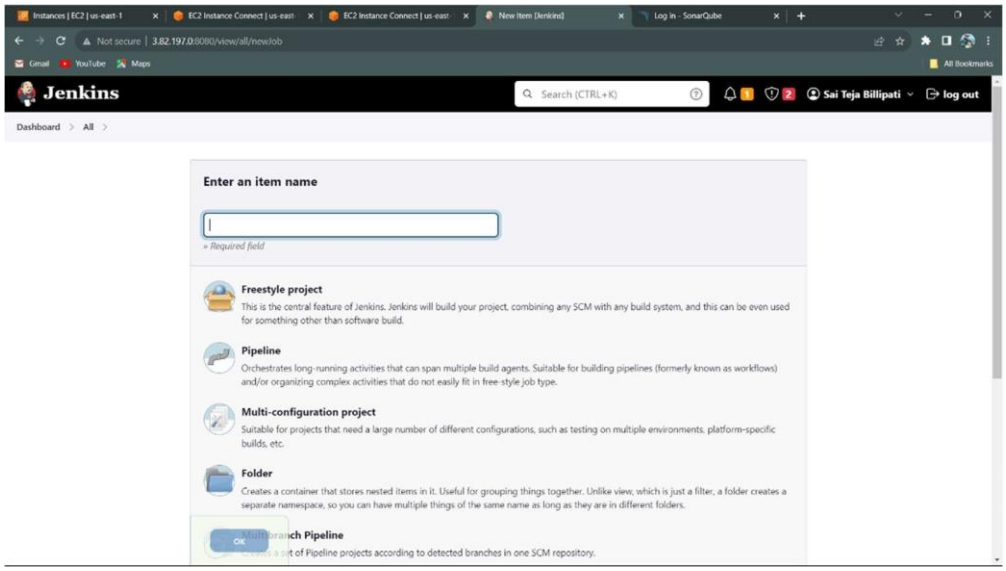
- Go to Manage Jenkins →Plugins → Available Plugins → Install below plugins:
    - Eclipse Temurin Installer
    - SonarQube Scanner

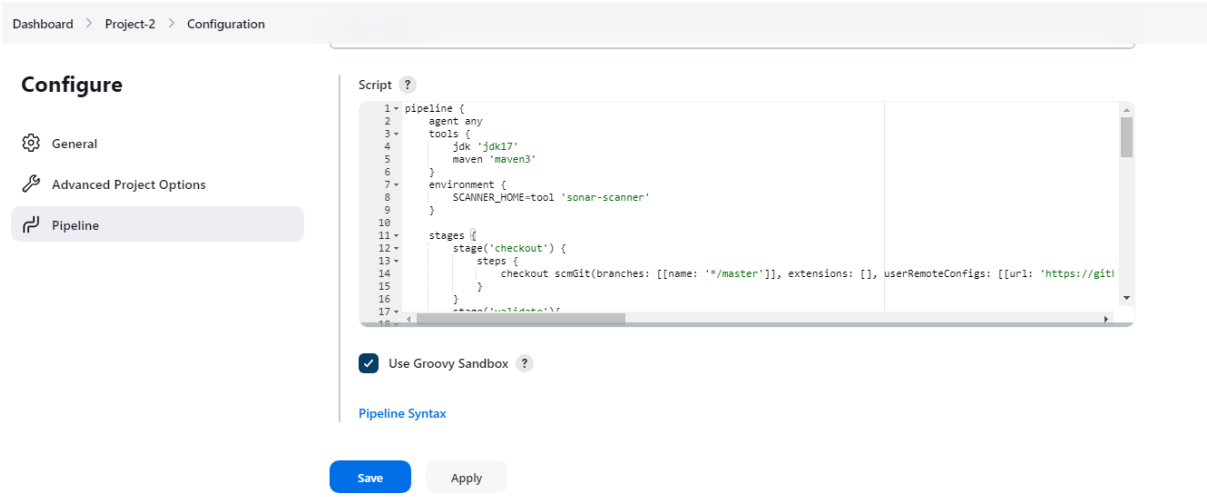


Go to Manage Jenkins → Tools → Install JDK(17.0.8.1+1) and Maven3(3.6.0) → Click onApply and Save



- Create a job by taking pipeline project and name it as pet project.



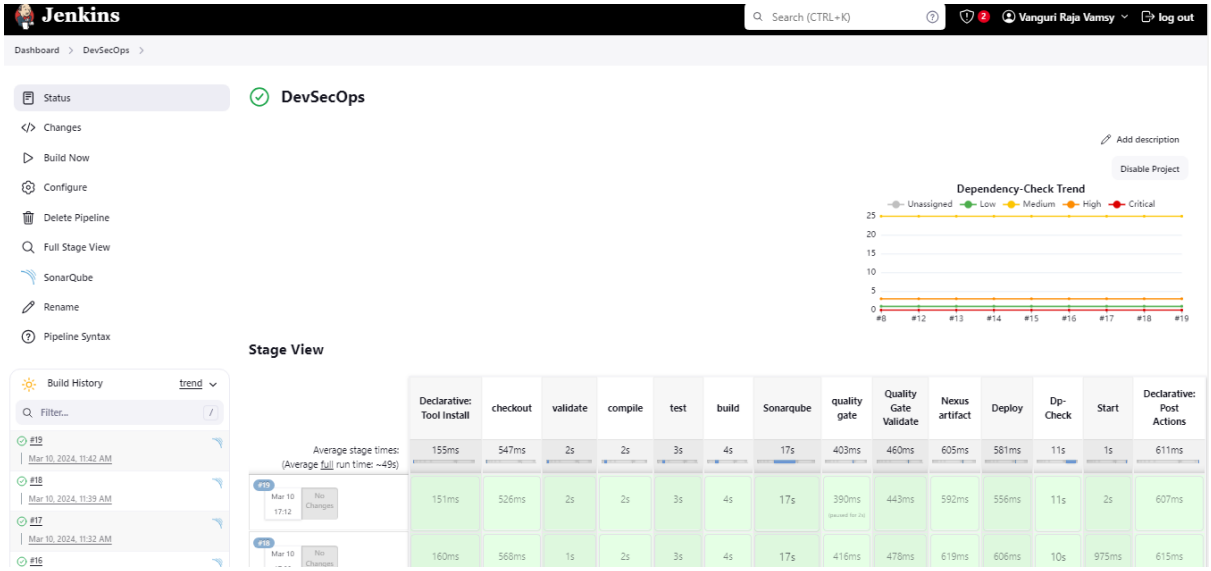


Above shows the pipeline job creation in the jenkins.

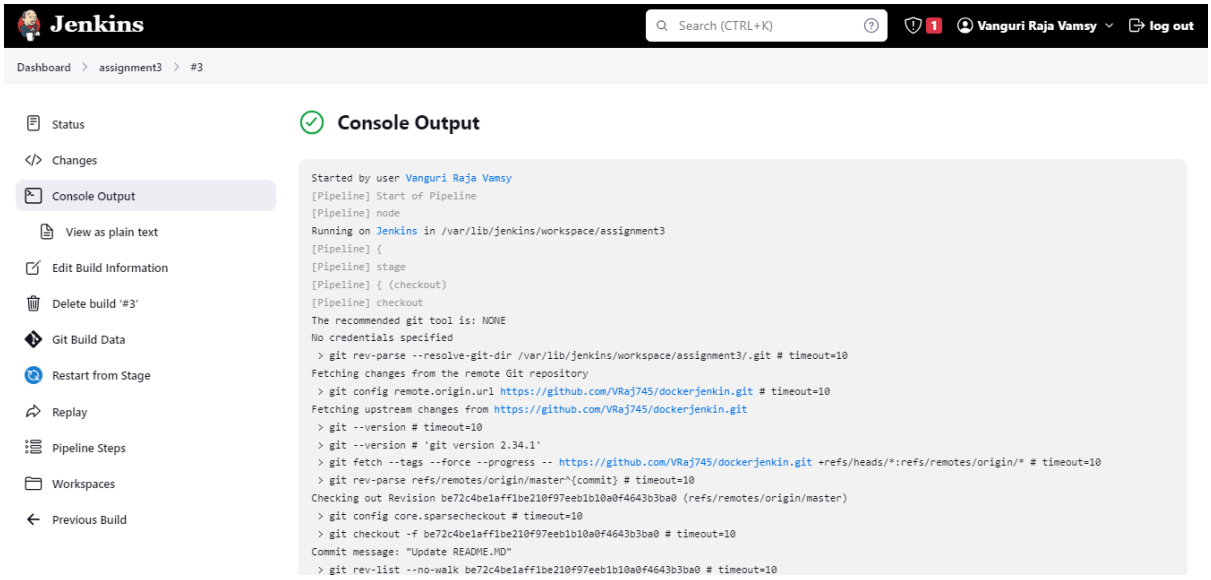
Pipeline code:

```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
        maven 'maven3'
    }
    stages {
        stage('checkout') {
            steps {
                checkout scmGit(branches: [[name: '*/master']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/VRaj745/dockerjenkin.git']])
            }
        }
        stage('validate'){
            steps{
                sh 'mvn validate '
            }
        }
        stage('compile'){
            steps{
```

```
            sh 'mvn compile'
        }
    }
    stage('test'){
        steps{
            sh 'mvn test'
        }
    }
    stage('build'){
        steps{
            sh 'mvn package'
        }
    }

    stage('Dp-Check'){
        steps{
            dependencyCheck additionalArguments: '--scan ./ --format XML ',
odcInstallation: 'Dp-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-
report.xml'
        }
    }
  }
}
```



This shows the creation and building the pipeline job and it shows the stage view.



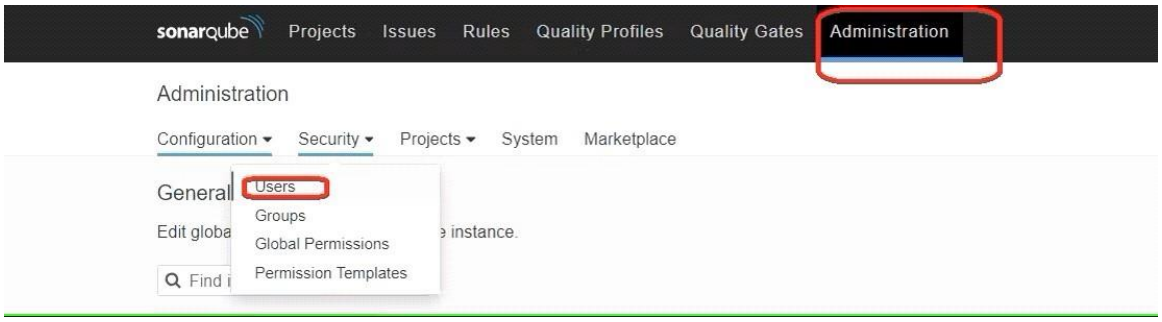
Dashboard > assignment3 > #3

```
Downloading: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/maven-metadata.xml
100 % completed (609 B / 609 B).
Downloaded: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/maven-metadata.xml (609 B at
2.6 kB/s)
Uploading: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/java-tomcat-maven-example-1.0-
20231222.153603-2.war
Uploaded: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/java-tomcat-maven-example-1.0-
20231222.153603-2.war (2.1 kB at 10 kB/s)
Uploading: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/maven-metadata.xml
Uploaded: http://3.144.16.217:8081/repository/maven-snapshots/com/example/java-tomcat-maven-example/1.0-SNAPSHOT/maven-metadata.xml (609 B at 5.5
kB/s)
Uploading artifact java-tomcat-maven-example.war completed.
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] deploy
[DeployPublisher][INFO] Attempting to deploy 1 war file(s)
[DeployPublisher][INFO] Deploying /var/lib/jenkins/workspace/assignment3/target/java-tomcat-maven-example.war to container Tomcat 9.x Remote with
context assignment3
  [/var/lib/jenkins/workspace/assignment3/target/java-tomcat-maven-example.war] is not deployed. Doing a fresh deployment.
  Deploying [/var/lib/jenkins/workspace/assignment3/target/java-tomcat-maven-example.war]
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

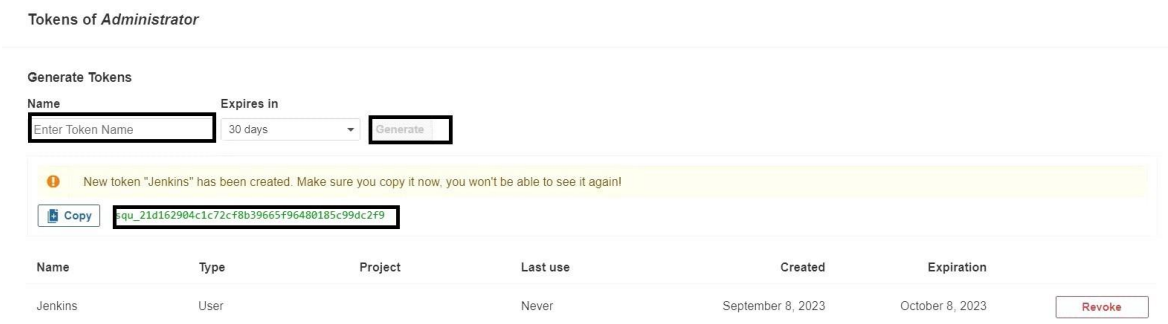
Above figures shoes the **console output** of the pipeline job.

STEP-4-SONARQUBE:

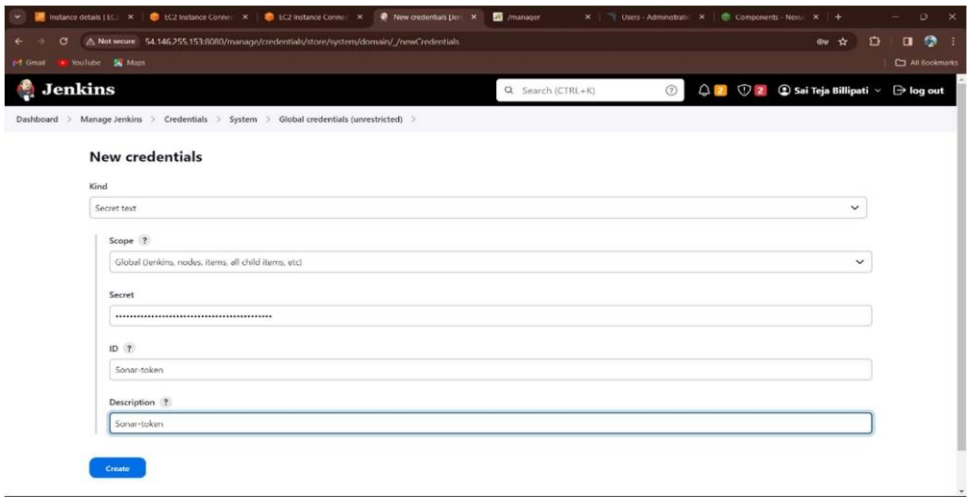
- Create a token with a name and generate.



- copy the Token.

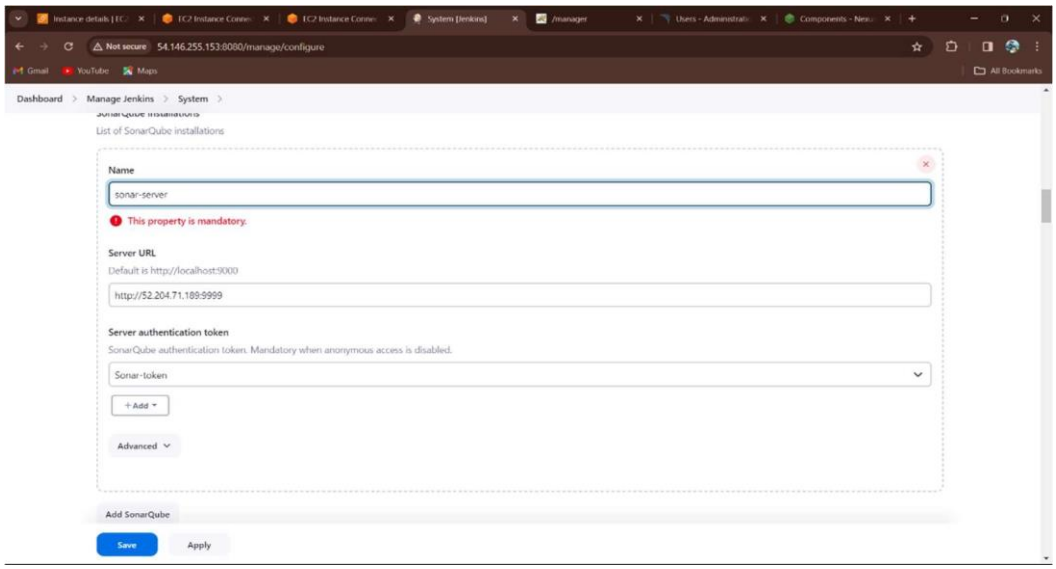


- Go to Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text. It shouldlook like this. In that secret paste the token that we copied.

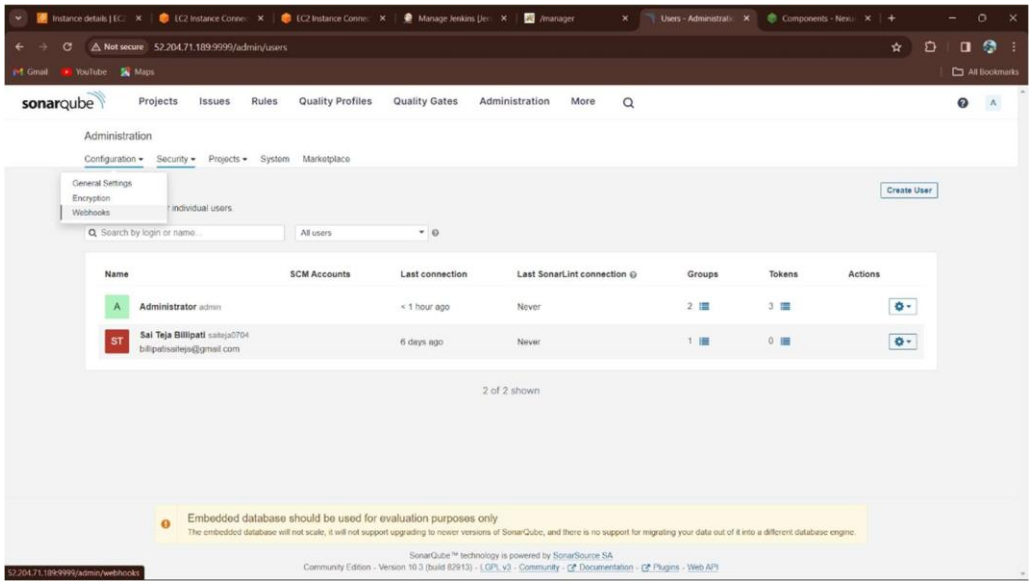


- After the creation of credentials, the interface will be like this.

- Now, go to Dashboard → Manage Jenkins → System and Add then click on apply and save.

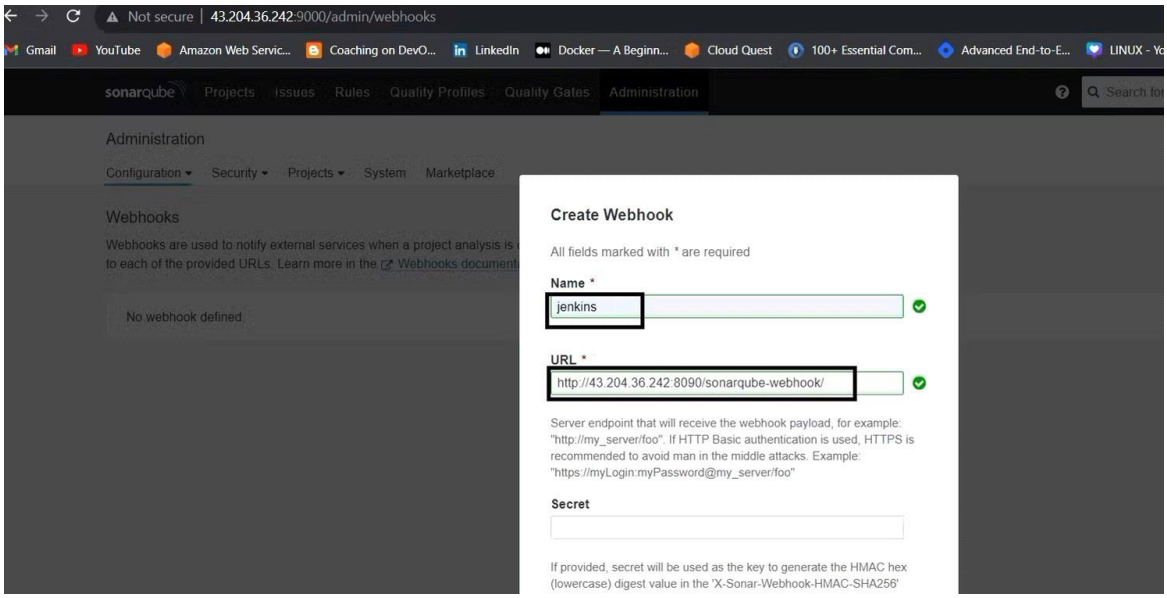


- In the SonarQube Dashboard add a quality gate also Administration--> Configuration--->Webhooks.



- Add details #in URL section of quality gate.

  <http://jenkins-public-ip:8080>/sonarqube-webhook/

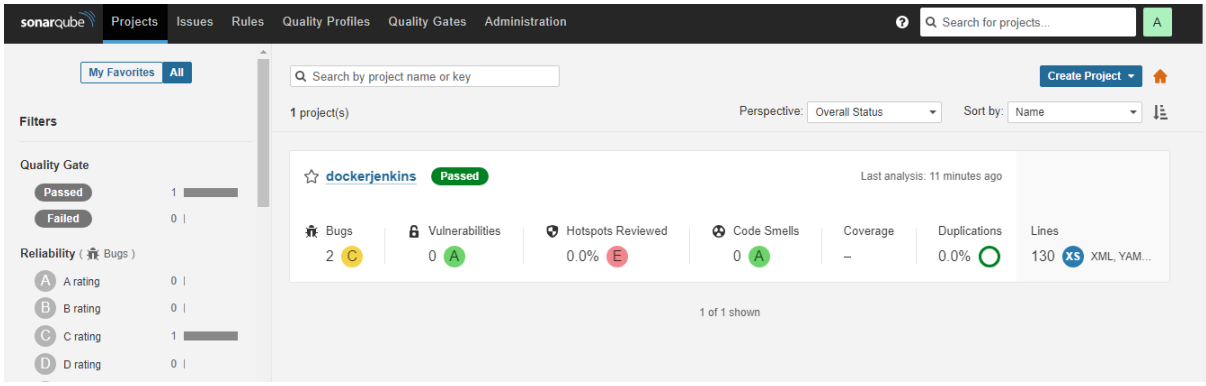


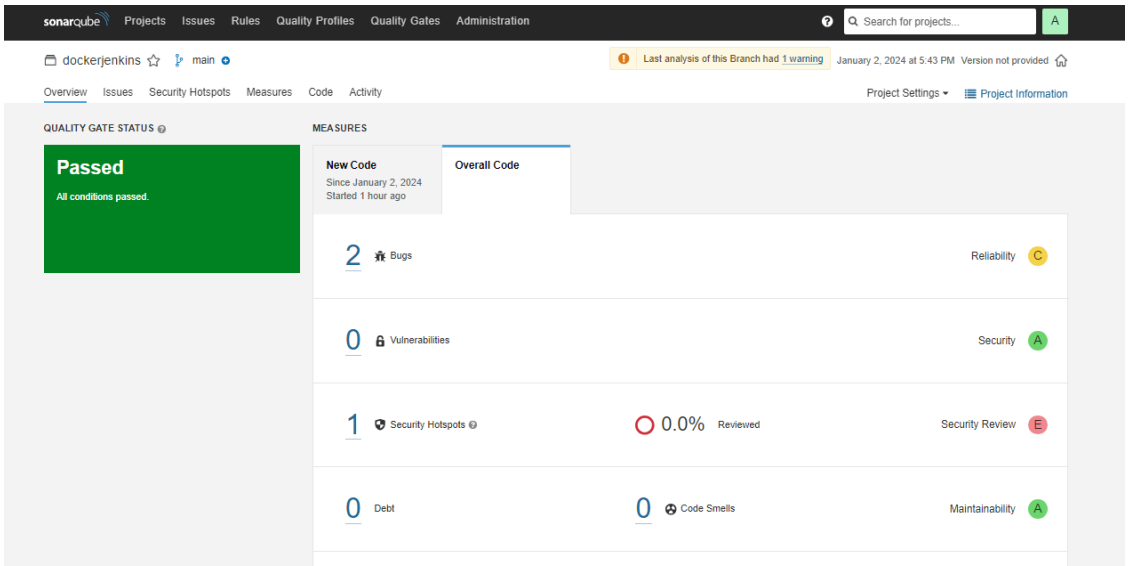
- Let's go to our Pipeline and add SonarQube Stage in our Pipeline Script.

```
environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
stage('Sonarqube'){
    steps{
```

```
            withSonarQubeEnv('sonar-server') {
            sh ''' $SCANNER_HOME/bin/sonar-scanner -
Dsonar.projectName=dockerjenkins \
            -Dsonar.java.binaries=. \
            -Dsonar.projectKey=dockerjenkins '''}
        }
    }
    stage('quality gate'){
        steps {
            script {
              waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-
token'
            }
        }
    }
```

By using the sonarqube we done the code quality analysis. It says us about the details about code which shows the vulnerabilities, bugs etc.



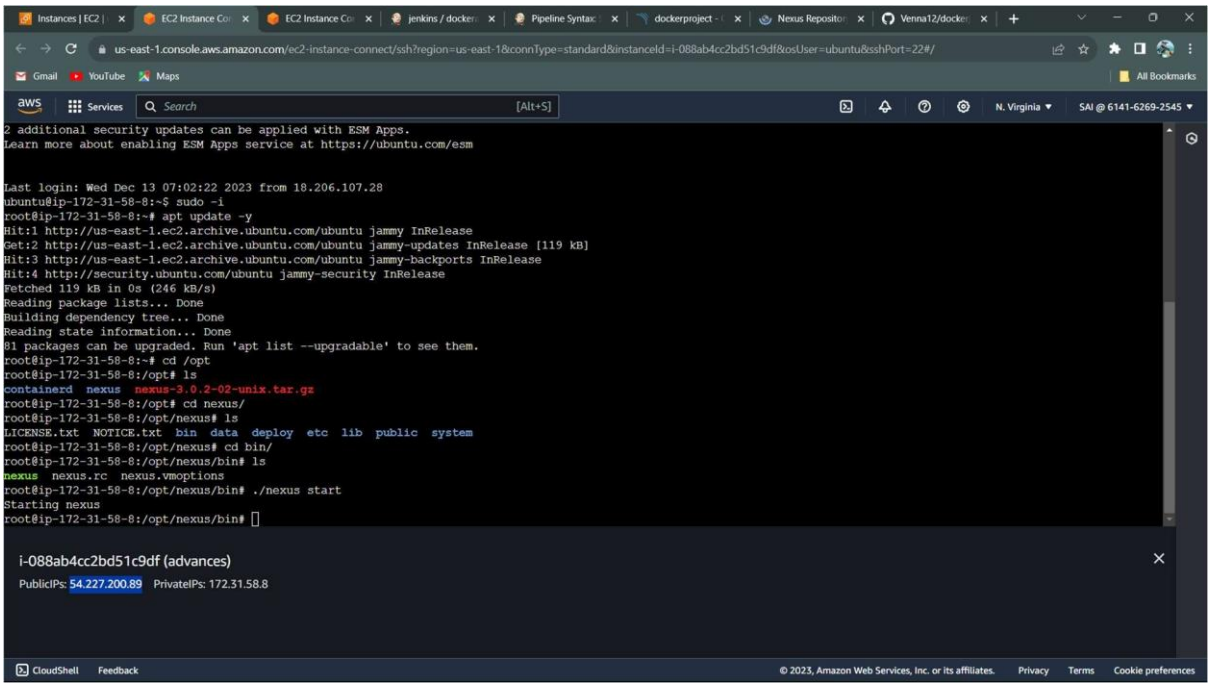


The above fig shows that the Analysis of code.

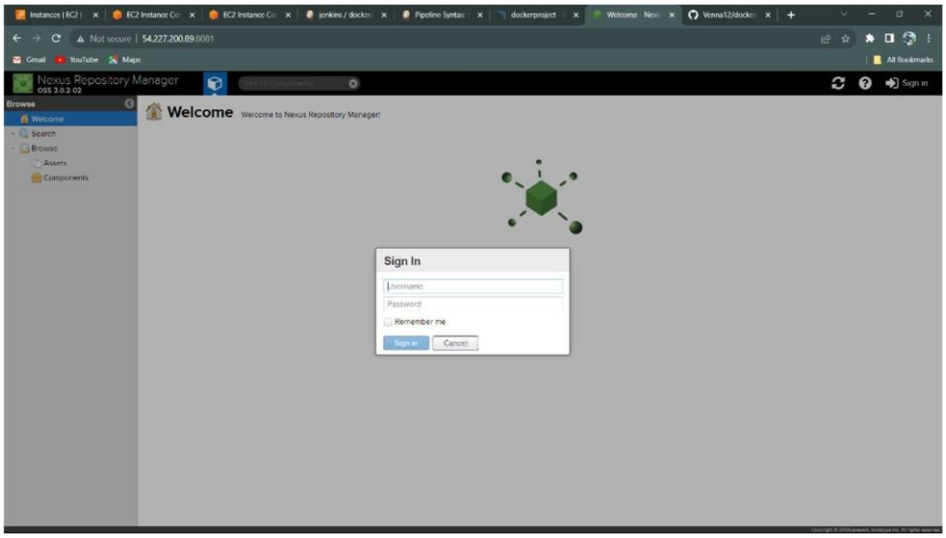
STEP-5-NEXUS:

Nexus is a Sonatype Artifactory repository manager [OSS]. It allows you to store, distribute, and retrieve build artifacts whenever it's required. Using Nexus, developers can easily access and deploy build artifacts in an organization from a single location.

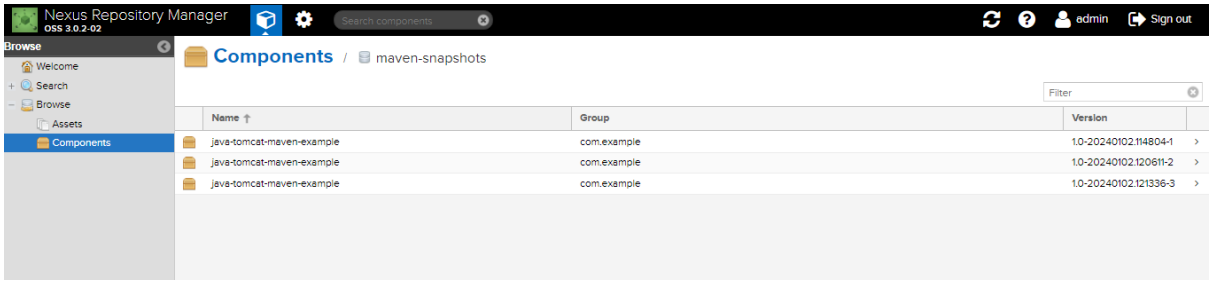
- Install the nexus and un-tar it as below process



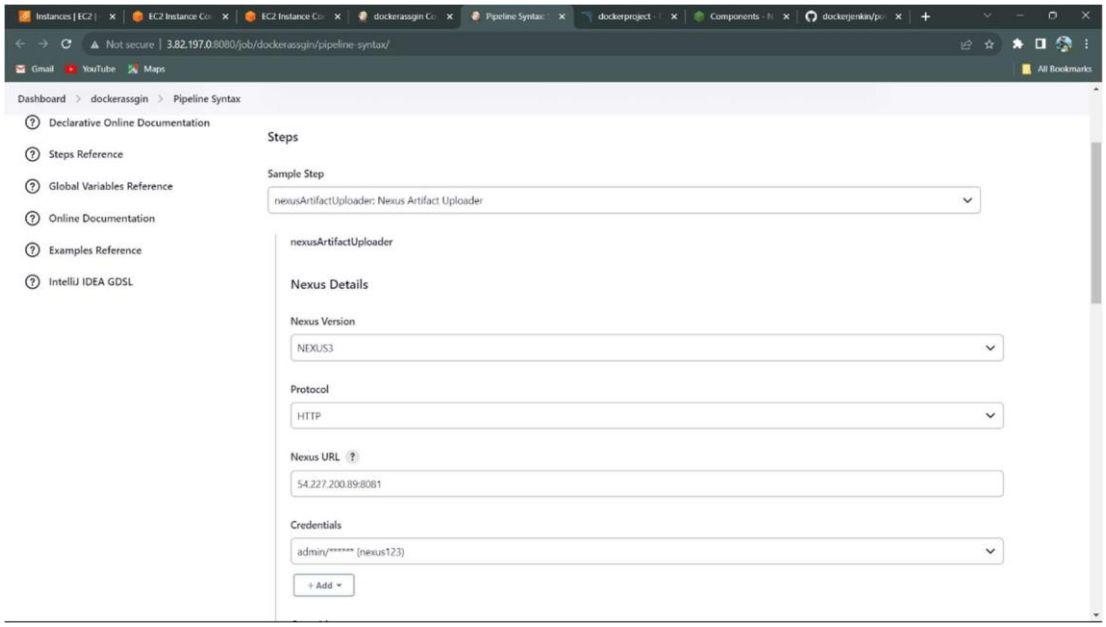
- And login through the nexus **USER:admin & PASSWD:admin**.



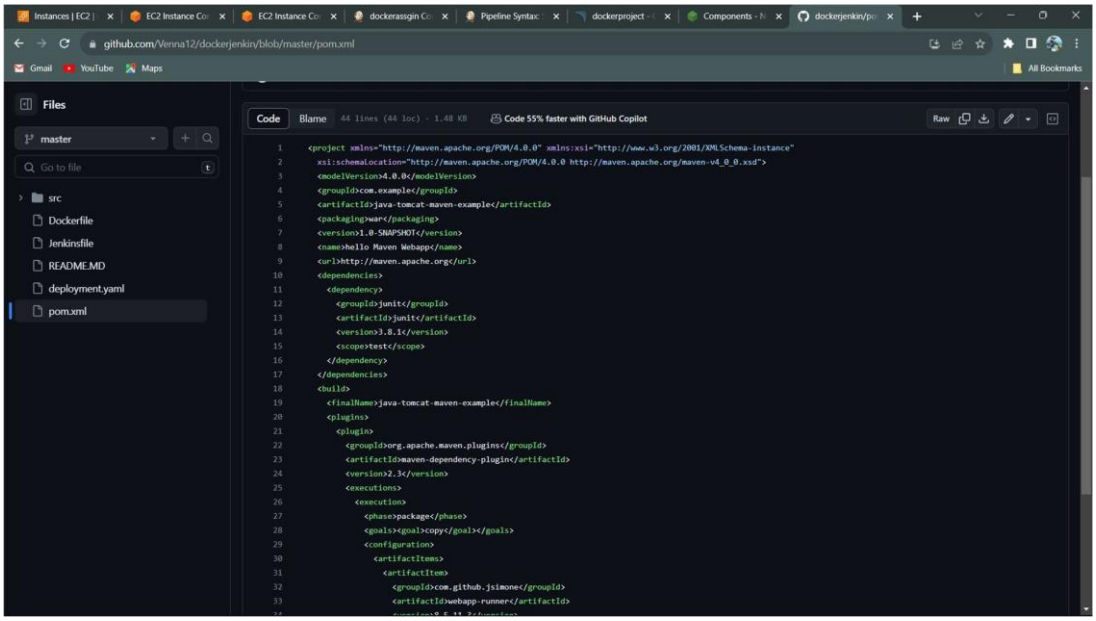
- Go to > Components > maven-Snapshots!



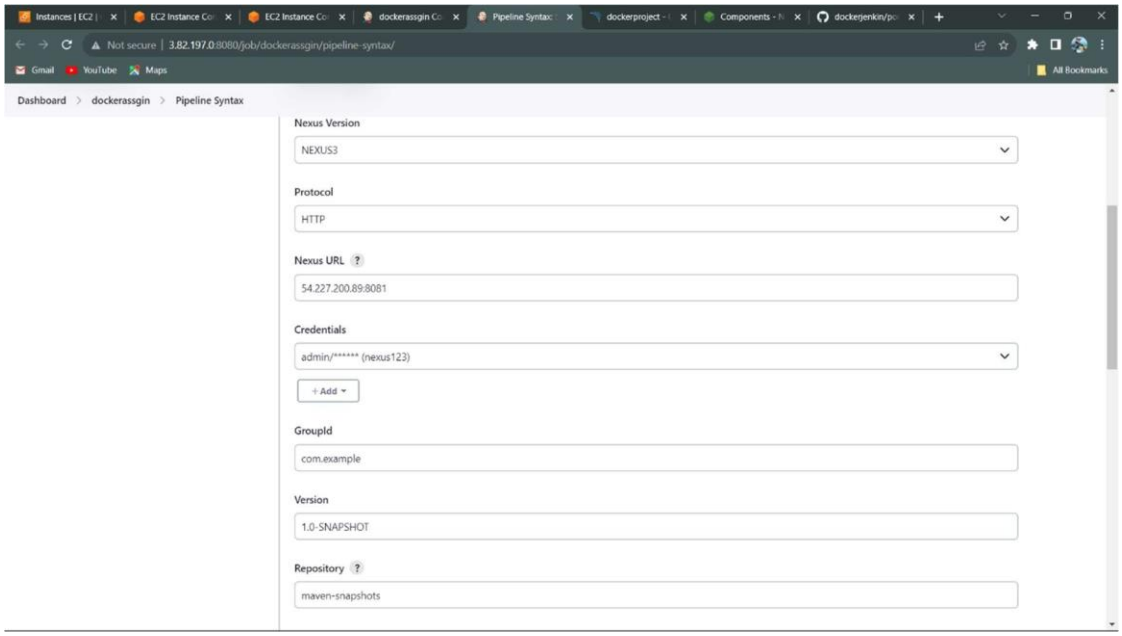
- Now go through the Jenkins and in the pipeline, syntax select the **Nexus artifact Uploder.**
- Fill it in accordance through that required one with the current Ip address of the server



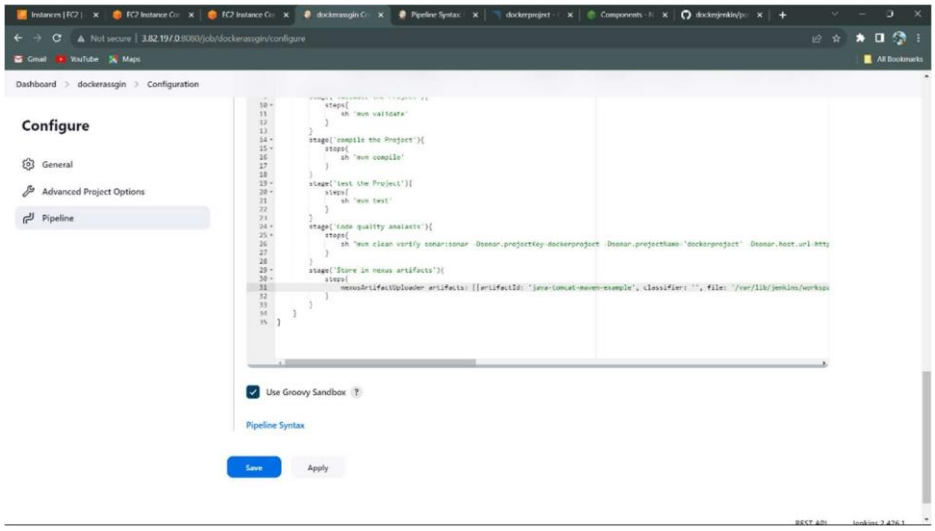
- For the Group-id, versions we can see that in the project of the **pom.xml.**



- Now we have a below Artifact we have group-id, version, and repository all we can by theport.xml



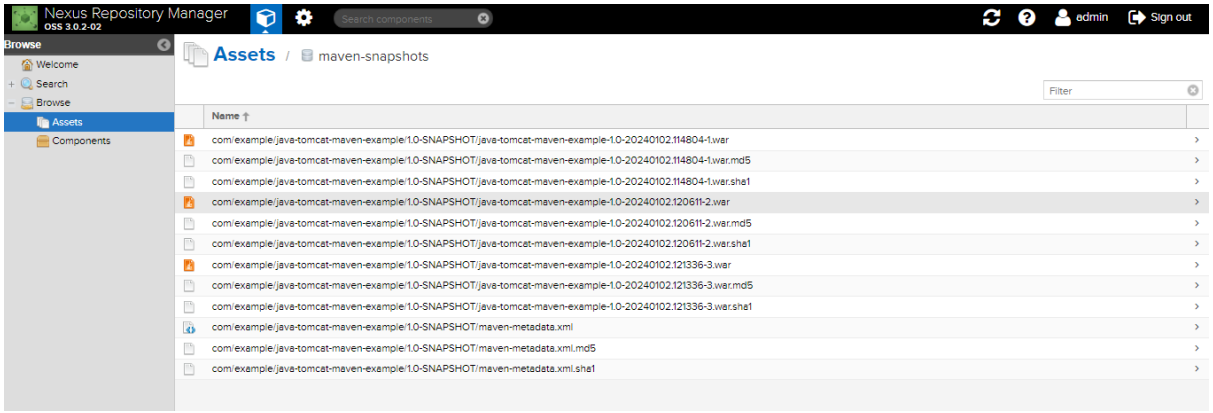
- Write the pipeline syntax for the nexus artifact.




```
stage('Nexus artifact'){
    steps{
        nexusArtifactUploader artifacts: [[artifactId: 'java-tomcat-maven-example', classifier: '', file: '/var/lib/jenkins/workspace/Project-2/target/java-tomcat-maven-example.war', type: 'war']], credentialsId: 'nexus123', groupId: 'com.example', nexusUrl: '13.59.143.13:8081', nexusVersion: 'nexus3', protocol: 'http', repository: 'maven-snapshots', version: '1.0-SNAPSHOT'
    }
}
```

- And build it.

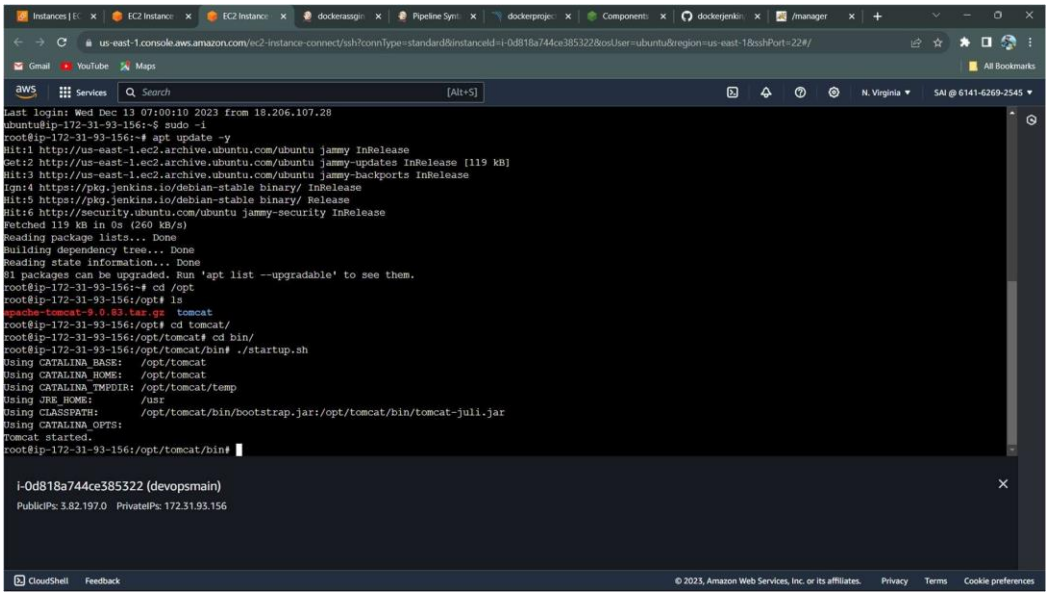
- The required artifact will be generated.



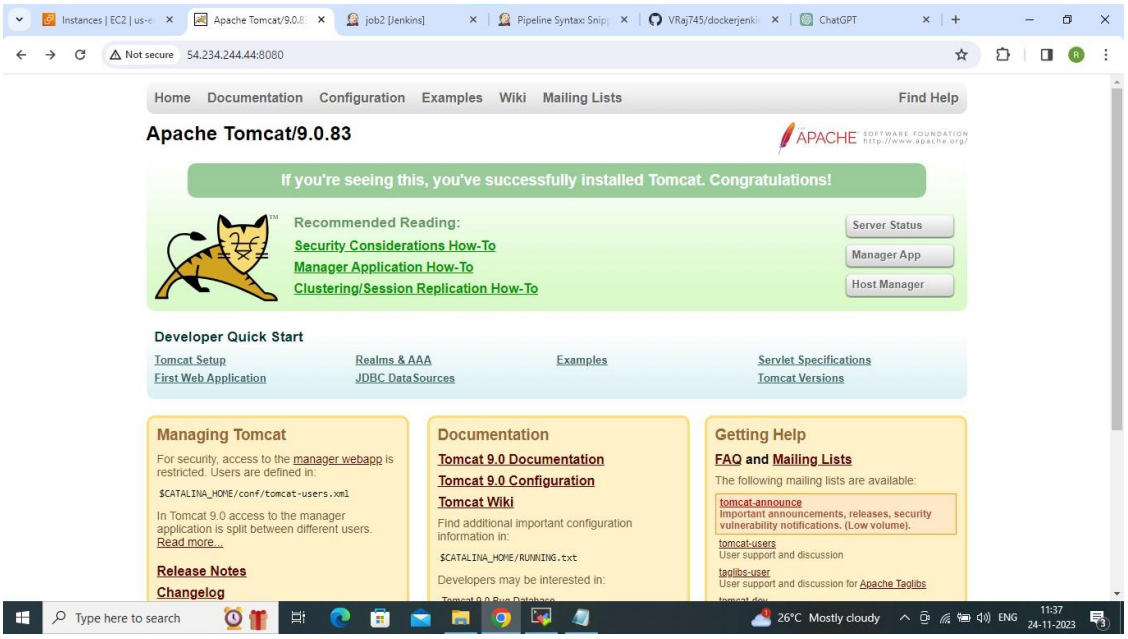
Above fig shows the nexus artifacts.

STEP-6-TOMCAT:

- Installing the tomcat in CD/OPT. Un-taring the tomcat and moving the file to the tomcat fileand run the tomcat.



- The tomcat interface will be like this when we run it.



- Above fig shows that the open page of the TOMCAT server. By clicking on the manager app button, it takes us into the below view of the web page.

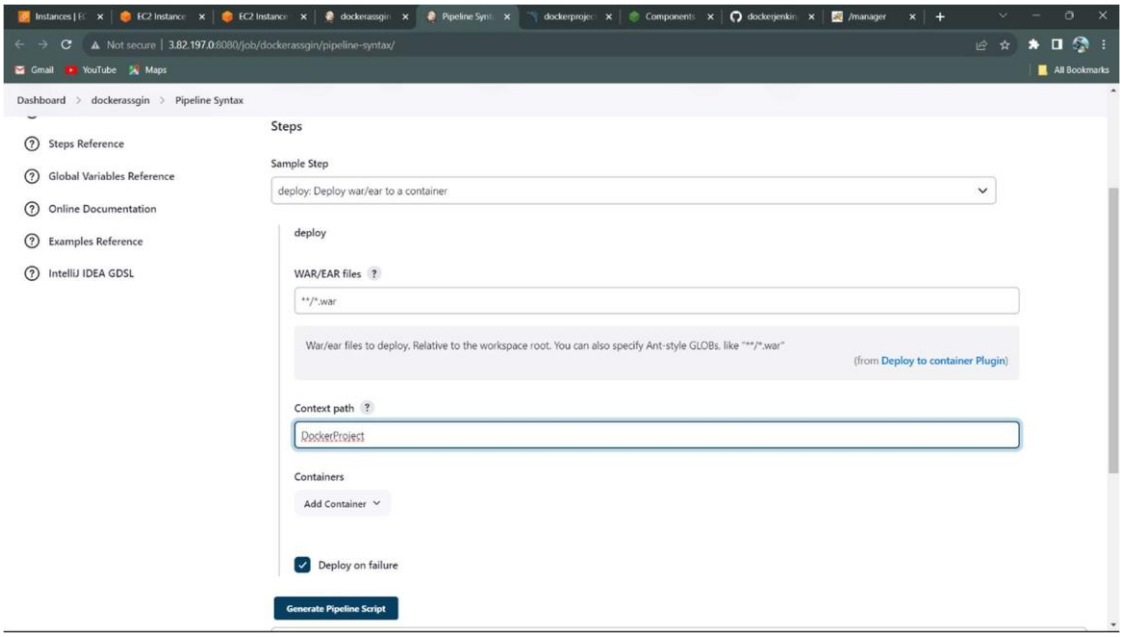
Tomcat Web Application Manager

Message:	OK

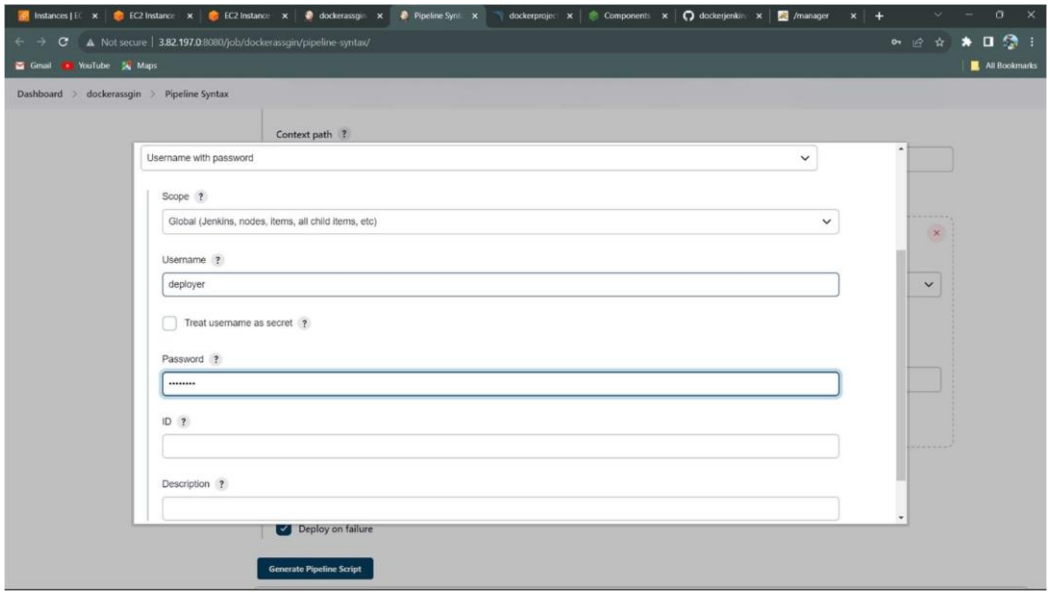
Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/PROJECT-3	None specified	Archetype Created Web Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

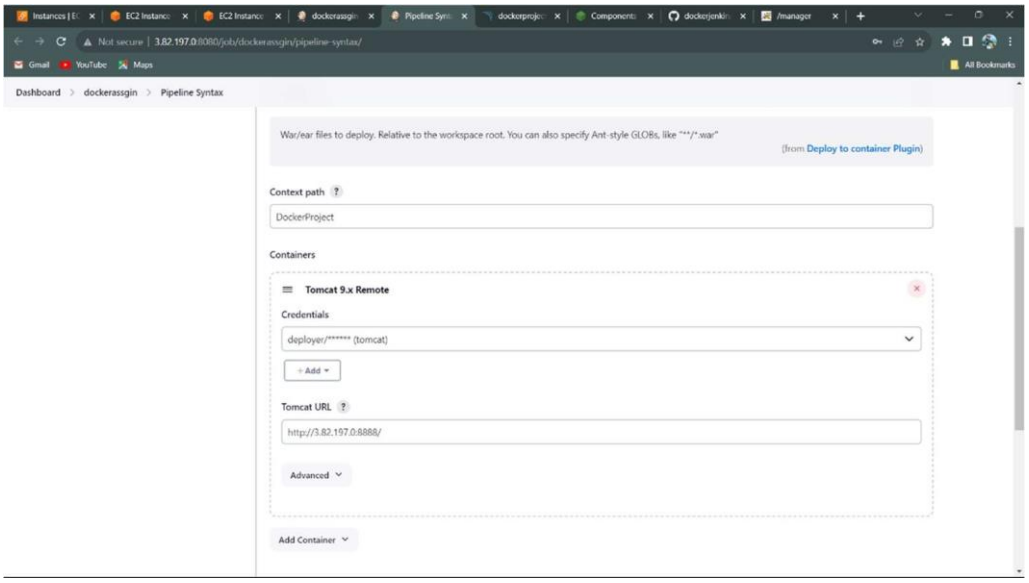
- Now go to Jenkins pipeline syntax and select the deploy the container and the context pathname it as project-3.



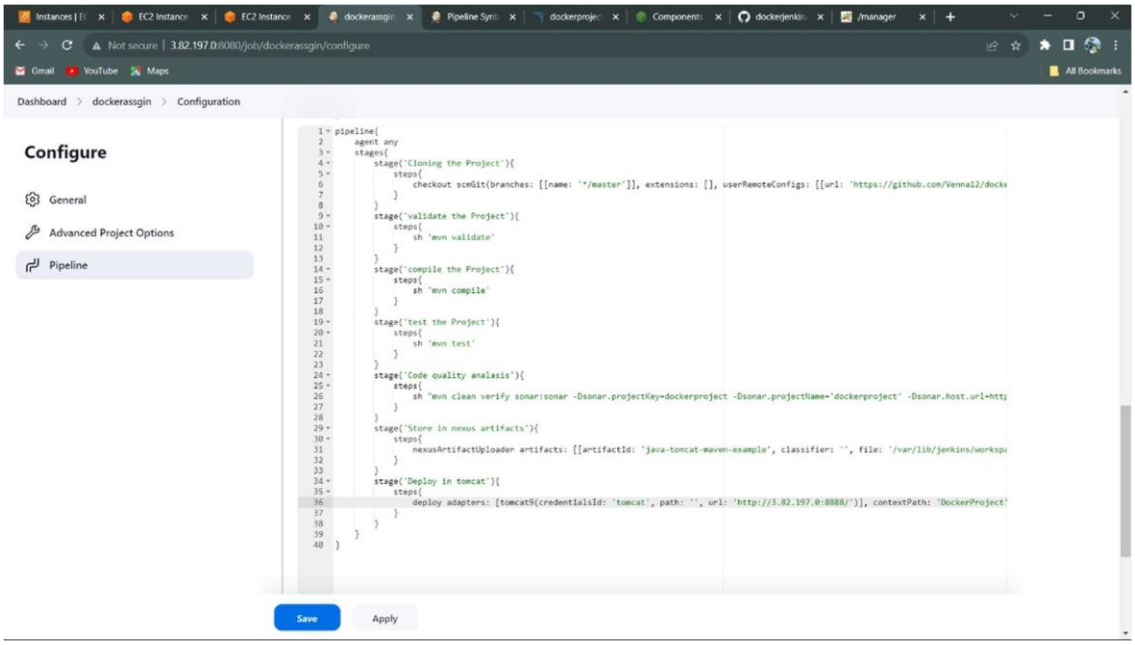
- Now go through the add container select the USERNAME AND PASSWORDS and give the tomcat credentials add and save.



- Select the TOMCAT9 container. add the credential's and the URL of the tomcat.



- Generate the pipeline for the deployment of tomcat.



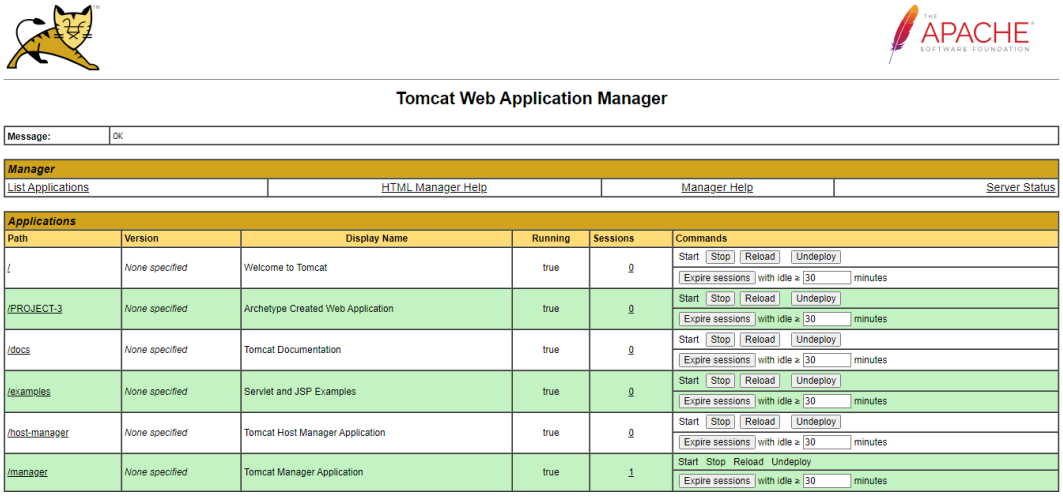
stage('Deploy'){

steps{

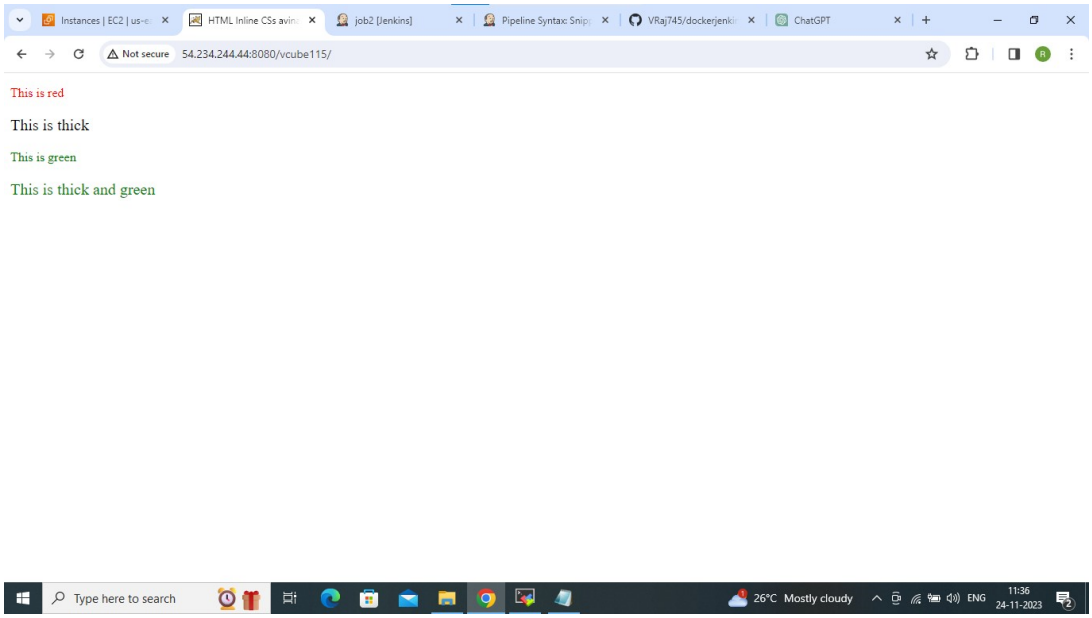
deploy adapters: [tomcat9(credentialsId: '52c24633-b9e7-4df0-b8aa-0fb09b4aa831', path: '', url: 'http://13.59.143.13:8086/')], contextPath: 'PROJECT-3', war: '**/*.war'

}

}



- In this page we can see the deployed webapp in the list as 'Project-3'.



- This is the Final output of the Project.

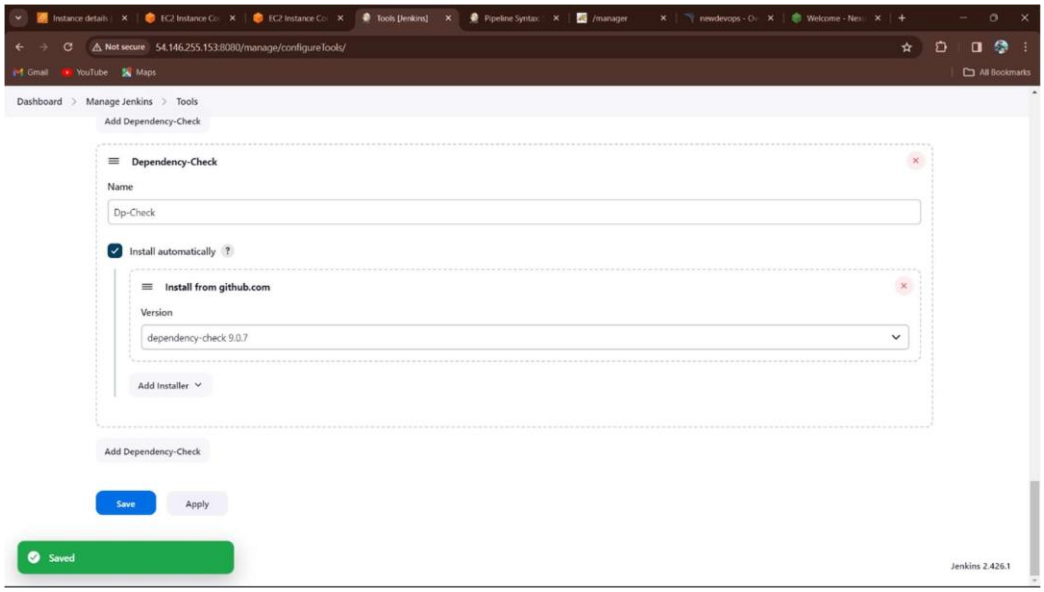
STEP-7-DP-CHECK:

Dependency-check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities. This tool can be part of the solution to the OWASP top 10 2017: a9 - using components with known vulnerabilities.

Install OWASP Dependency Check Plugins:

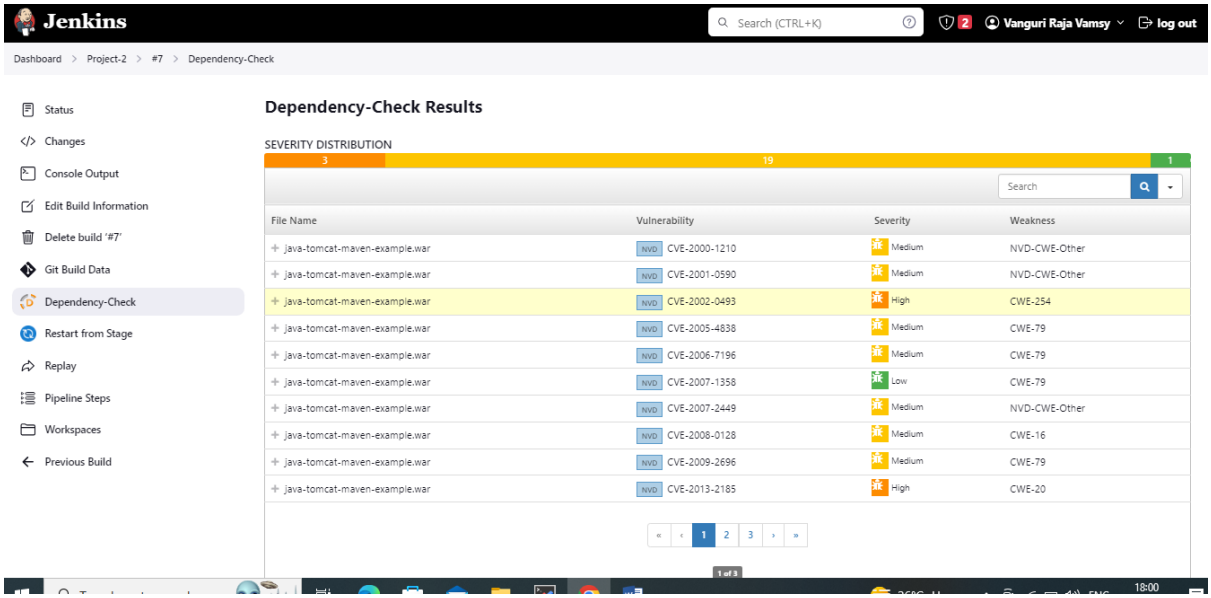
Go to Dashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click onit and install it.

- Go to Dashboard → Manage Jenkins → Tools → Click on Apply and Save here.



- Now go configure → Pipeline and add this stage to your pipeline and build

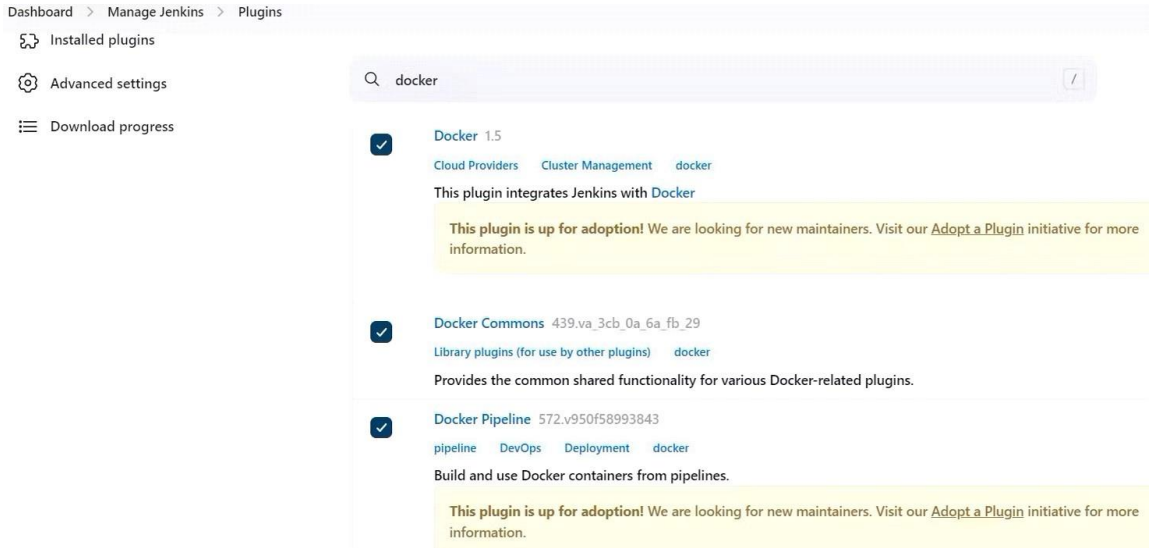
```
stage("OWASP Dependency
Check"){steps{

dependencyCheck additionalArguments: '--scan ./ --
format XML ',odcInstallation: 'DP-Check'

dependencyCheckPublisher pattern: '**/dependency-check-report.xml'

        }

}
```



STEP-8-DOCKER:

We need to install the Docker tool in our system, Goto Dashboard → Manage Plugins →Available plugins → Search for Docker and install these plugins.

- Docker
- Docker commons
- Docker pipeline
- Docker API
- Docker-Build-Step.



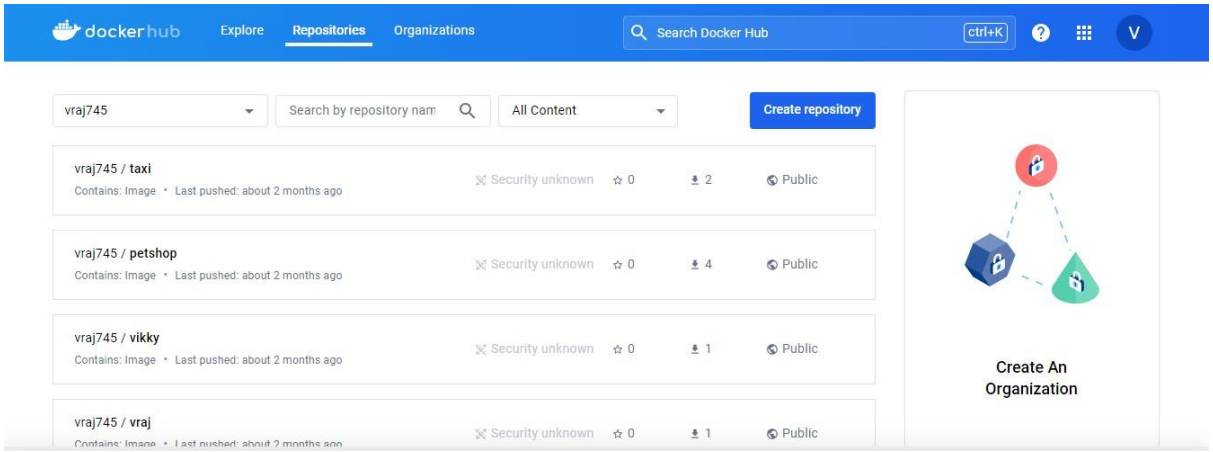
- Now, go to Dashboard → Manage Jenkins → Tools →

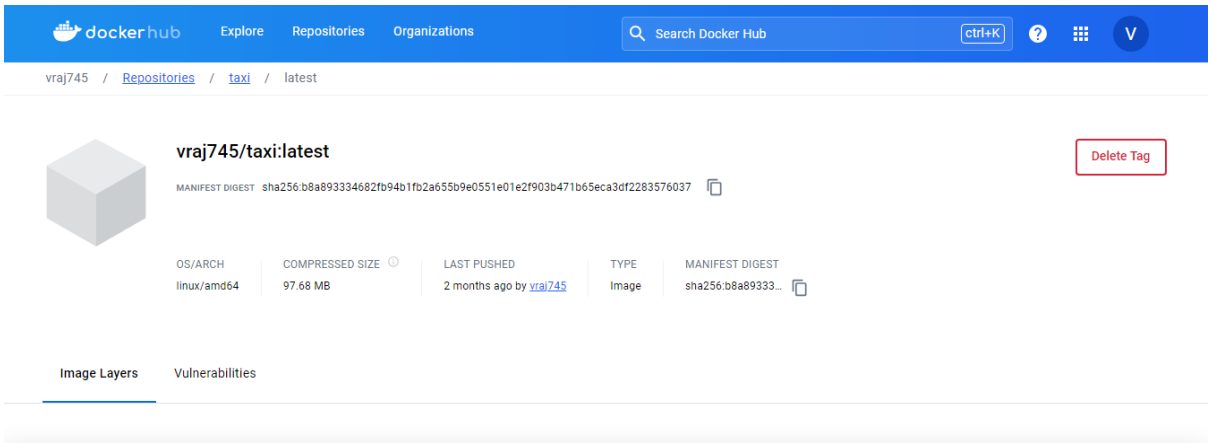
- Add Docker Hub Username and Password under Global Credentials

- Add this stage to Pipeline Script

```
stage ('Build and push to docker hub'){
  steps{
        script{
        withDockerRegistry(credentials: 'docker', toolName: 'docker') {
        sh "docker build -t taxi."
        sh "docker tag taxi vraj745/taxi:latest"
        sh "docker push vraj745/taxi:latest"
        }
      }
    }
}
stage ('Deploy to container'){
    steps{
        sh 'docker run -d --name pet1 -p 8080:8080 vraj745/taxi:latest'
        }
}
```

- When we login in the Docker we can see a new image has been added.





Now the Final Output will be seen by using the

`<Ec2-public-ip:8082/taxi>`