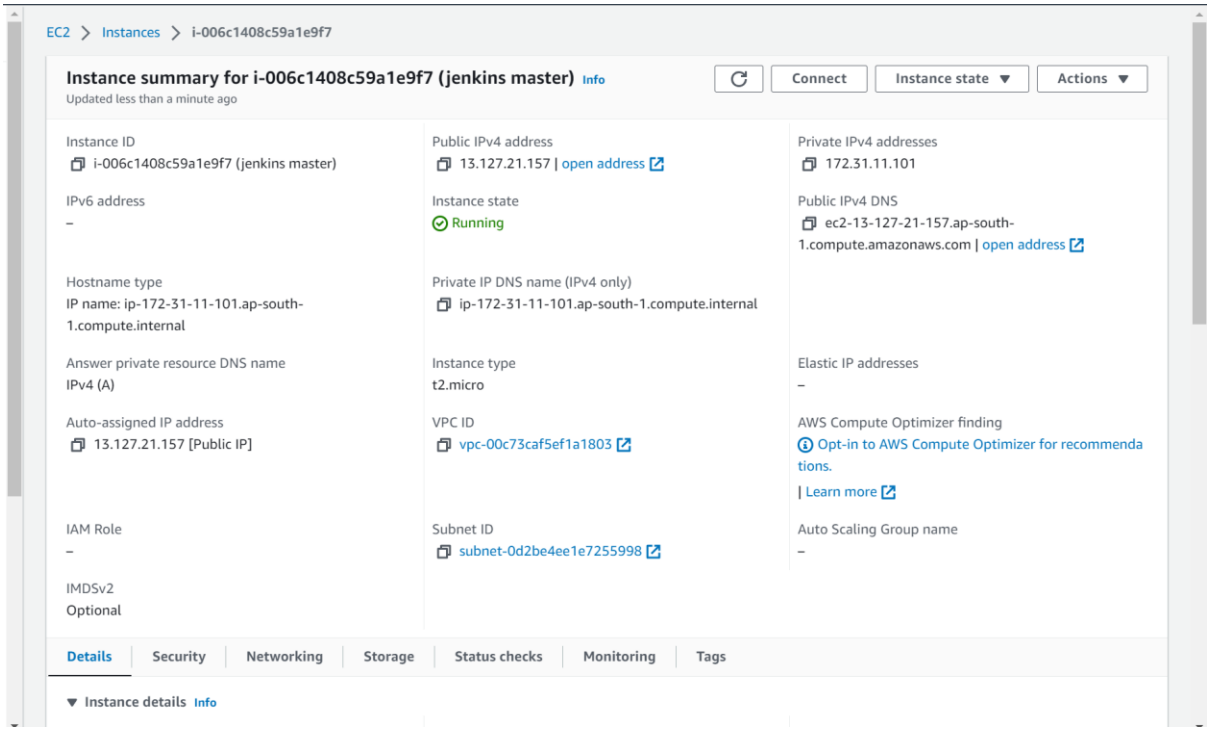
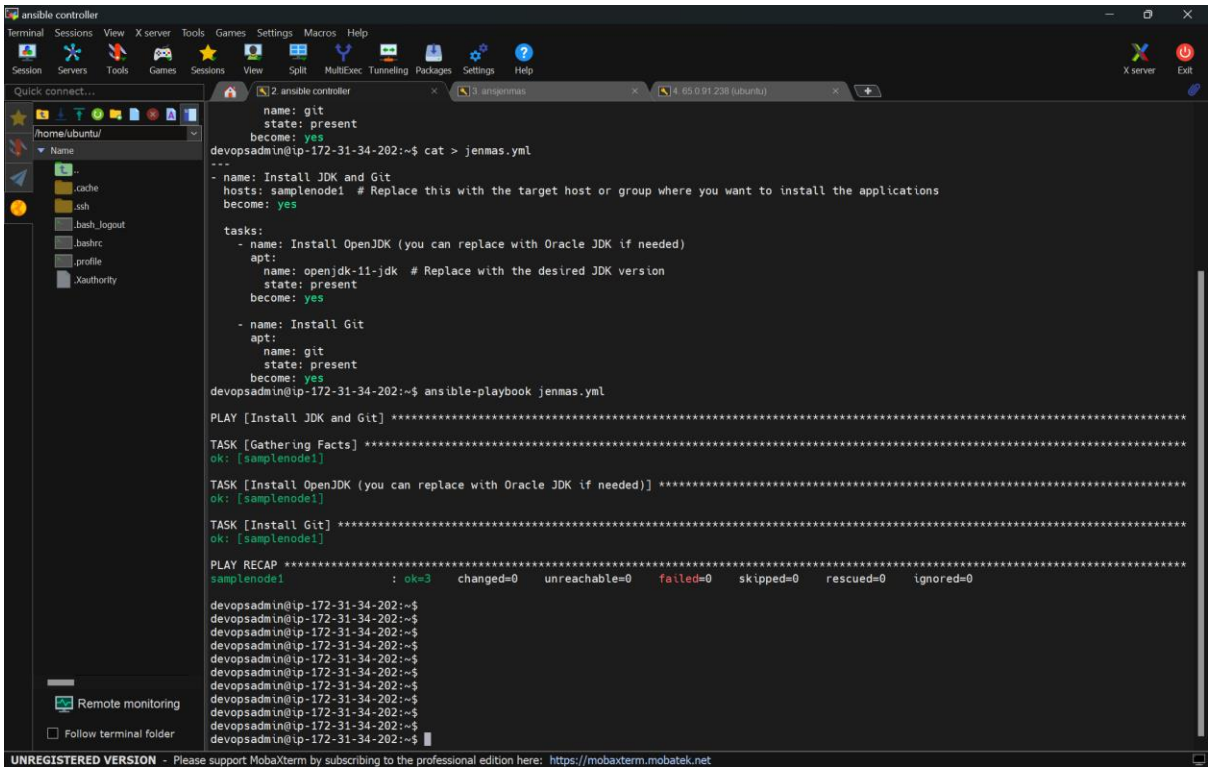
Summary: Creating an end-to-end CI/CD pipeline for Java application using Jenkins, Maven, Docker, Ansible and deployment using Kubernetes.

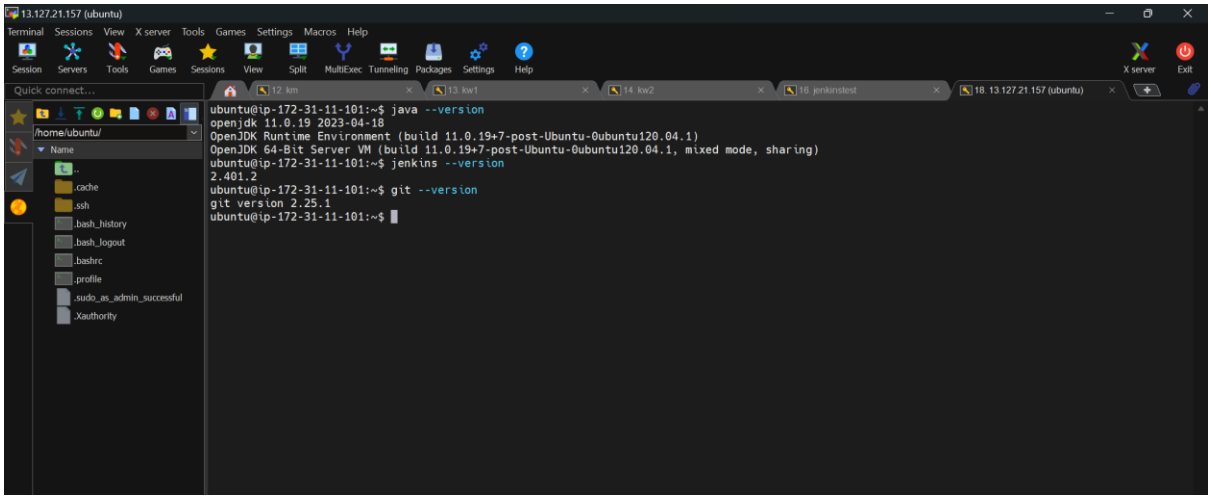
Creating EC2 instances for Jenkins



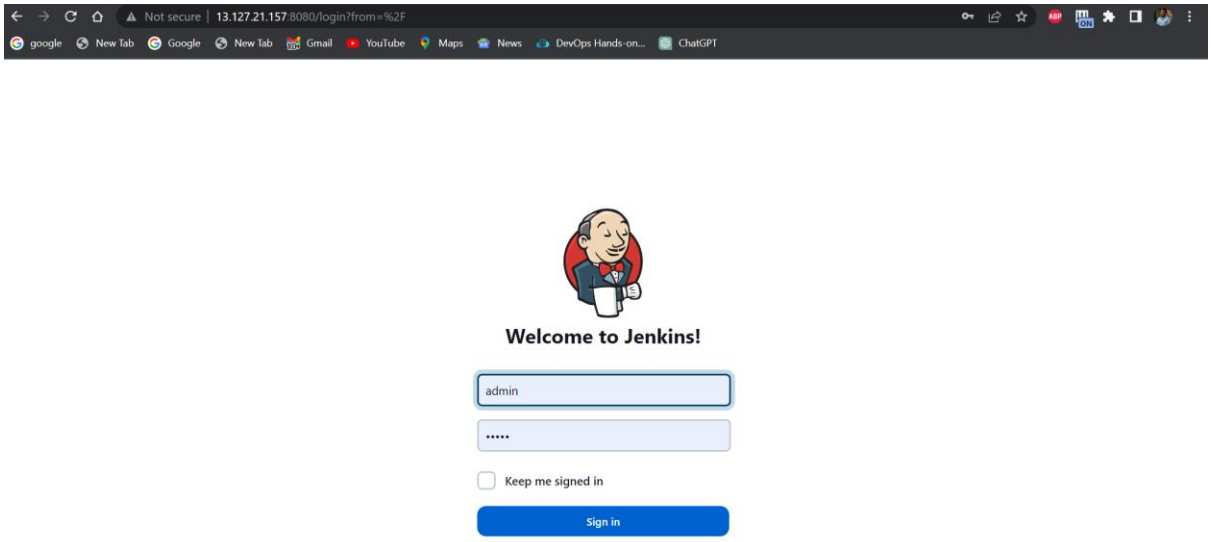
Installing Java, Git through Ansible in Jenkins Master.



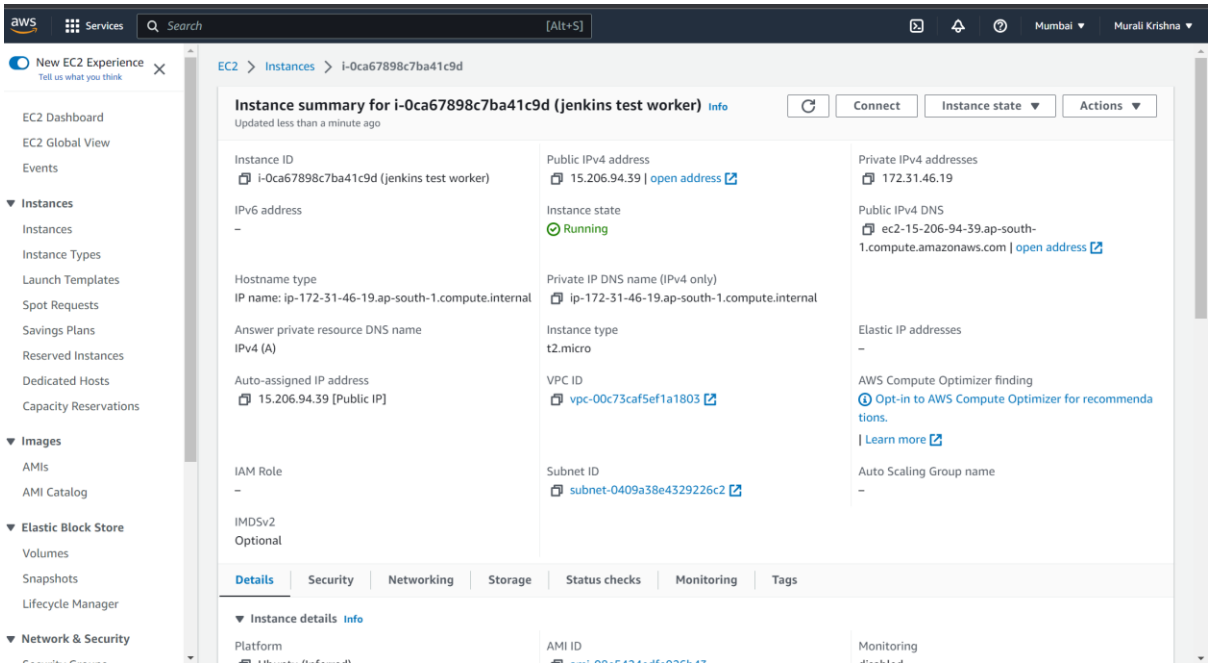
Checking the versions of the installed packages for confirmation.



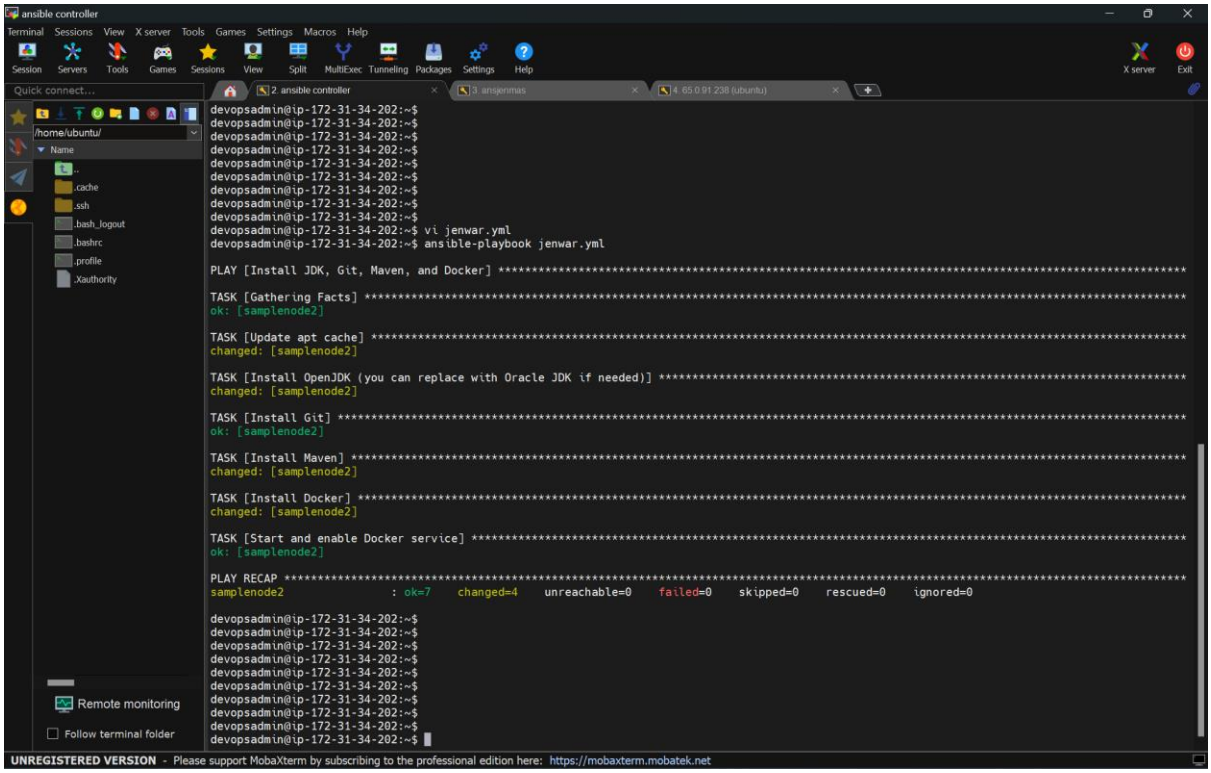
Entering into Jenkins web page



Launching EC2 instance which will be used as Jenkins working node

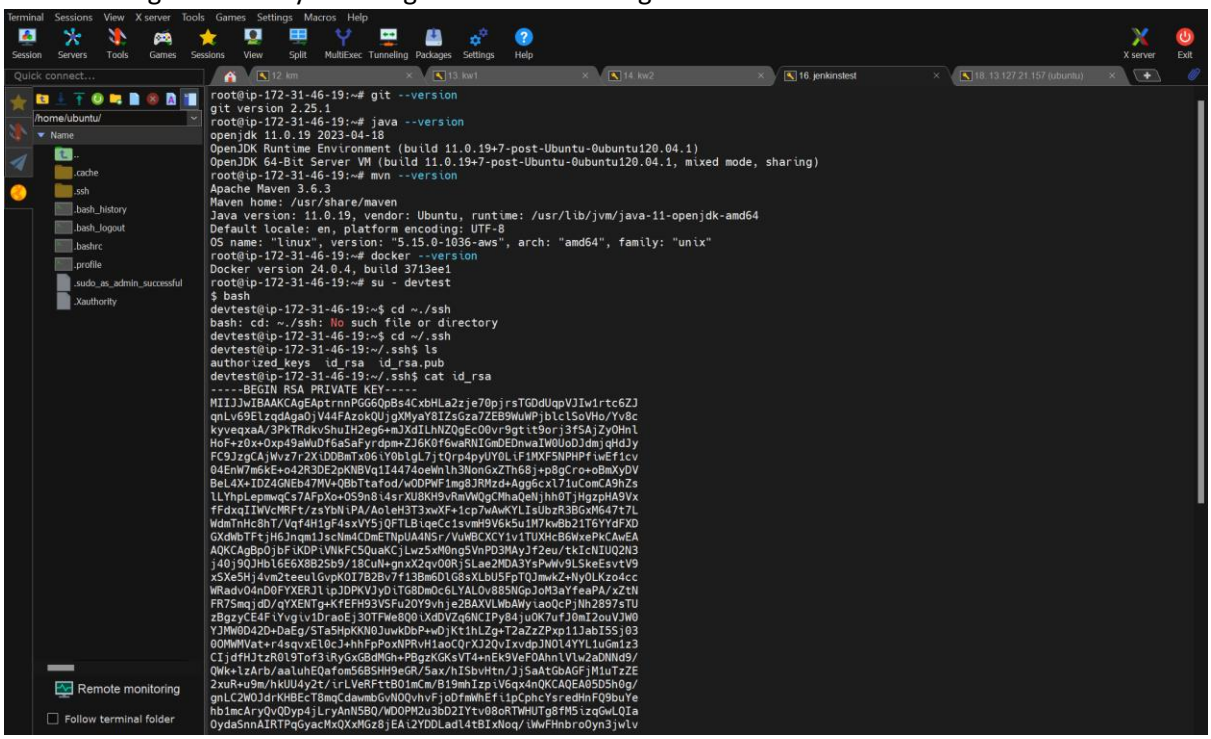


Installing the JDK, GIT, MAVEN and Docker packages through Ansible.

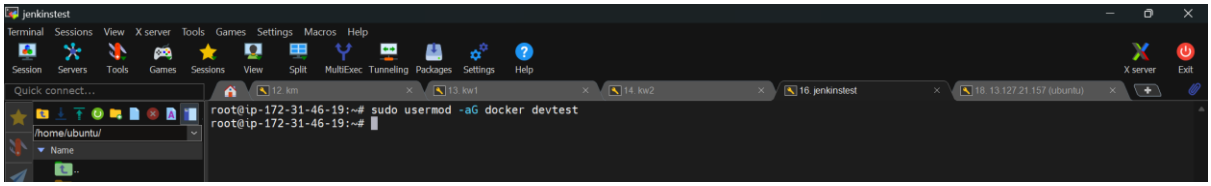


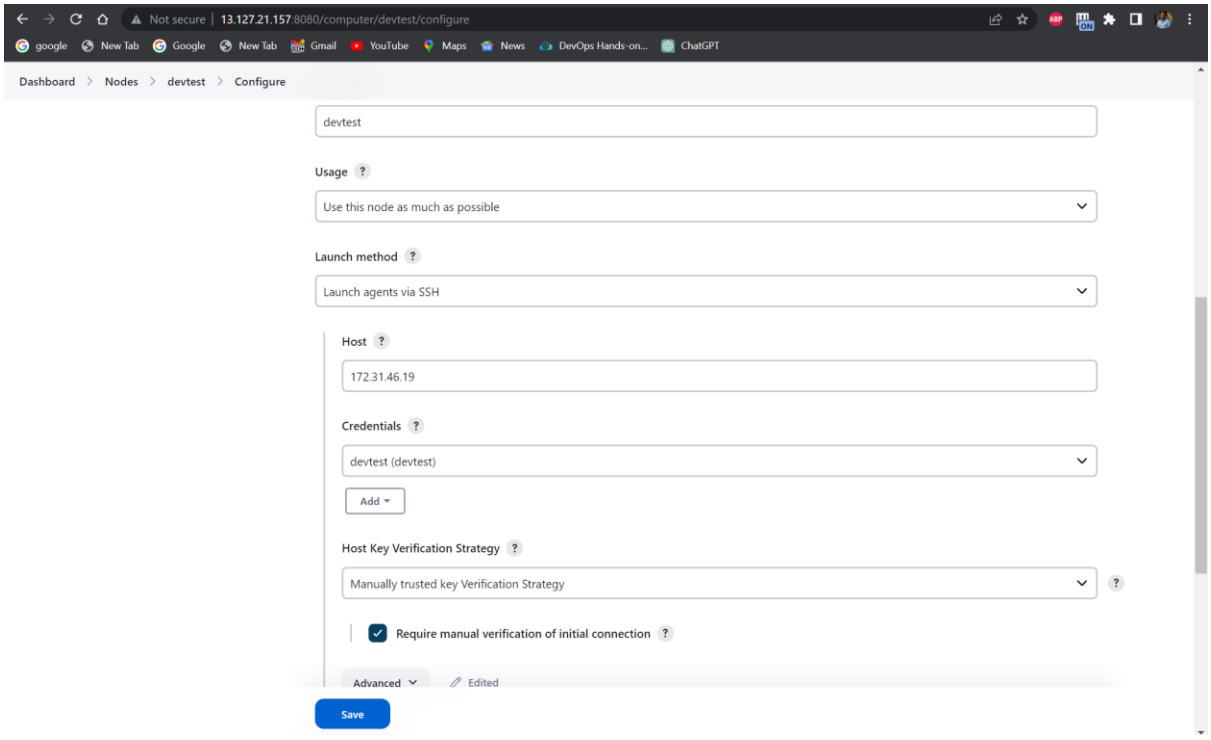
Checking the versions of the installed packages in Jenkins working Node

And creating the SSH key to configure Jenkins working Node to Jenkins Master

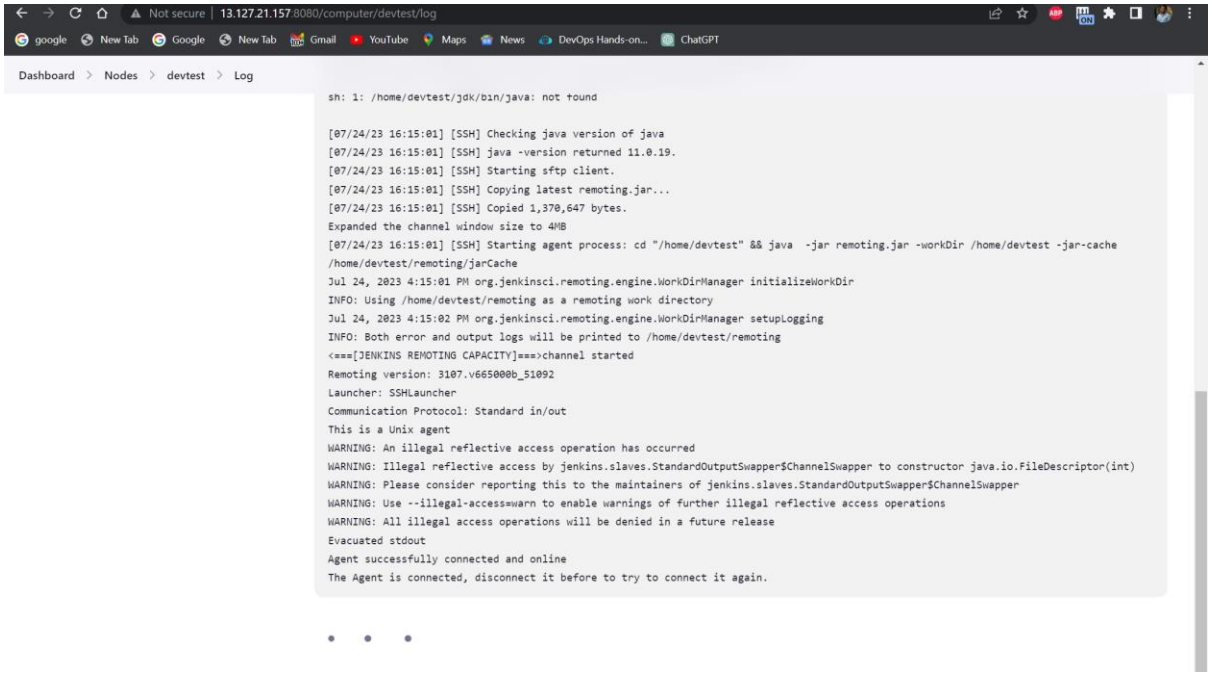


Configuring Jenkins working Node to Jenkins Master

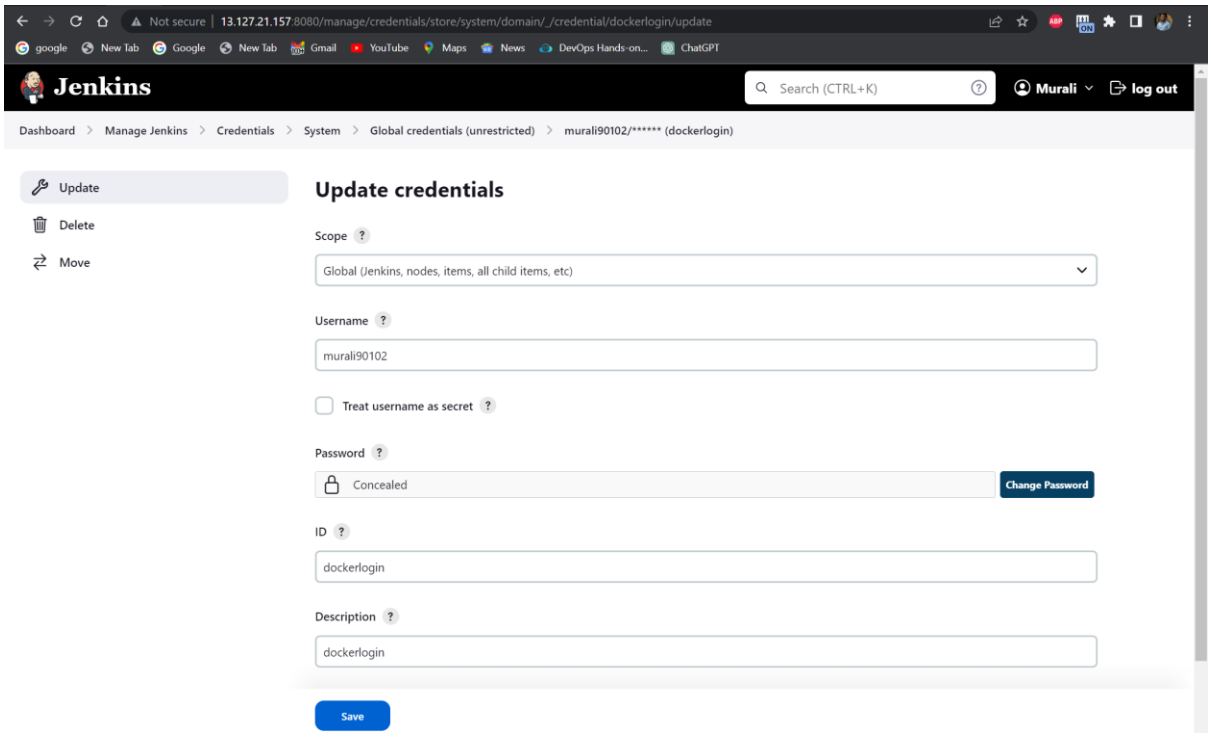


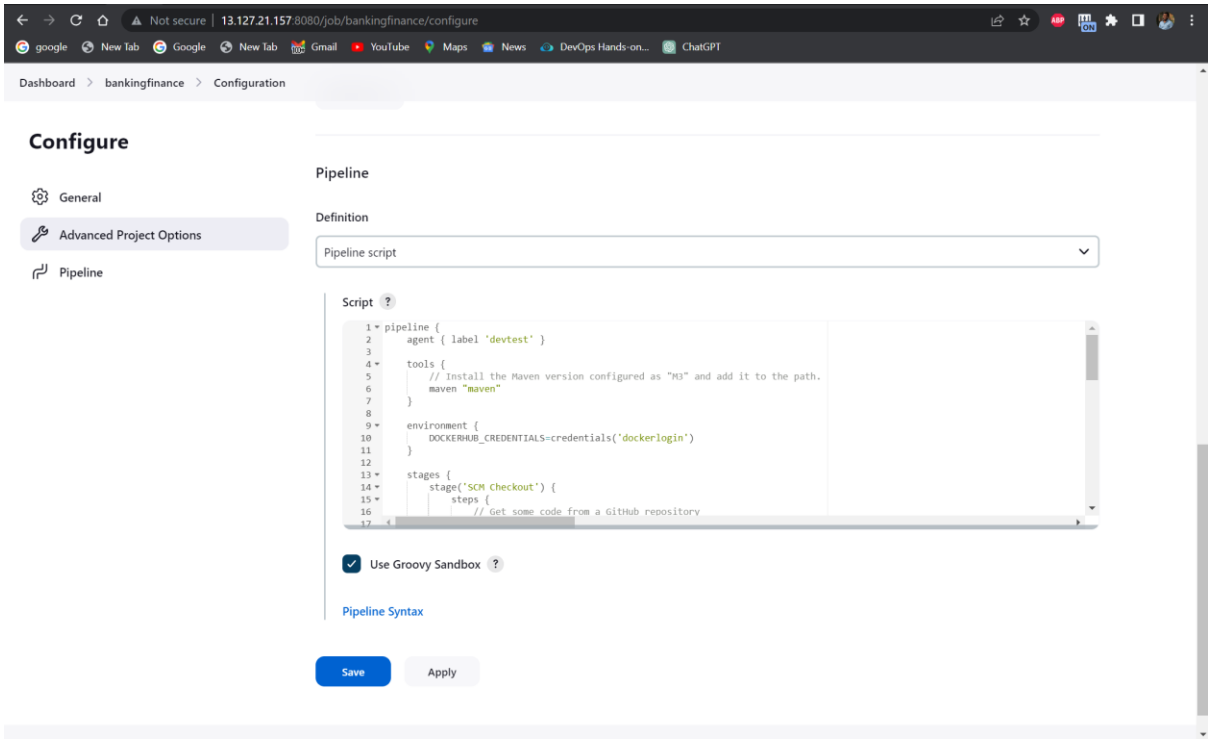


Checking the Agent Connection.

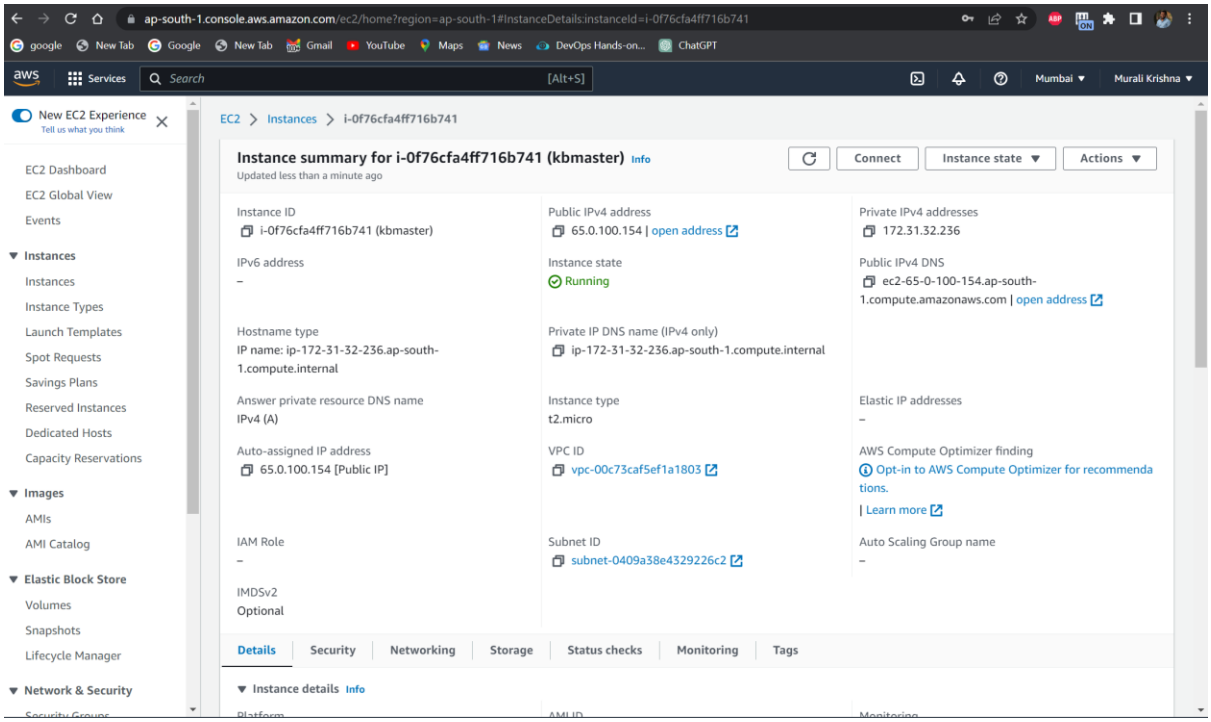


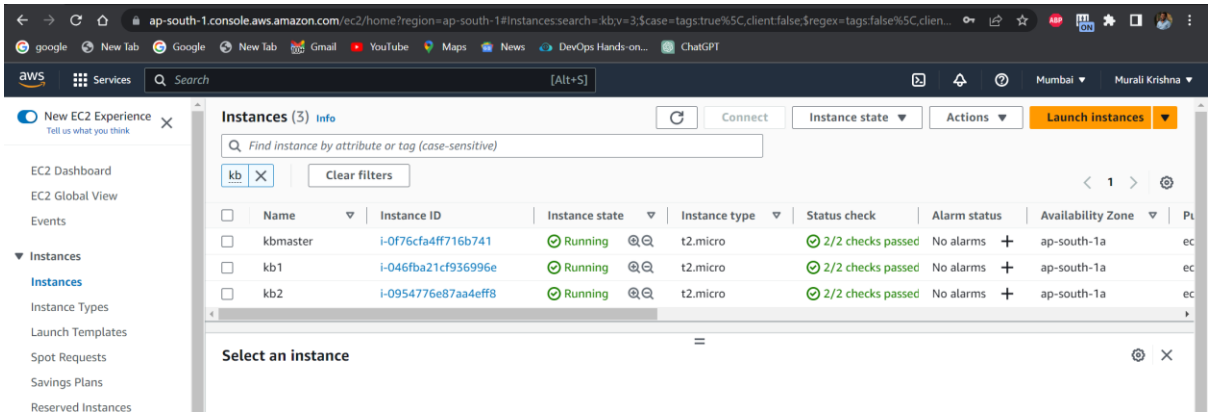
Adding the DockerHub login credentials in Jenkins



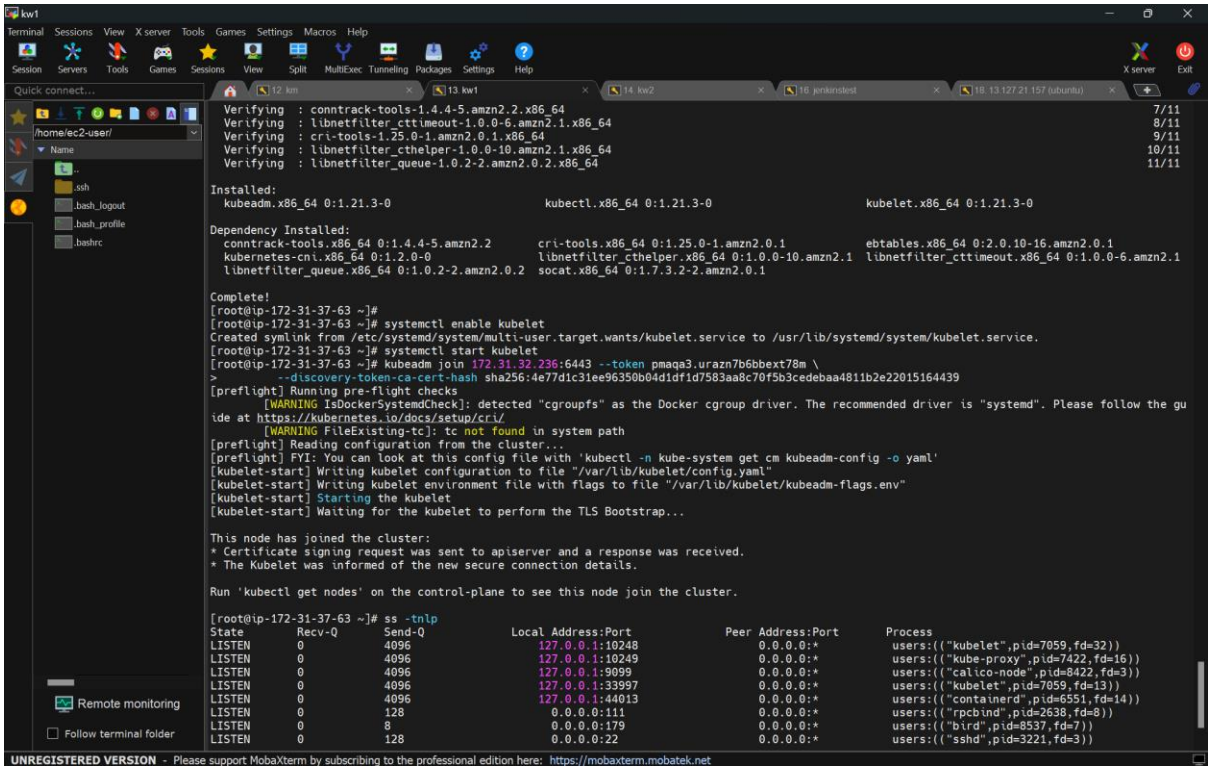


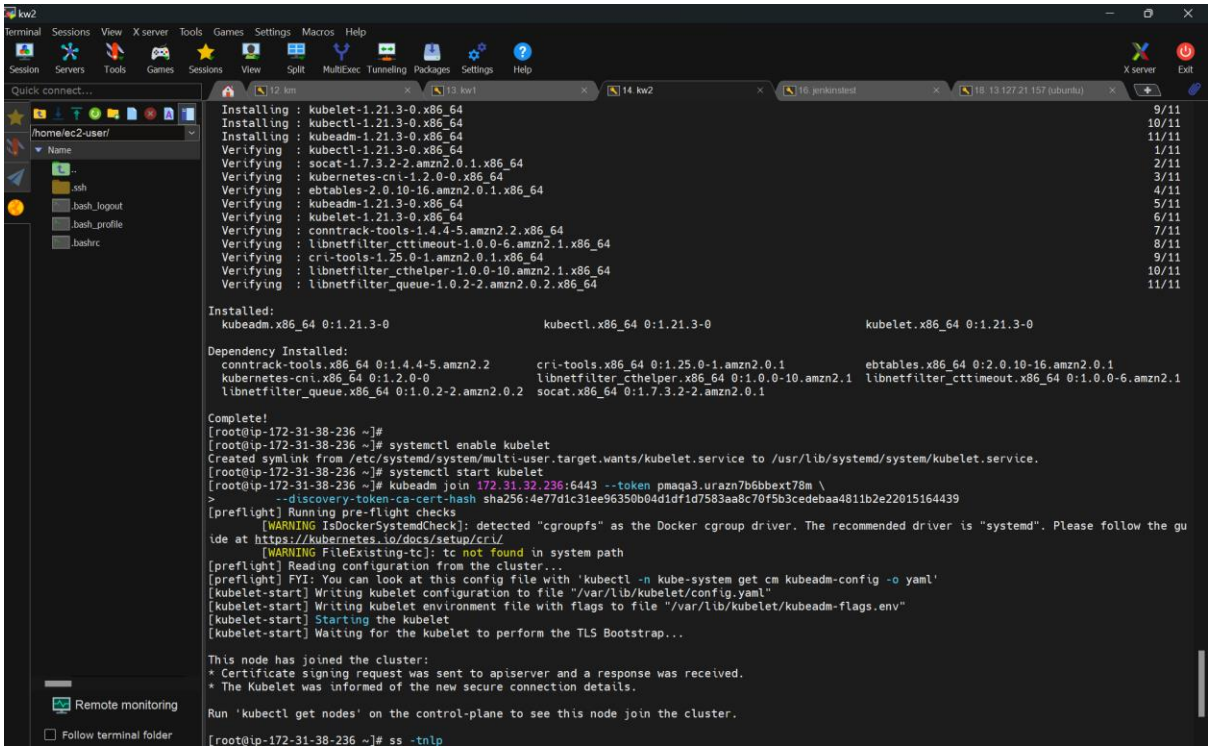
Creating instances for Kubernetes master(kbmaster) and working nodes 1(kb1) and 2(kb2)



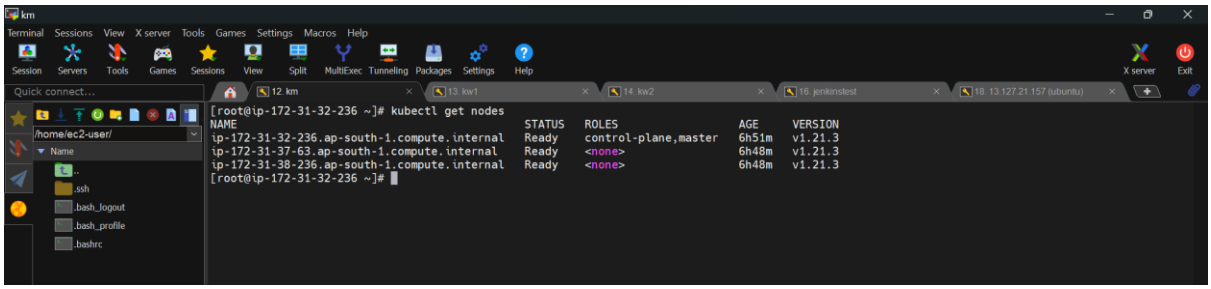


Installing Kubernetes in Kubernetes Master Instance and worker nodes instances.

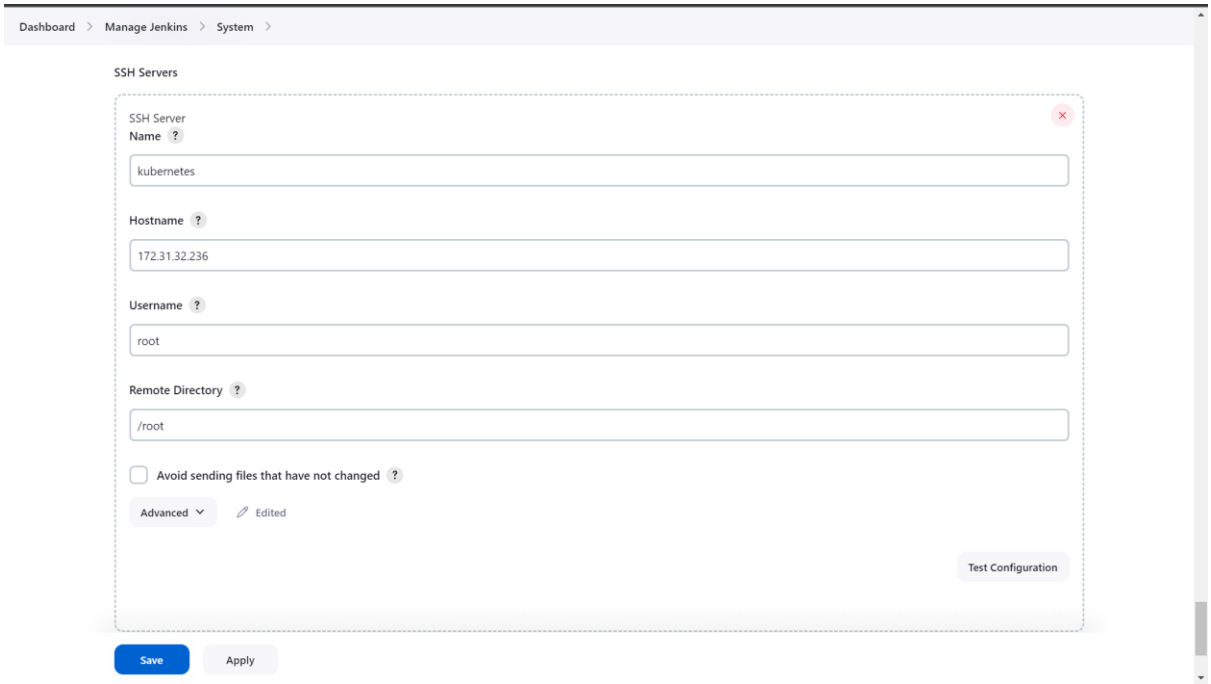


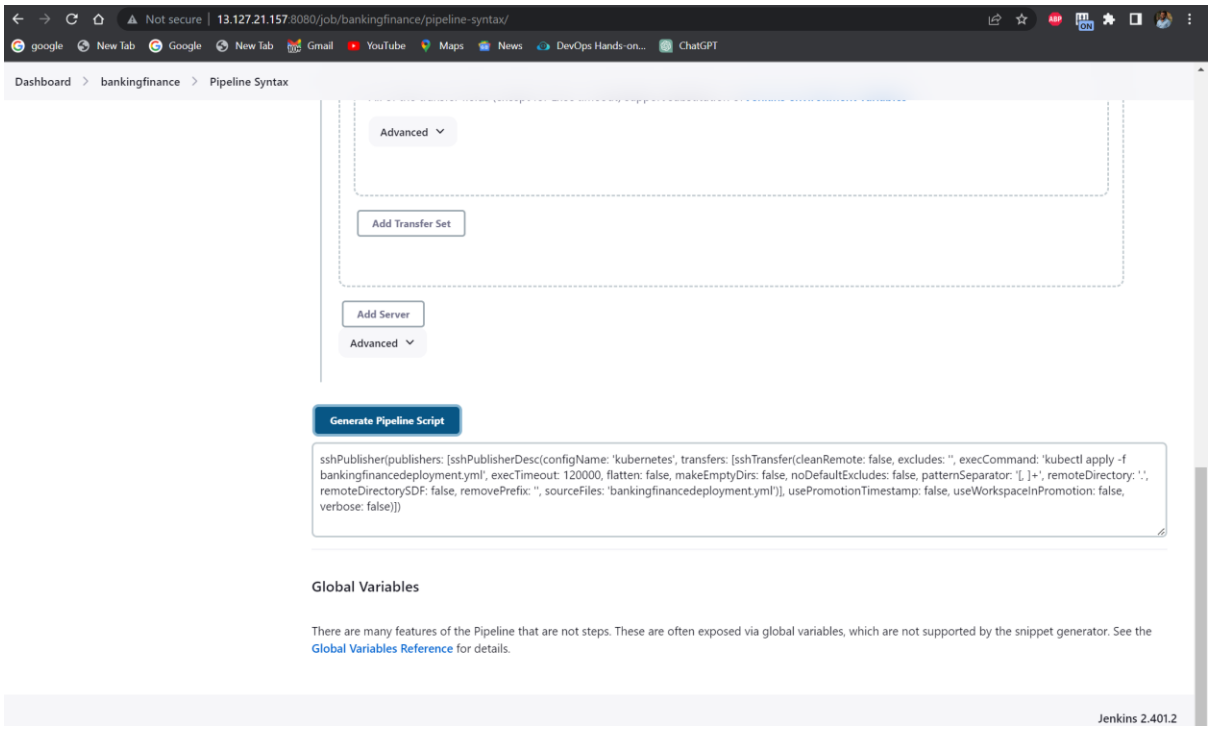


Checking the number of nodes available for Kubernetes Master

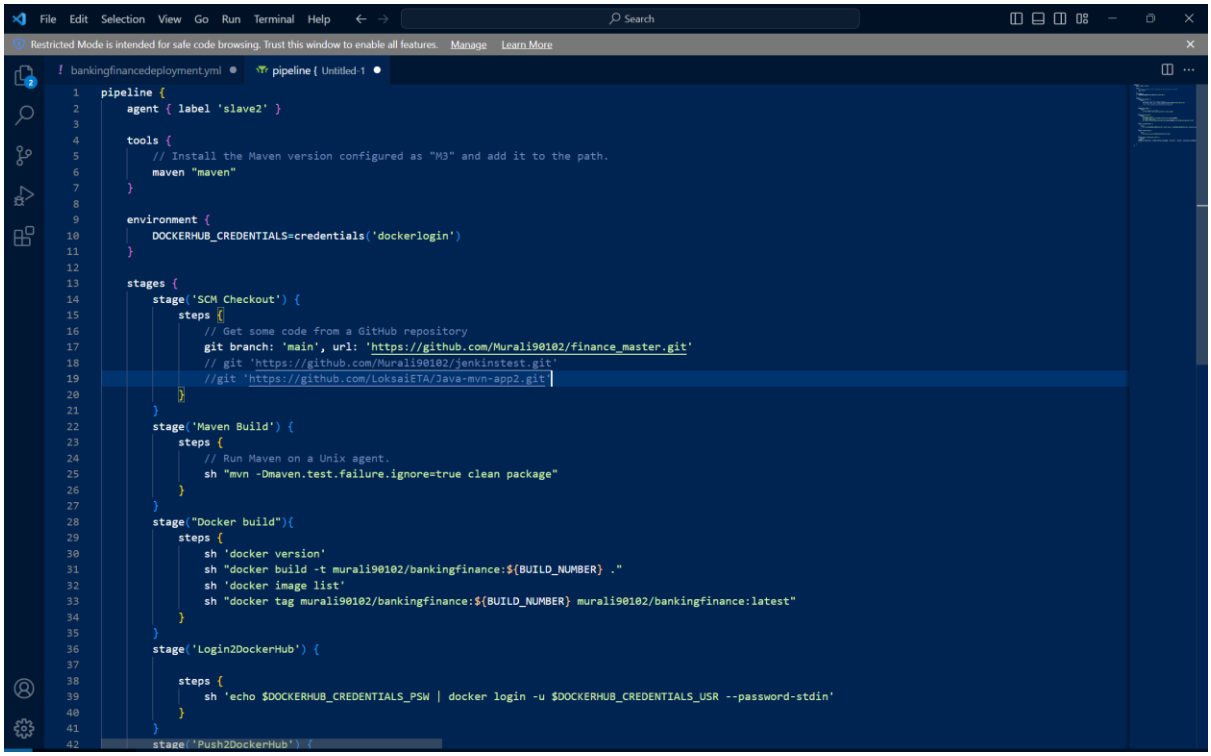


Configuring the Kubernetes Master server to Jenkins Master through SSH key and generating the pipeline script.



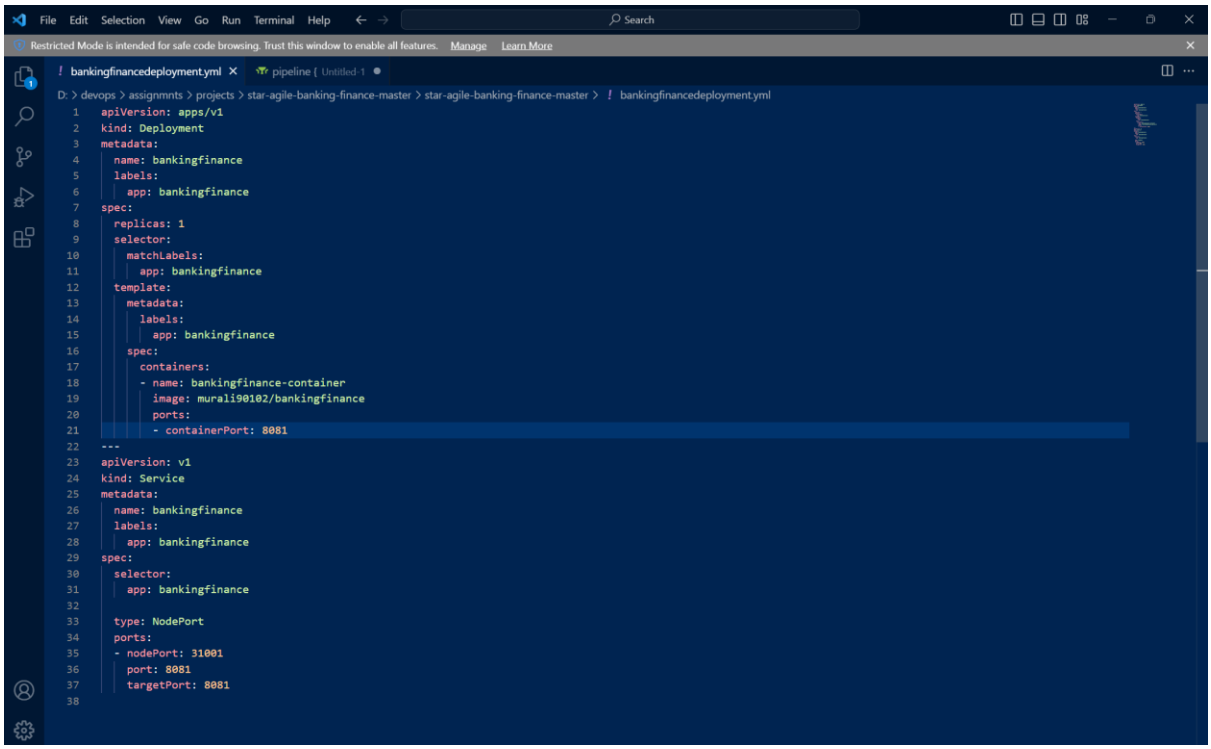


Coding the script in VS Code where we enter all the required commands for the source code to checkout, to build the project using Maven and to create a Docker image and push it into the Docker hub using the given credentials.



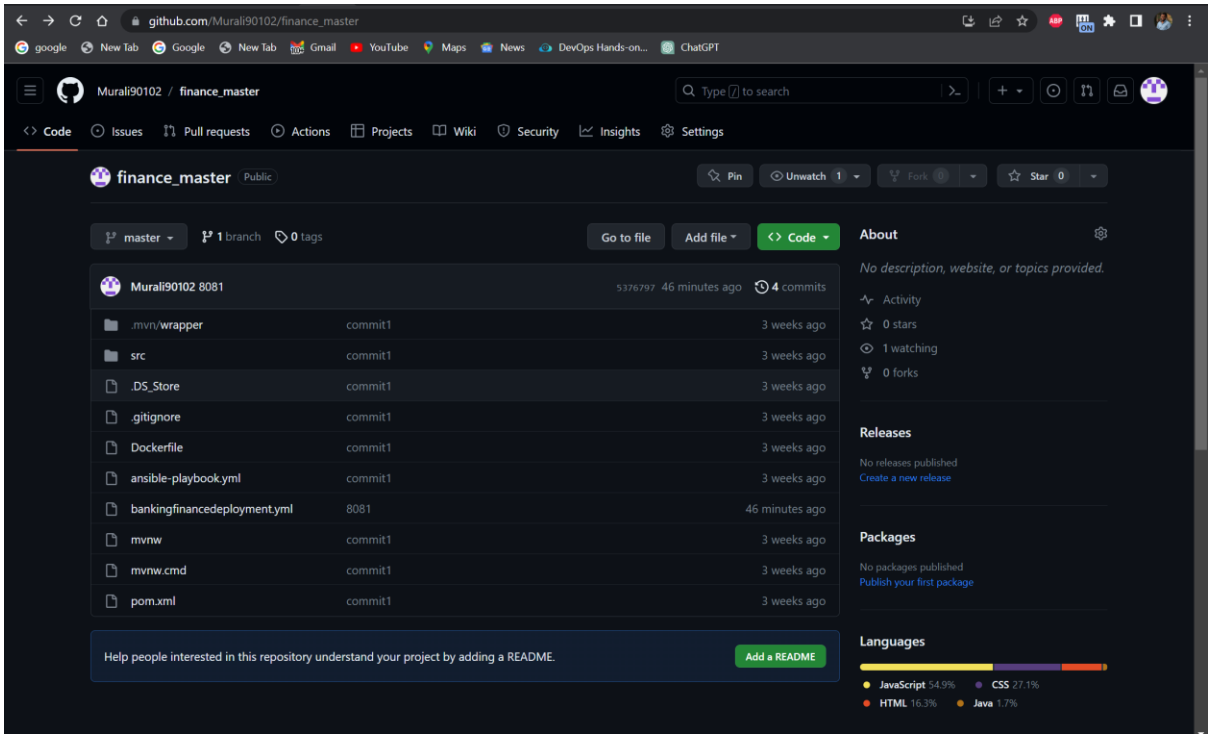
```
pipeline {
    agent { label 'slave2' }

    tools {
        // Install the Maven version configured as "M3" and add it to the path.
        maven "maven"
    }

    environment {
        DOCKERHUB_CREDENTIALS=credentials('dockerlogin')
    }

    stages {
        stage('SCM Checkout') {
            steps {
                // Get some code from a GitHub repository
                git branch: 'main', url: 'https://github.com/Murali90102/finance_master.git'
                // git 'https://github.com/Murali90102/jenkinstest.git'
                //git 'https://github.com/LoksaiETA/Java-mvn-app2.git'
            }
        }
        stage('Maven Build') {
            steps {
                // Run Maven on a Unix agent.
                sh "mvn -Dmaven.test.failure.ignore=true clean package"
            }
        }
        stage("Docker build"){
            steps {
                sh 'docker version'
                sh "docker build -t murali90102/bankingfinance:${BUILD_NUMBER} ."
                sh 'docker image list'
                sh "docker tag murali90102/bankingfinance:${BUILD_NUMBER} murali90102/bankingfinance:latest"
            }
        }
        stage('Login2DockerHub') {

            steps {
                sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
            }
        }
        stage('Push2DockerHub') {
```

Creation of Manifest file for deployment of the application using Kubernetes

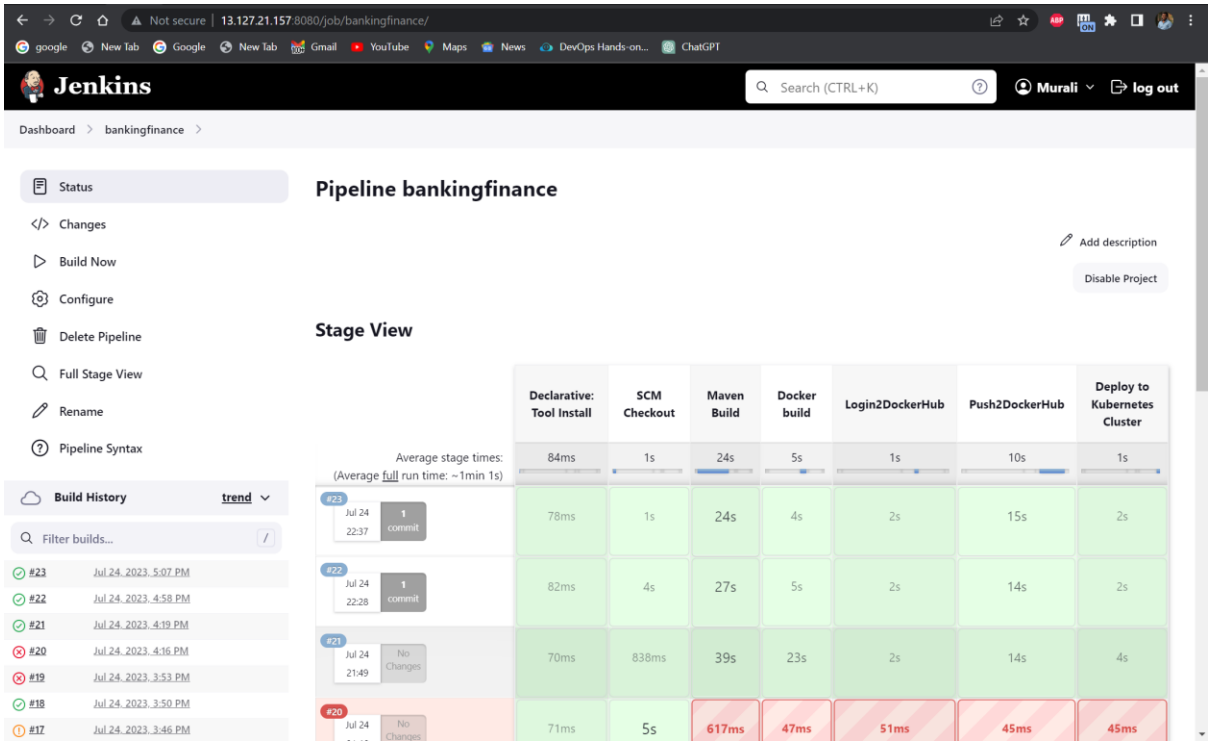


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bankingfinance
  labels:
    app: bankingfinance
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bankingfinance
  template:
    metadata:
      labels:
        app: bankingfinance
    spec:
      containers:
      - name: bankingfinance-container
        image: murali90102/bankingfinance
        ports:
        - containerPort: 8081
---
apiVersion: v1
kind: Service
metadata:
  name: bankingfinance
  labels:
    app: bankingfinance
spec:
  selector:
    app: bankingfinance

  type: NodePort
  ports:
  - nodePort: 31001
    port: 8081
    targetPort: 8081
```

Pushing all the files into GitHub.



Started building the CI/CD pipeline, executed and deployed successfully after some hiccups. Check the logs for verification of all activities.



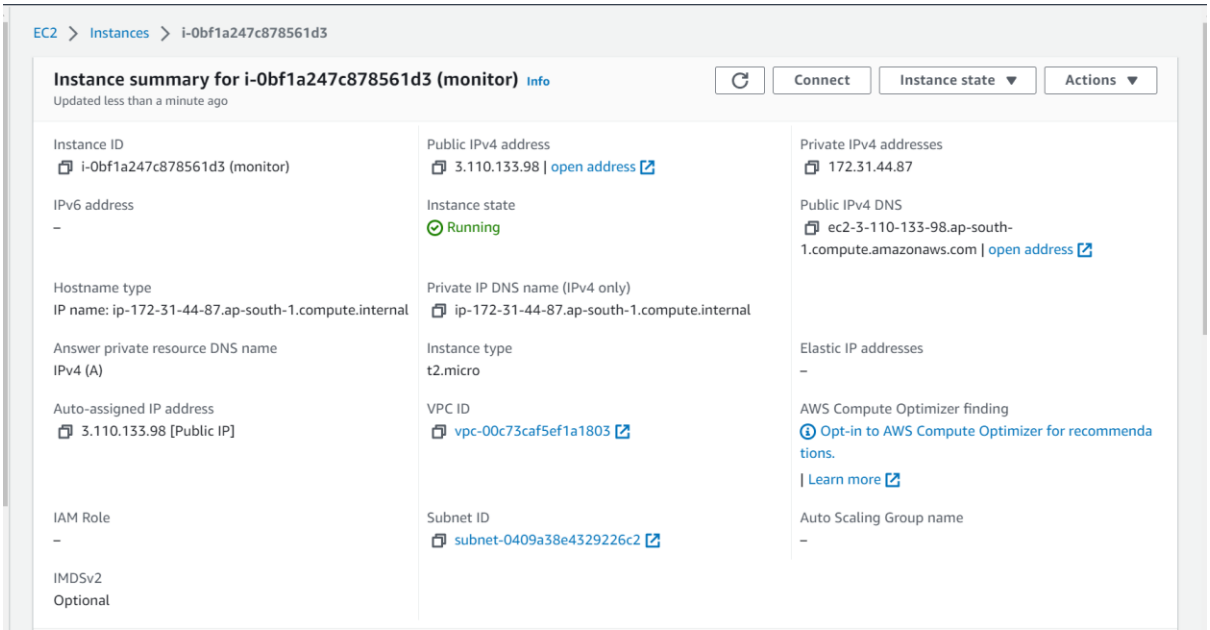
Dashboard > bankingfinance > #23

```
[ [1;34mINFO [m]
[ [1;34mINFO [m] [1m--- [0;32mmaven-surefire-plugin:2.22.2:test [m [1m(default-test) [m @ [36mbanking [0;1m ---[m
[ [1;34mINFO [m]
[ [1;34mINFO [m] -------------------------------------------------------
[ [1;34mINFO [m]  T E S T S
[ [1;34mINFO [m] -------------------------------------------------------
[ [1;34mINFO [m] Running com.project.staragile.banking. [1mBankingApplicationTests [m
17:07:28.781 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class
[org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate]
17:07:28.805 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public
org.springframework.test.context.support.DefaultBootstrapContext(java.lang.Class,org.springframework.test.context.CacheAwareContextLoad
erDelegate)]
17:07:28.875 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class
[com.project.staragile.banking.BankingApplicationTests] from class
[org.springframework.boot.test.context.SpringBootTestContextBootstrapper]
17:07:28.903 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Neither @ContextConfiguration nor
@ContextHierarchy found for test class [com.project.staragile.banking.BankingApplicationTests], using SpringBootContextLoader
17:07:28.914 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for
test class [com.project.staragile.banking.BankingApplicationTests]: class path resource
[com/project/staragile/banking/BankingApplicationTests-context.xml] does not exist
17:07:28.917 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not detect default resource location for
test class [com.project.staragile.banking.BankingApplicationTests]: class path resource
[com/project/staragile/banking/BankingApplicationTestsContext.groovy] does not exist
17:07:28.920 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not detect default resource locations
for test class [com.project.staragile.banking.BankingApplicationTests]: no resource found for suffixes {-context.xml, Context.groovy}.
17:07:28.923 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils - Could not detect default
configuration classes for test class [com.project.staragile.banking.BankingApplicationTests]: BankingApplicationTests does not declare
any static, non-private, non-final, nested classes annotated with @Configuration.
17:07:29.035 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring class'
for annotation type [org.springframework.test.context.ActiveProfiles] and class [com.project.staragile.banking.BankingApplicationTests]
17:07:29.206 [main] DEBUG org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider - Identified candidate
component class: file [/home/devtest/workspace/bankingfinance/target/classes/com/project/staragile/banking/BankingApplication.class]
17:07:29.208 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - Found @SpringBootConfiguration
com.project.staragile.banking.BankingApplication for test class com.project.staragile.banking.BankingApplicationTests
17:07:29.433 [main] DEBUG org.springframework.boot.test.context.SpringBootTestContextBootstrapper - @TestExecutionListeners is not
```

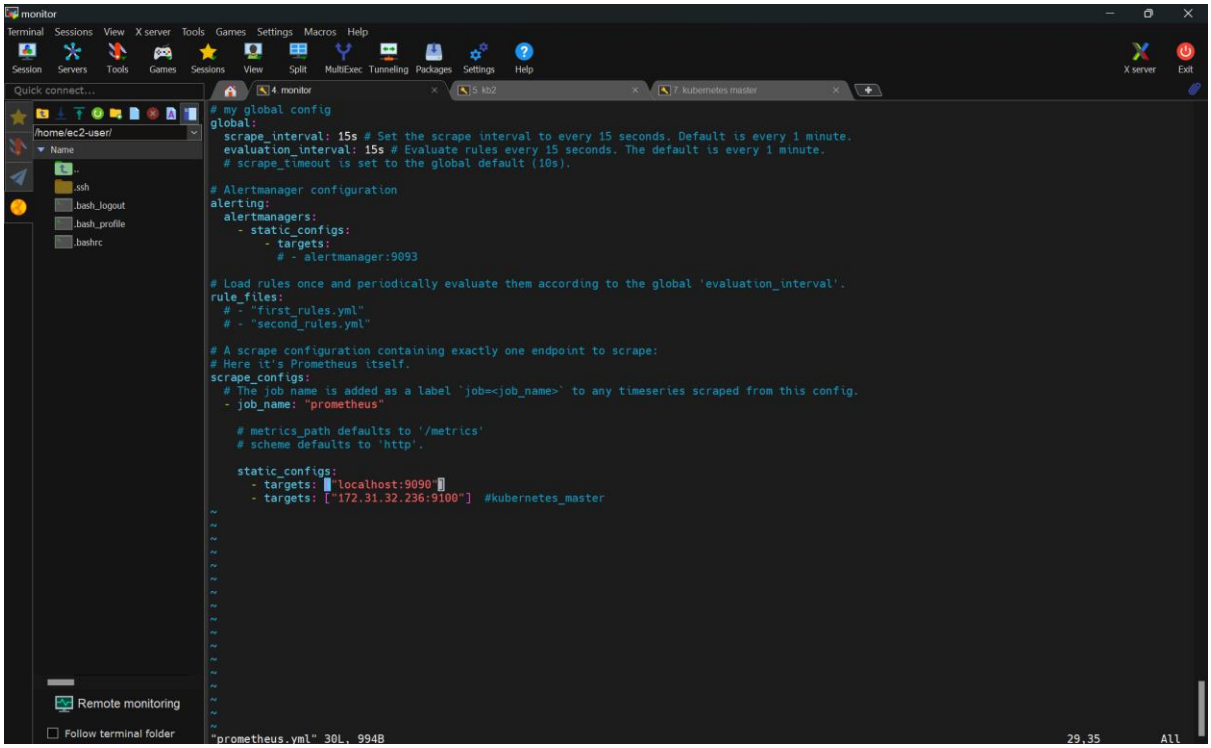
Dashboard > bankingfinance > #23

```
SSH: SFTP channel open
SSH: Connecting SFTP channel ...
SSH: Connected
SSH: cd [/root]
SSH: OK
SSH: cd [/root]
SSH: OK
SSH: put [ansible-playbook.yml]
SSH: put [bankingfinancedeployment.yml]
SSH: Opening exec channel ...
SSH: EXEC: channel open
SSH: EXEC: STDOUT/STDERR from command [kubectl apply -f bankingfinancedeployment.yml] ...
SSH: EXEC: connected
deployment.apps/bankingfinance configured
service/bankingfinance configured
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

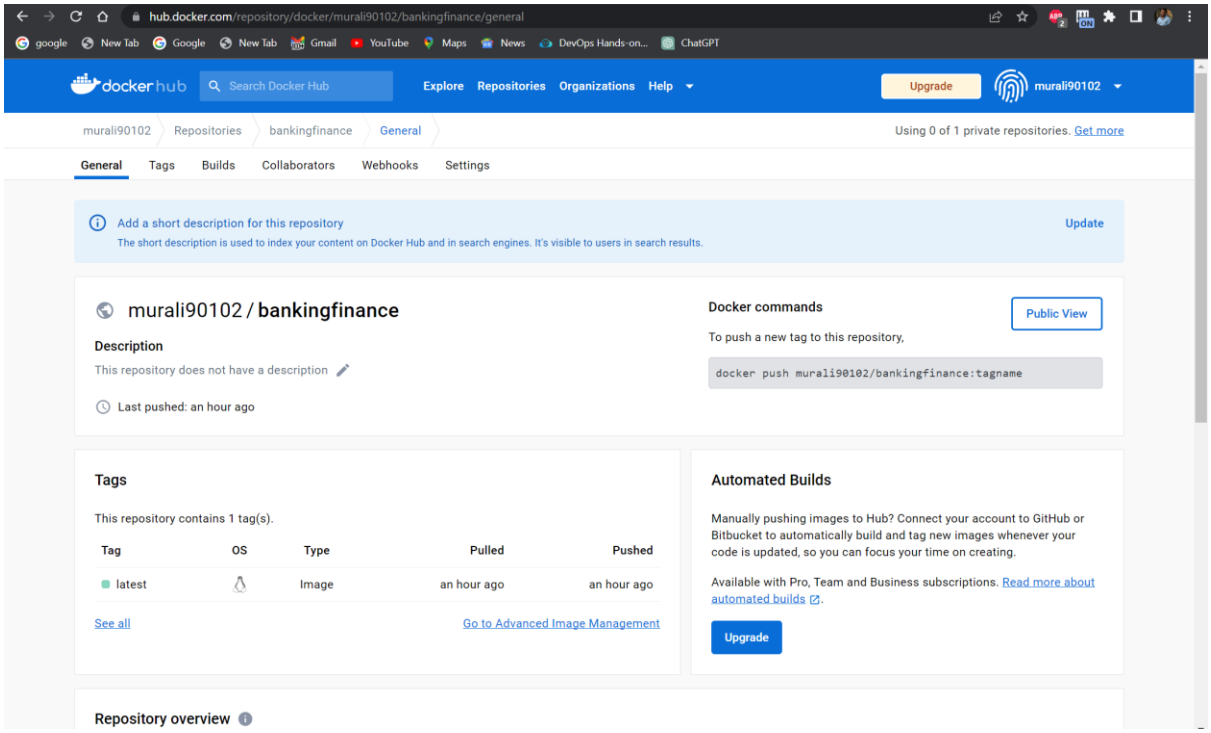
Launching the Monitoring server to monitor the deployment.



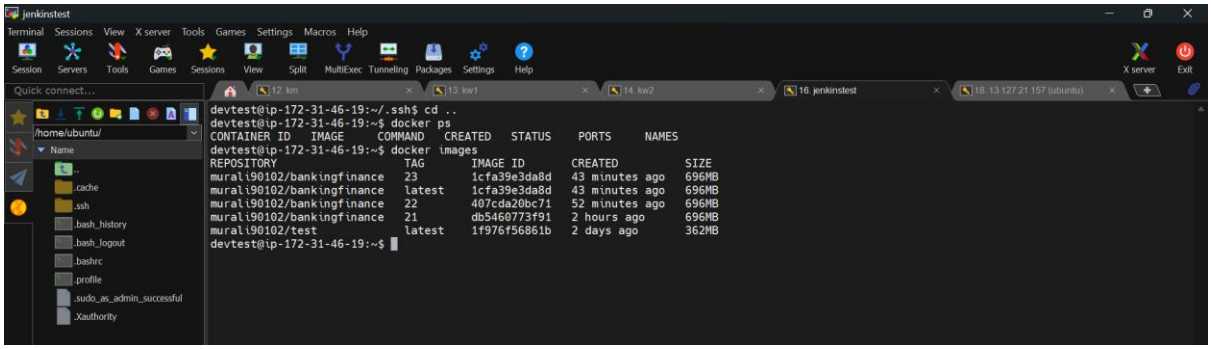
Installing Prometheus and configuring the Kubernetes server with Prometheus server, which we want to Monitor.



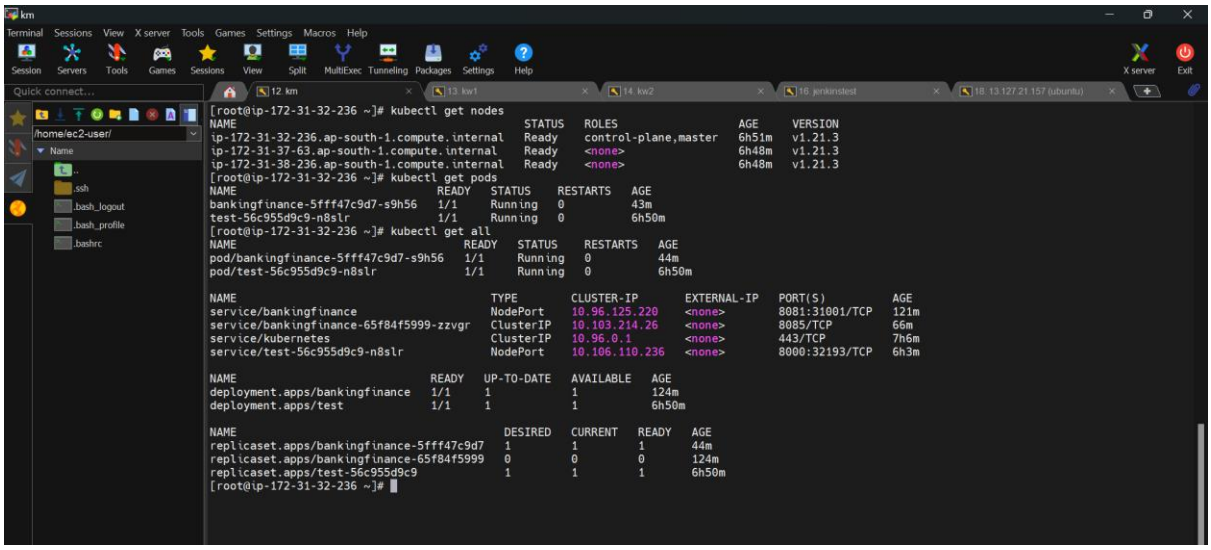
Verifying Dockerhub for the successful creation of Docker image and publishment in Dockerhub.



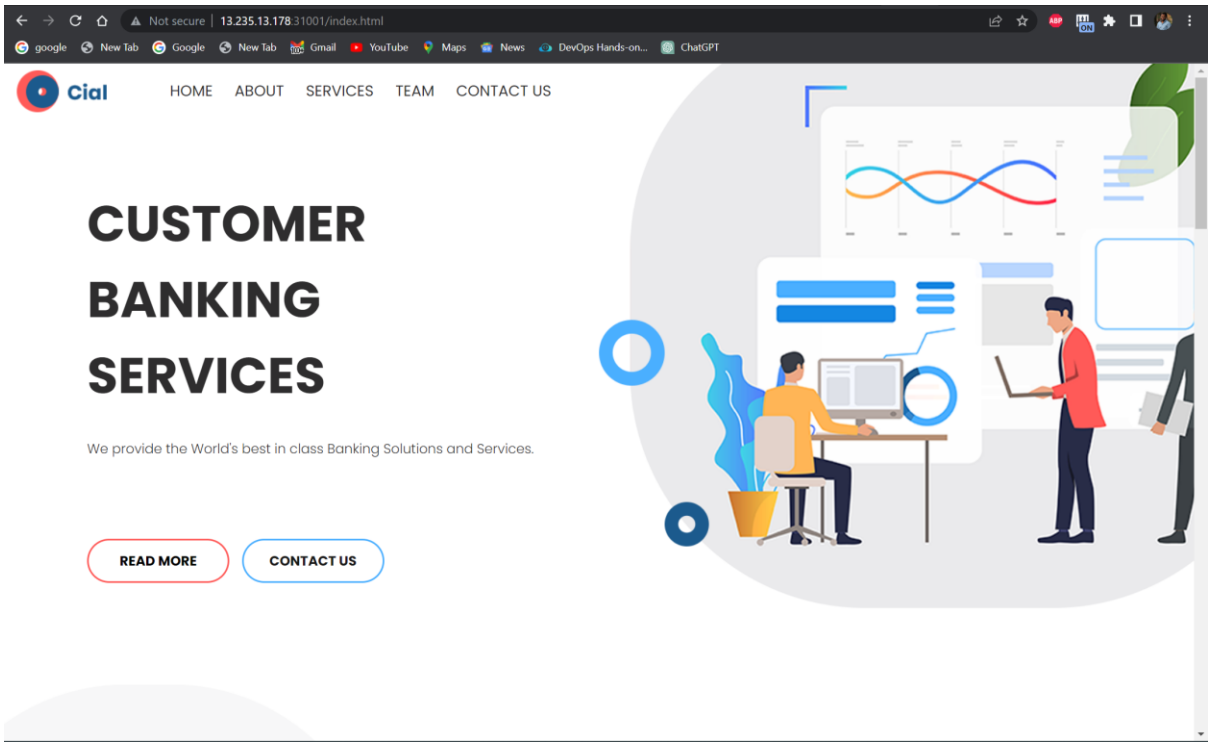
Verifying the creation of Docker image in server.



Verifying the deployment in Kubernetes server



Accessing the application with the Node IP address and port number.



Prometheus and Grafana monitoring graphs of Kubernetes worker Node where the application is Deployed.