Maven with Jenkins Job with Java Application

Introduction:

This Documentation provides step by step guide to integrating Maven with Jenkins to streamline the build process of Java applications. Whether you're a seasoned developer or a newcomer to CI/CD practices, this guide will walk you through the setup and configuration steps, empowering you to efficiently manage your Java projects within a continuous integration environment

Key Objectives:

- ✓ Understanding Maven.
- ✓ Introduction to Jenkins.
- ✓ Setting up Jenkins.
- ✓ Configuring Maven Integration.
- ✓ Creating Jenkins Jobs.
- ✓ Building Java Projects with Maven.
- ✓ Triggering Builds.
- ✓ Managing Dependencies.
- ✓ Running Tests.
- ✓ Deploying Artifacts.

Prerequisites:

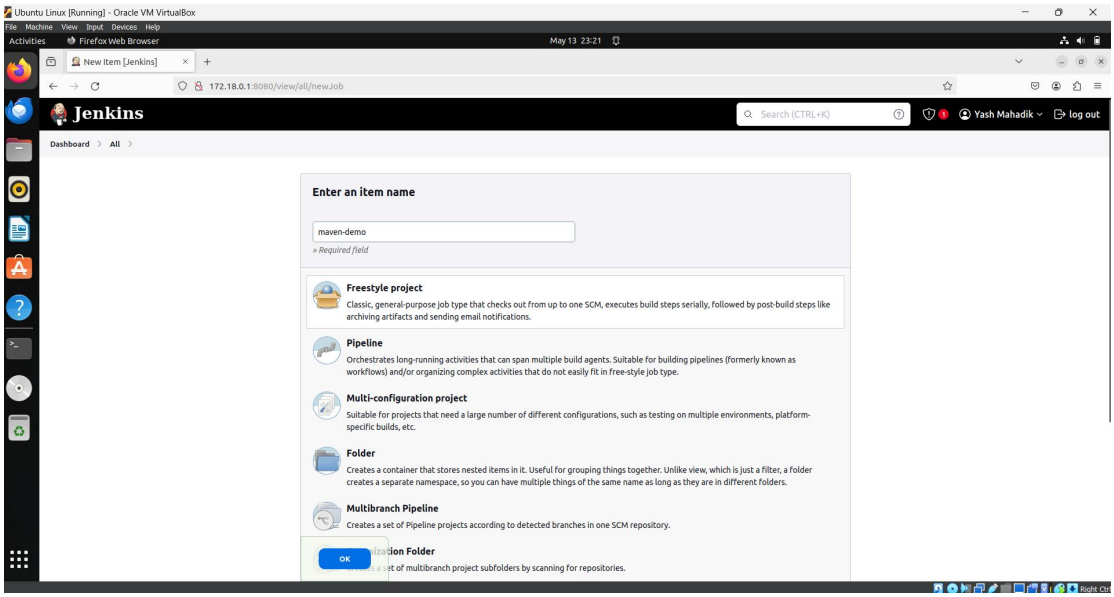
- ✓ Basic understanding of Java development and Maven concepts.
- ✓ Familiarity with CI/CD principles is beneficial but not mandatory.
- ✓ Access to a system where Jenkins can be installed and configured.

What is Maven ?

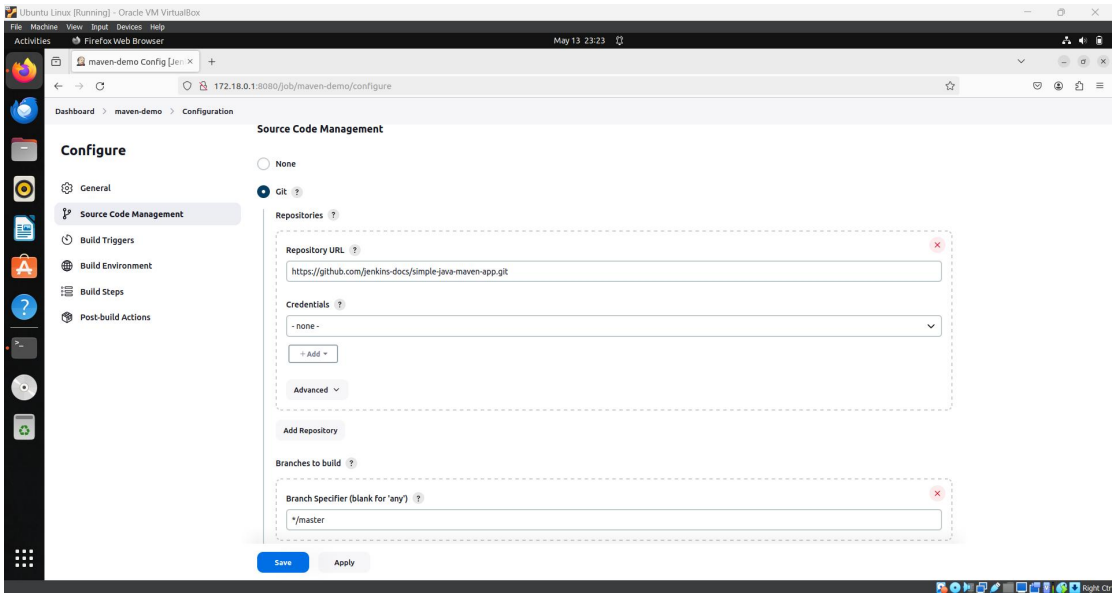
A maven is a build tool designed to manage dependencies and the software lifecycle. It is also designed to work with plugins that allow users to add other tasks to the standard compile, test, package, install, deploy tasks. While, Jenkins is designed for the purpose of implementing Continuous Integration (CI).

A step-by-step guide to Set up a MAven with Jenkins Job

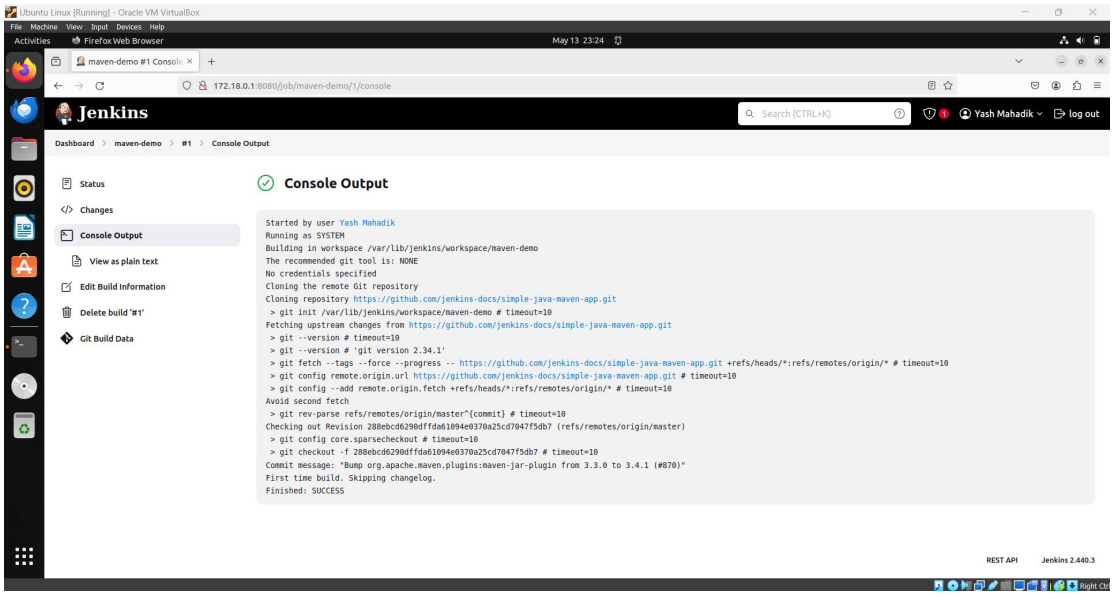
1) Setting up Jenkins , and login to your Jenkins account. Create new Job Name it as maven-demo and click on OK.



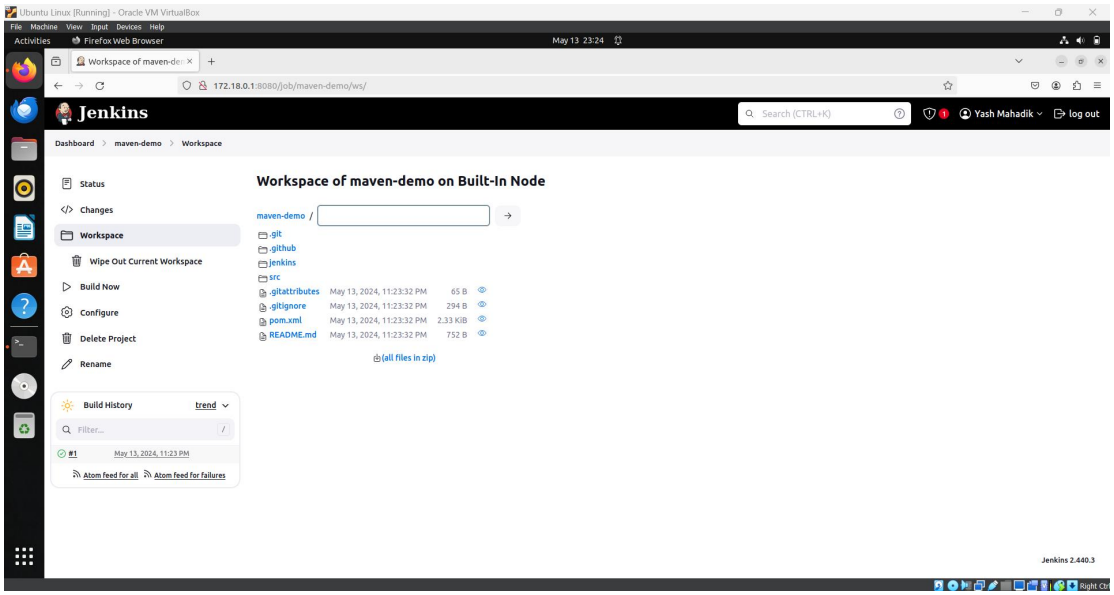
2) First we need Sample maven project refer to [maven project](#) from git-hub. In Source code management section click on git and paste the git URL and click on save.



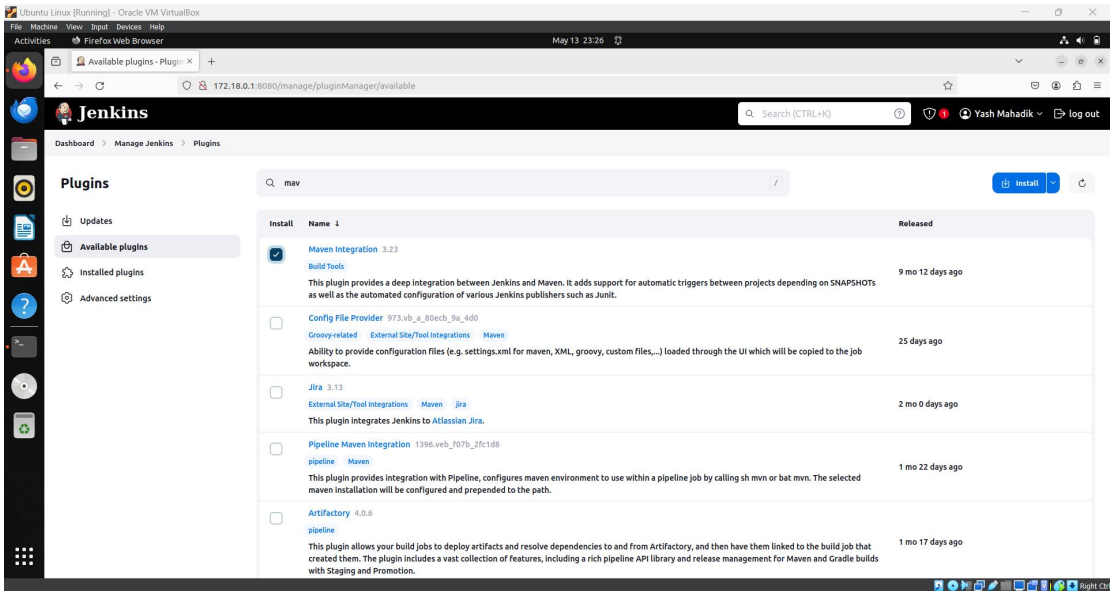
3) After Building the job , you can see the whole repository of git has been clone in your jerkins and its successfully build



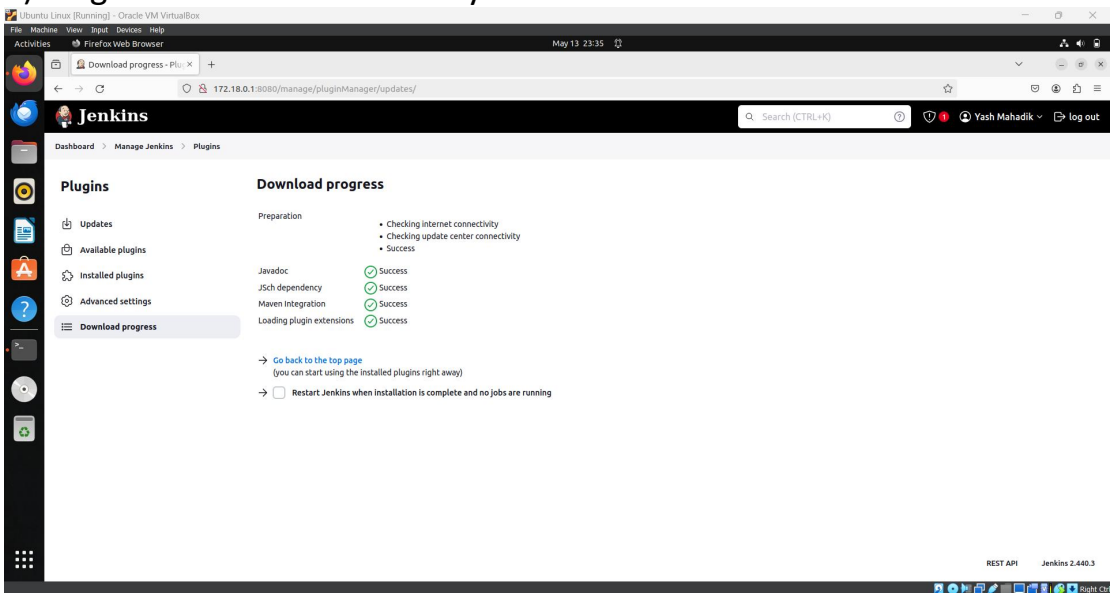
4) You can do check all the file from git hub to Jenkins workspace.



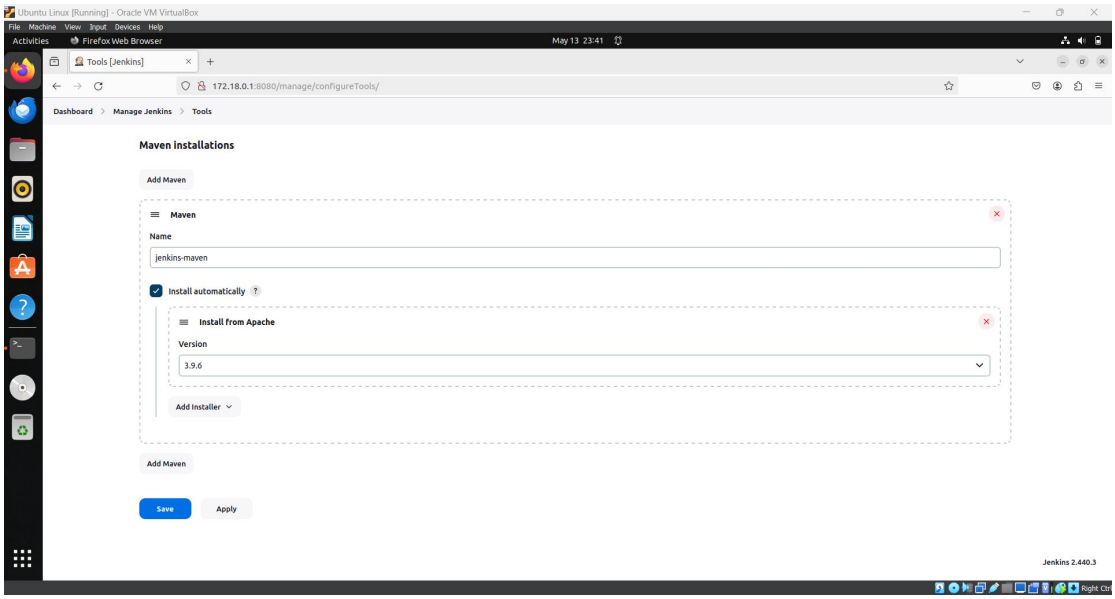
5) As it is a Maven project we need to Configuring Maven Integration by installing Maven plugins.



6) Plugins has been successfully Downloaded.



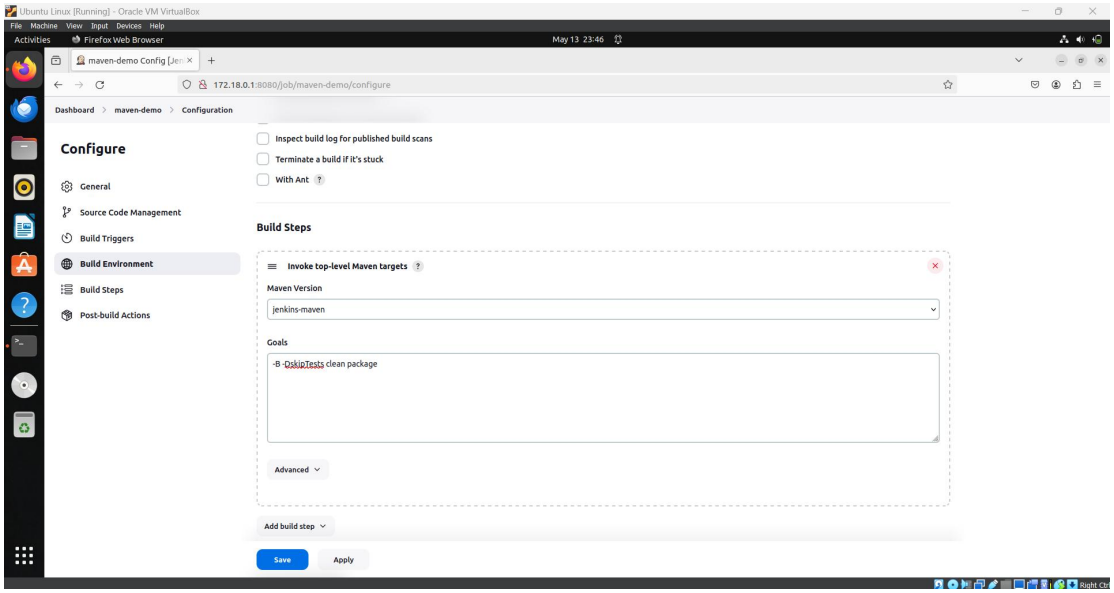
7) Go to Manage Jenkins , go to tools and if you scroll down you will get Maven installing section. Add maven name as jenkins-maven with suitable version. And click on Save.

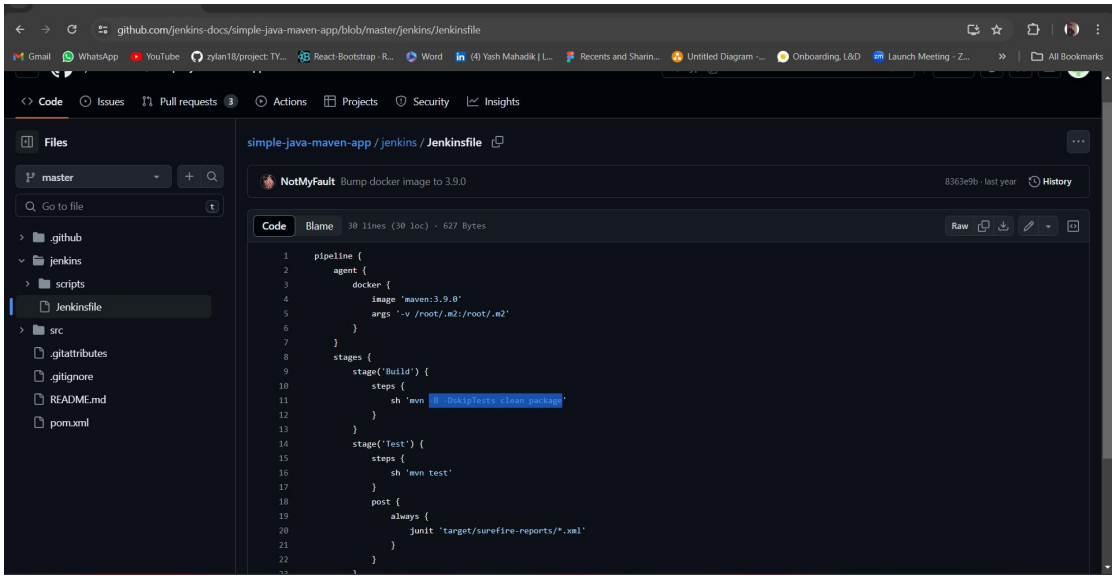


8) In Your Maven-demo go to configure and scroll down to search Build Steps. You will get **Invoke top-Level Maven targets** and add maven version and in Goal section add the firststep from git's Jenkinsfile. Copy -B -DskipTests clean package and add in Goal section. And click on save.

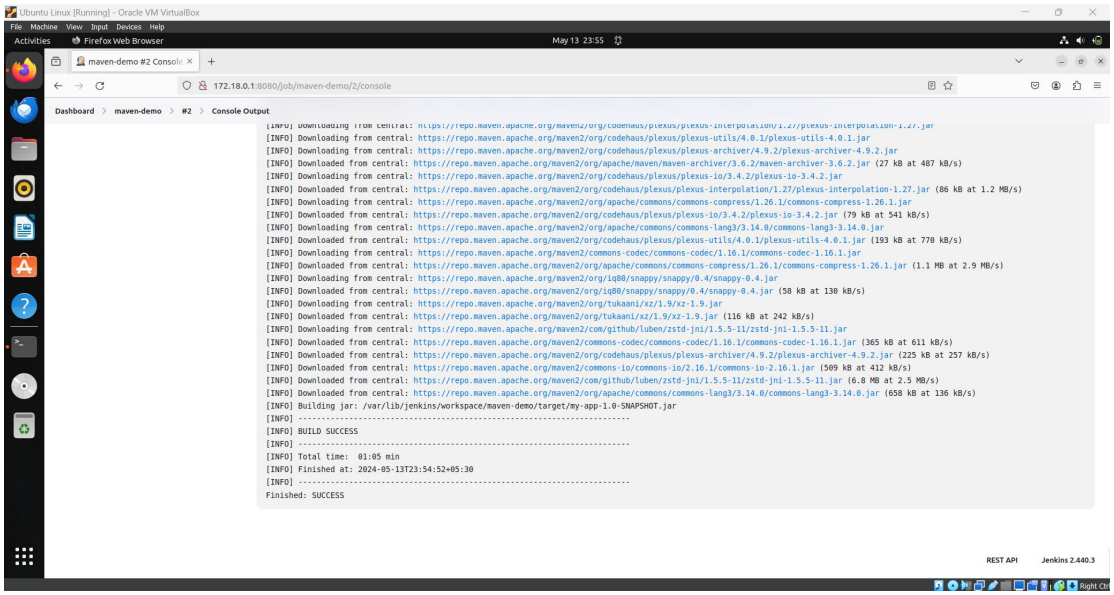
```
pipeline {
  agent {
    docker {
      image 'maven:3.9.0'
      args '-v /root/.m2:/root/.m2'
    }
  }
  stages {
    stage('Build') {
      steps {
        sh 'mvn -B -DskipTests clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
      }
      post {
        always {
```

```
            junit 'target/surefire-reports/*.xml'
        }
      }
    }
    stage('Deliver') {
      steps {
        sh './jenkins/scripts/deliver.sh'
      }
    }
  }
}
```



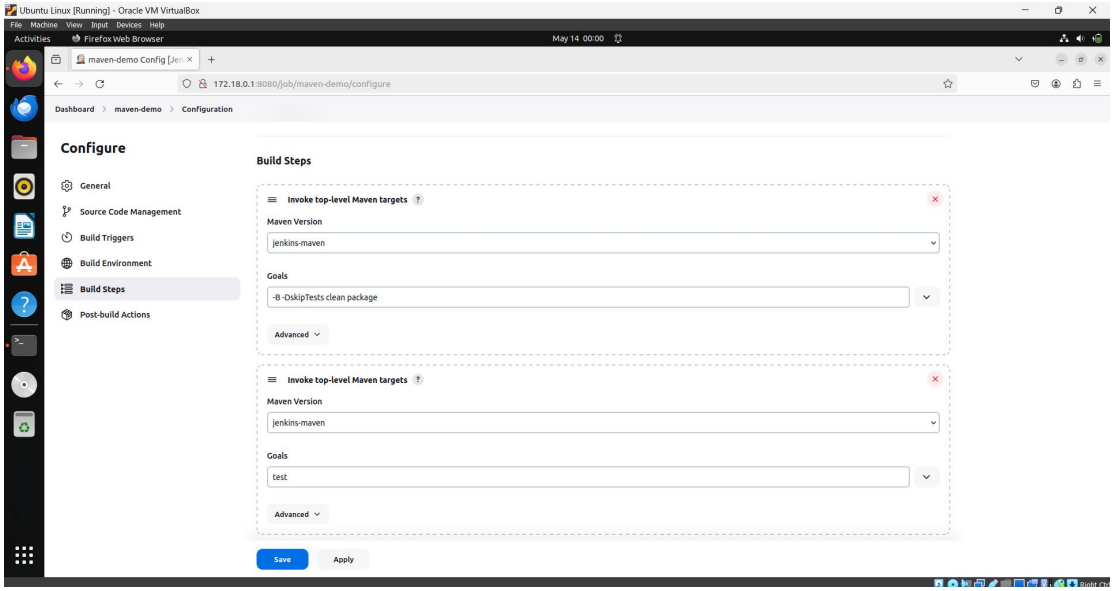


9) Click on Build Now and you will see the build has been done successfully.

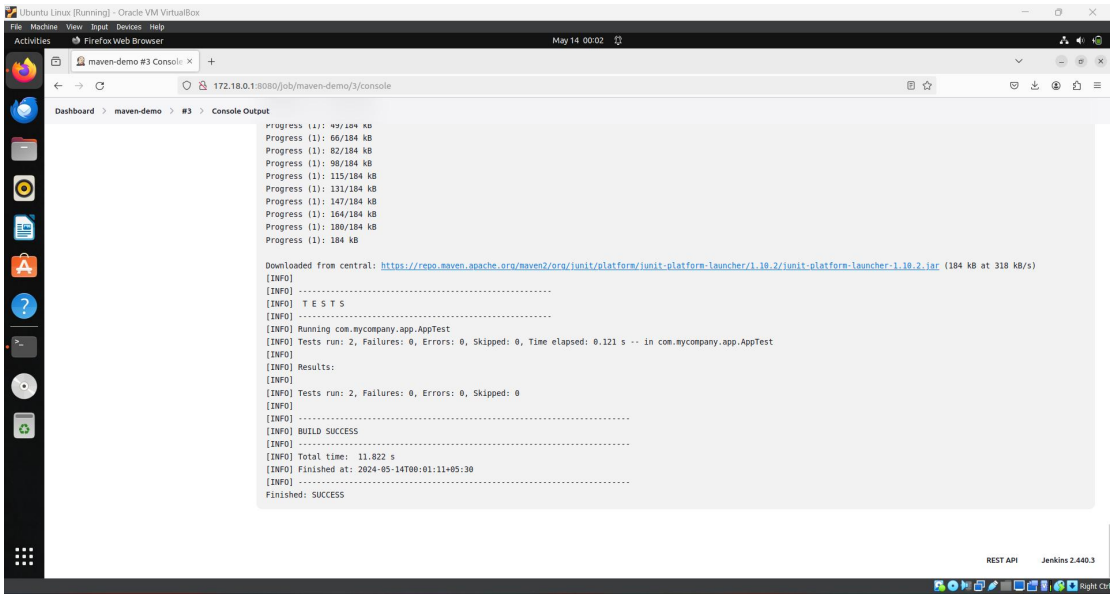


10) Now to add 2nd Step in Build step section again with **Invoke top-Level Maven targets** with maven version and goal as test. (steps of Jenkinsfile). Click on Save.

```
}
  stage('Test') {
    steps {
      sh 'mvn test'
    }
```



11) After Building the job, in console output you can see test and build has been done successfully.

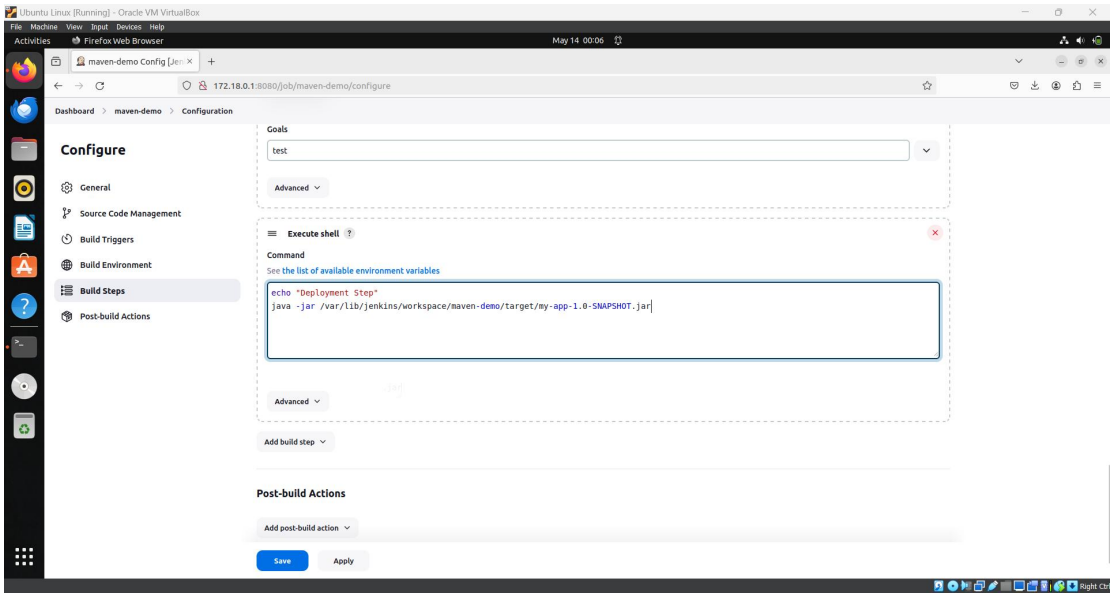


12) Now to execute deployment step. Go to configure in Build step choose Execute shell and add command

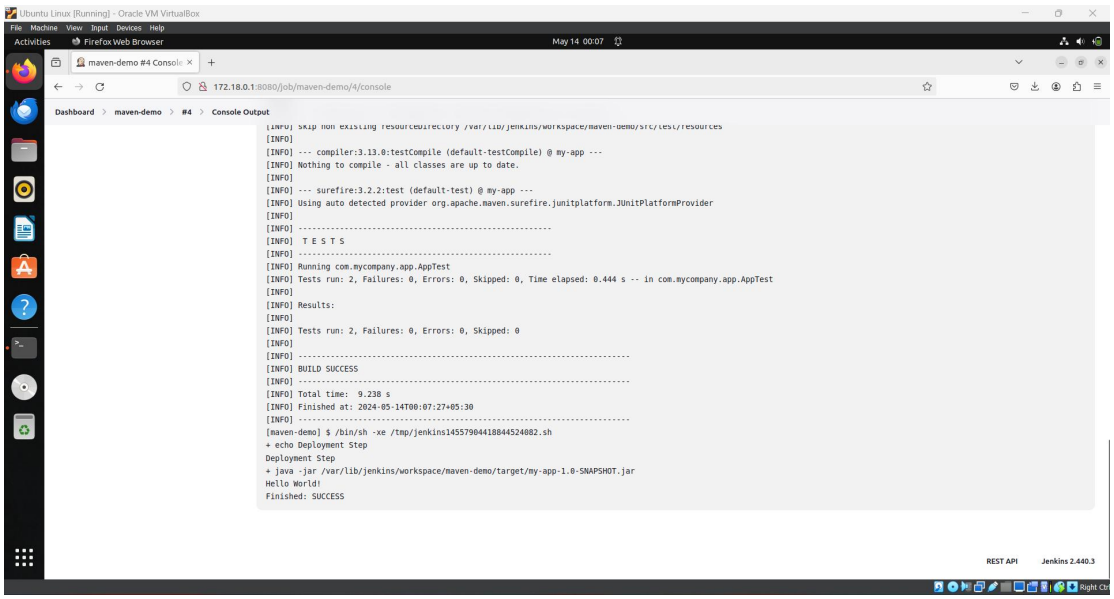
echo "Deployment Step"

java -jar /var/lib/jenkins/workspace/maven-demo/target/my-app-1.0-SNAPSHOT.jar

(You will find this .jar address in your local host or in the console output)

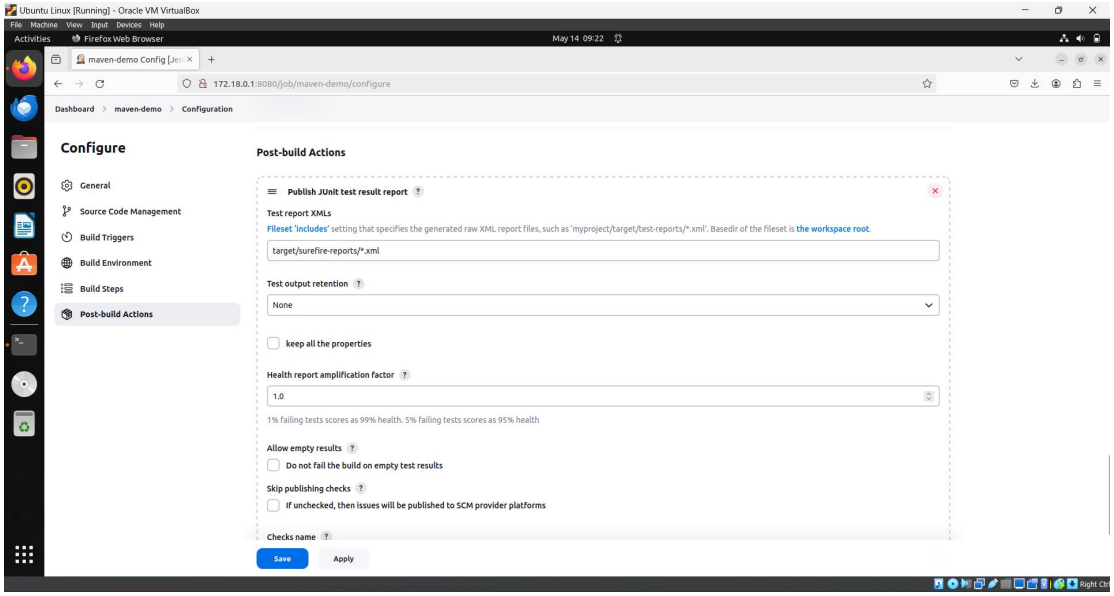


13) Again after Building , in console output you will see that test build and development step has been successfully done with an output of maven project as Hello world.

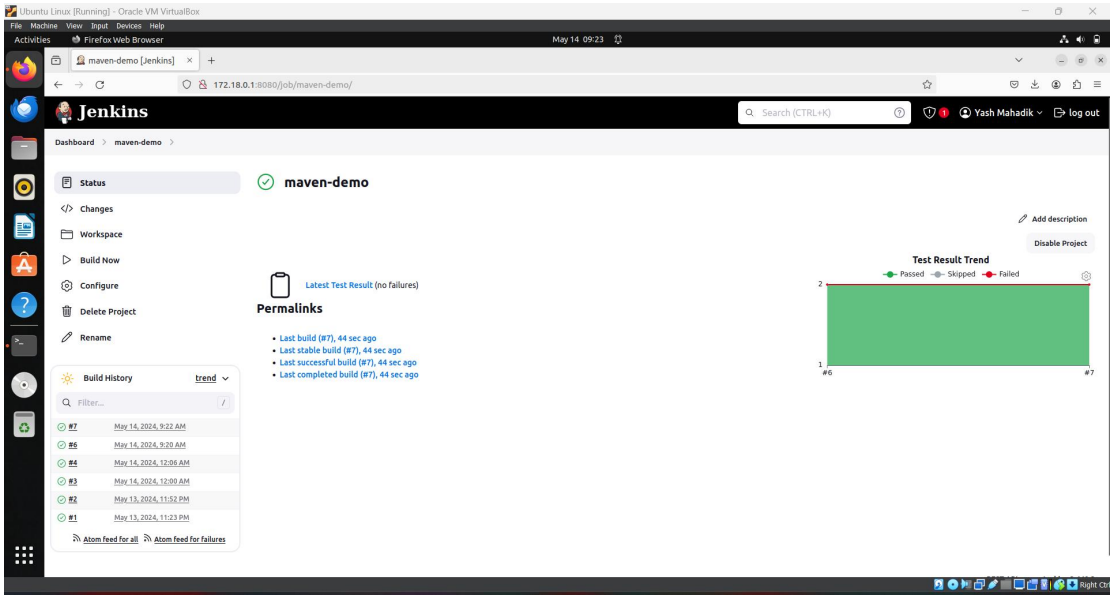


#Test Reuslt Graph with Junit

1) In project configuration section go to Post-build Actions. Select Publish junit test result report and add the location of your XML file **target/surefire-report/*.xml .** Click on Save and Observe the Graph representation.

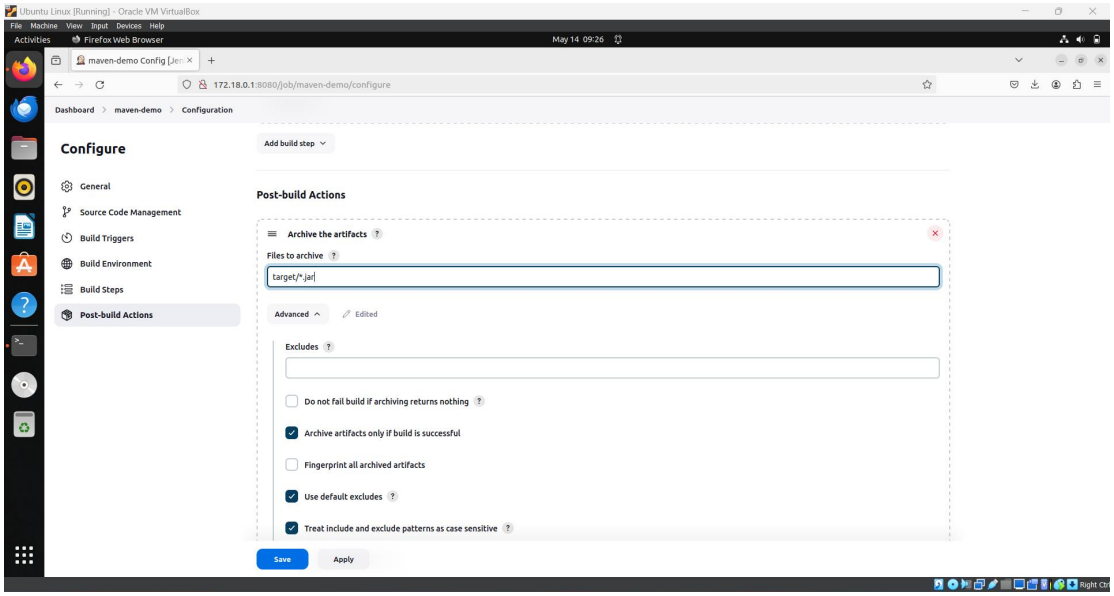


2) After building successfully you can see Graphical representation for test result.

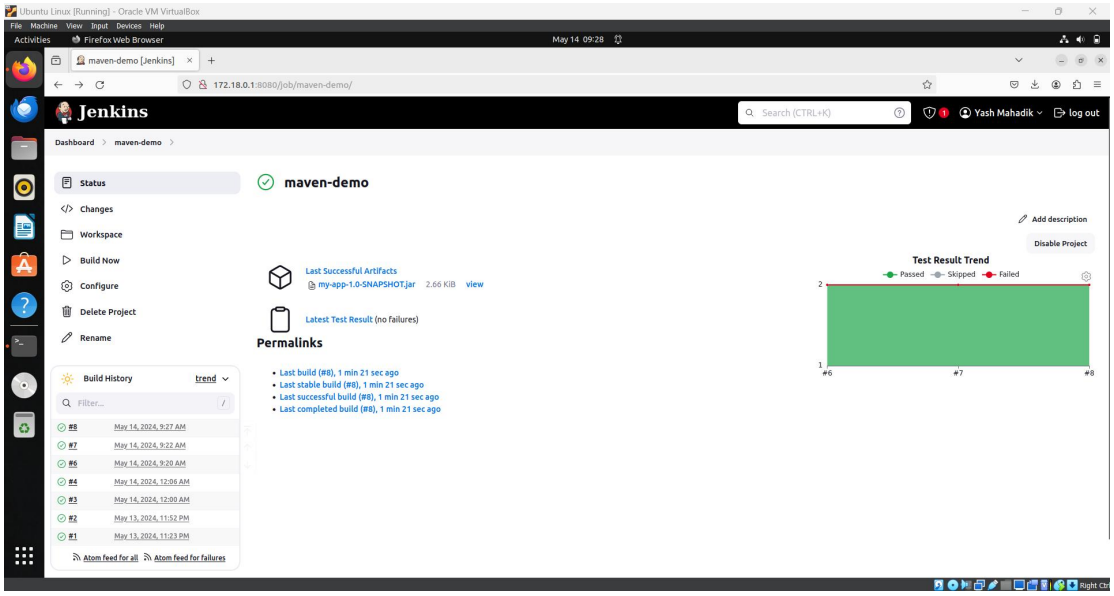


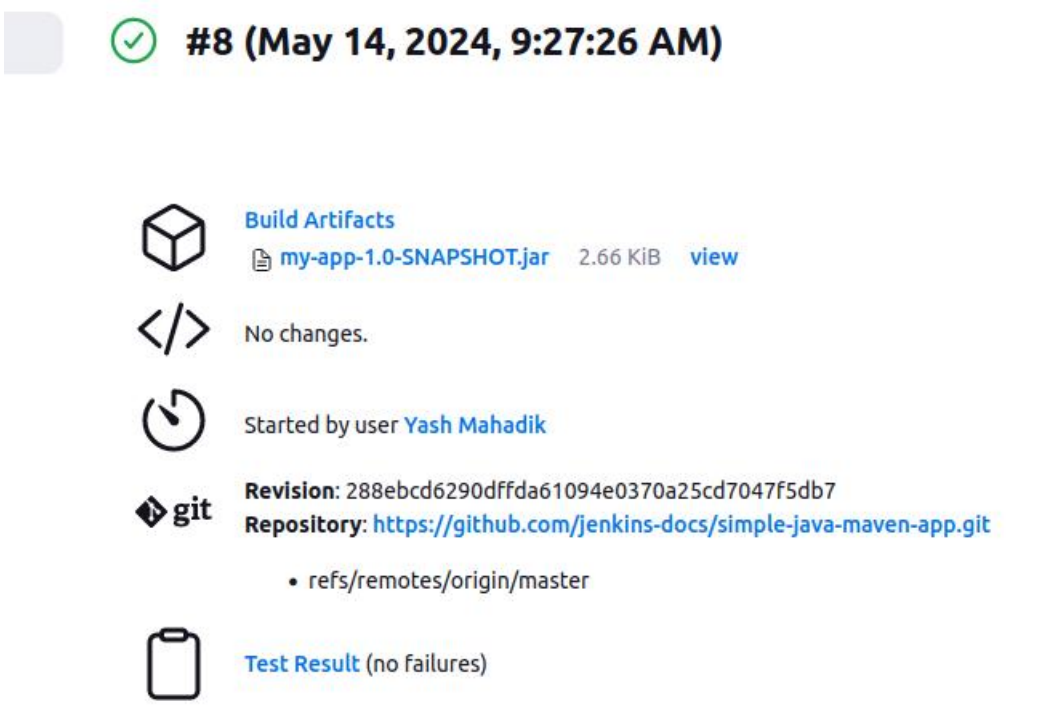
#Archive Artifact

1)In configuration section, in **post-build action**, select **archive the artifact** and add the jar file location **target/*.jar.** Mark check box for Archive artifact only if the build is successful. Click on save.



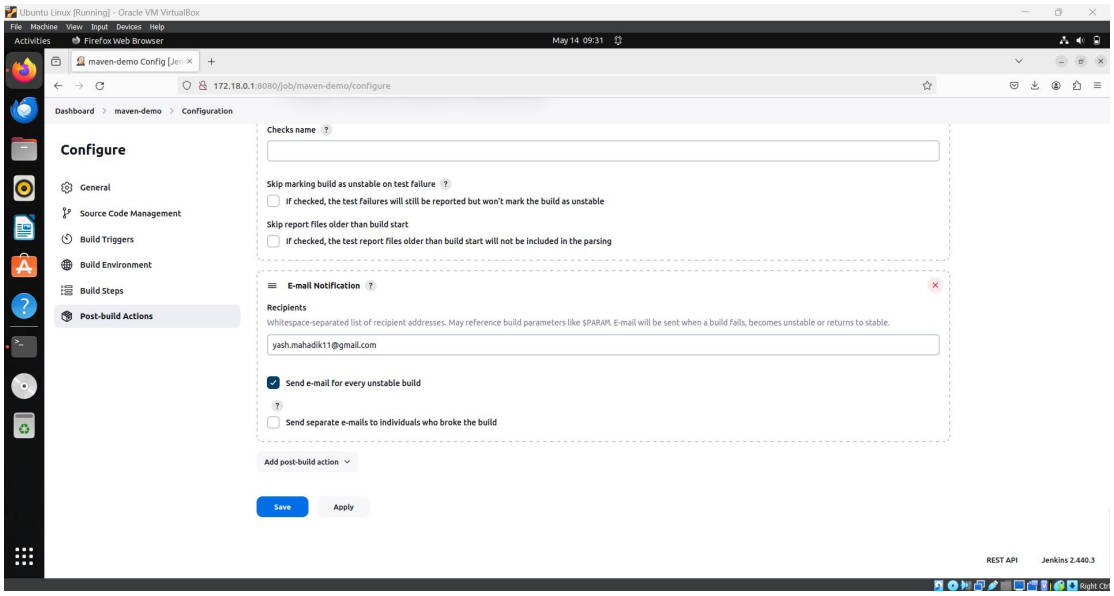
3) After Building successfully you can observe Artifact of your jar file.



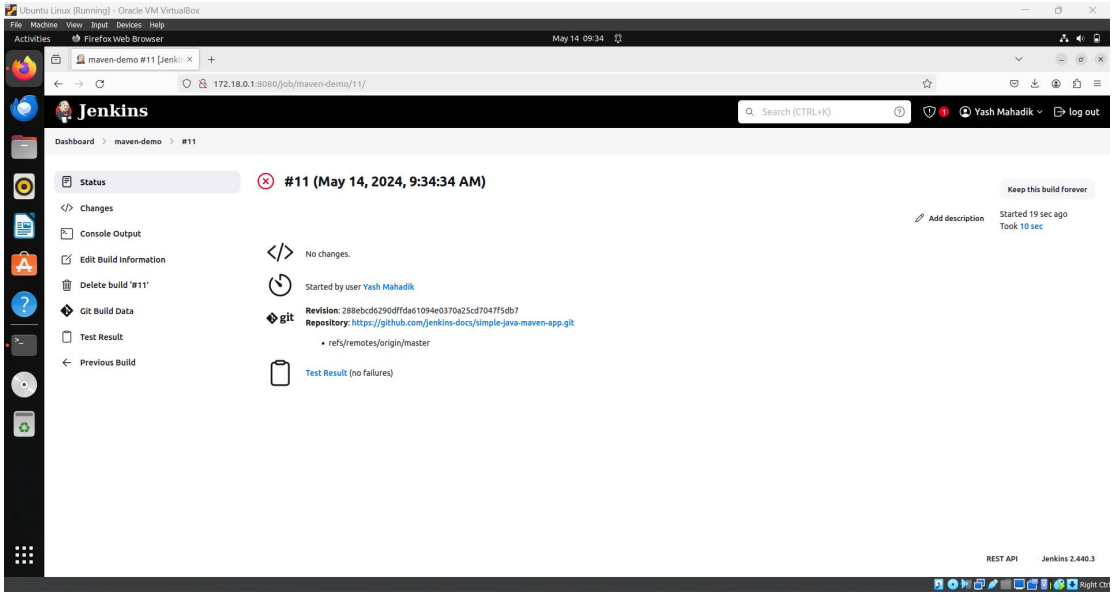


Email Notification

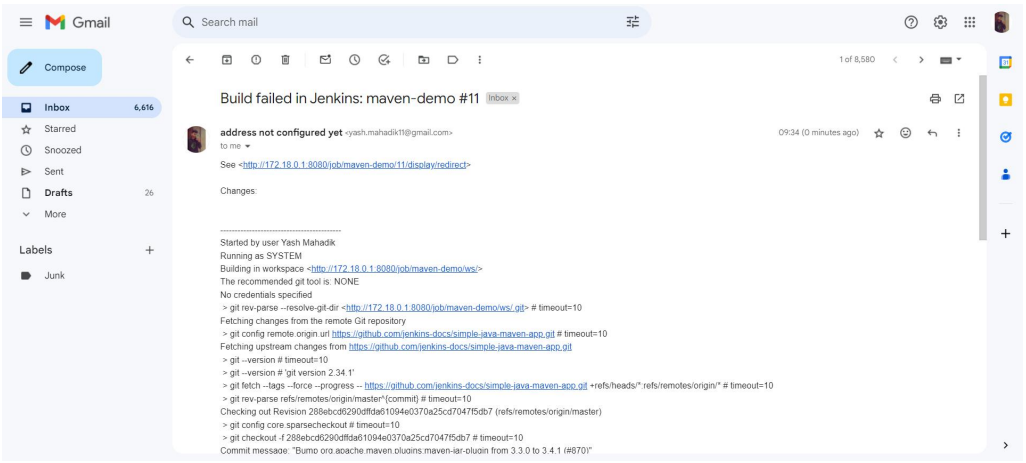
1) To get the Email notification for every Build you do.
2) Go to Configuration in the **post- build action**, select **Email Notification** and add the email id in which you want to get notified. Click on Okay (on every Unsuccessful Build you will get Email notification).



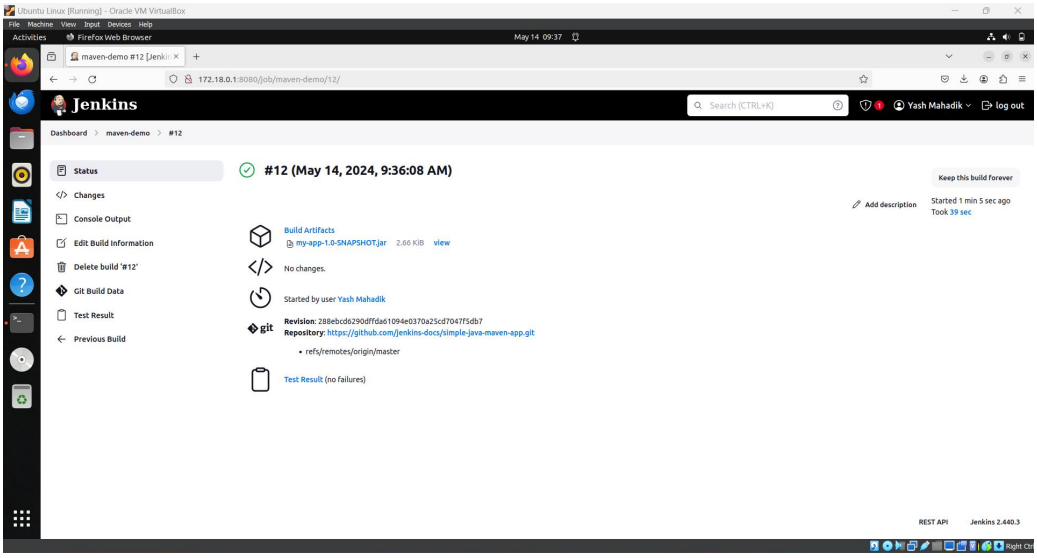
3) Build in unsuccessful.

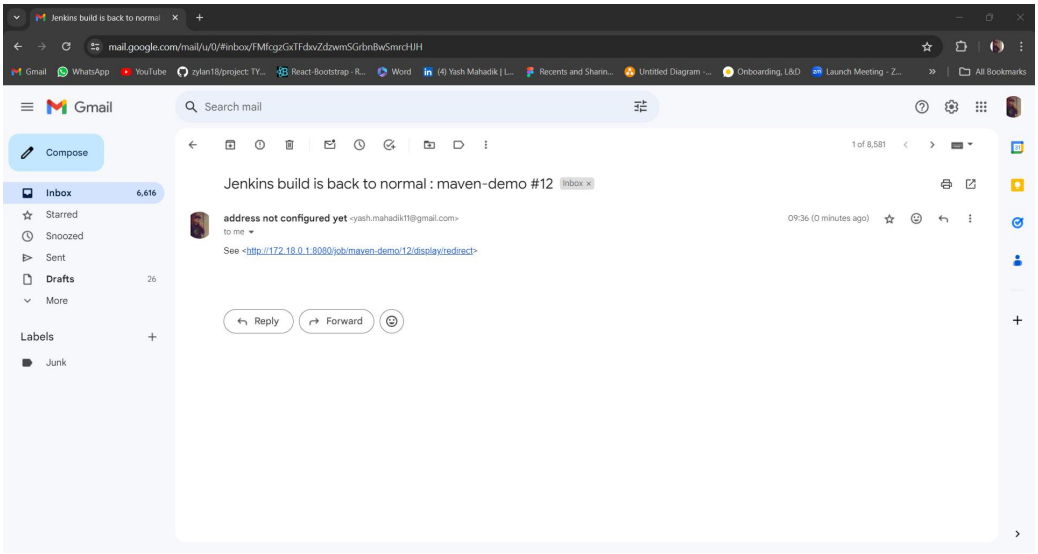


4) You can see email has been send to the email id which we have mentioned.



5) Again on Successful Build will get email notification as Jenkins Build is back to normal.





Conclusion :

This step-by-step document structure will guide readers through the process of integrating Maven with Jenkins for Java application builds, from initial setup to advanced configurations and troubleshooting. Each section provides detailed instructions and guidance to ensure a smooth integration process.