**DEVOPS CICD INFRASTRUCTURE**

**Team**

**Tejas Sudarshan, Supreet Bant, Piya Kothari, Pragya Sengar, Divin Ponnannah.**

* Deploy java program on docker hub.
* Pipeline involves,

1. Git
2. Git hub
3. Jenkins
4. Maven
5. Docker

* Git is a version control system used to track the changes made in source code in repository.
* Git hub allows us to collaborate to work on code and save code online and track changes in code, define workflows to automate tasks.
* Jenkins is used to integrate Git, pull the code from GitHub repository, monitoring and tracking the build progress, building, and deploying docker image.
* Maven trigger builds, run tests and produce artifacts, uses pom.xml which have project information in it to perform consistent builds.
* Docker is used to build the docker image of java application from docker file and publish it onto docker hub.

**Stage -1**

**Building project – maven**

Maven is more than just a build tool. It’s a project management tool that encompasses a project object model (POM), a set of standards, a dependency management system.

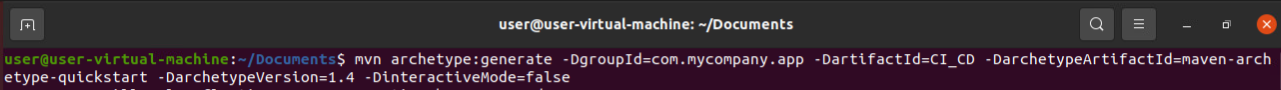
**mvn archetype:generate -DgroupId=org.website.codekru -DartifactId=DemoProject -DpackageName=org.website.codekru -Dversion=1.0-SNAPSHOT -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false**

**mvn** is the Maven2 command, and **archetype:generate** is a Maven goal. The -**Dname=value** pairs are the arguments passed to the goal, like what we do while passing system property options to the JVM via the command line

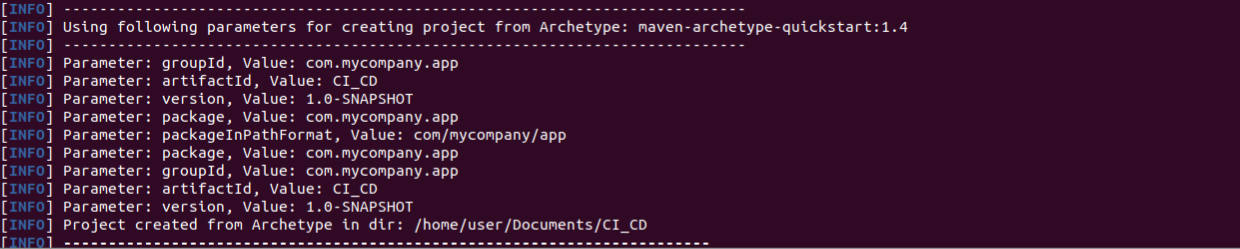
**archetype:generate** helps create a project from an archetype that we have defined using -**DarchetypeArtifactId=maven-archetype-quickstart.** If we do not define the archetype using this argument, then it will prompt us to choose an archetype from a list of archetypes. Many archetypes are available in Maven almost for anything like creating a web application.

Maven archetype plugin will create a folder or directly named DemoProject, the same as we passed in the artifactId argument in our command.

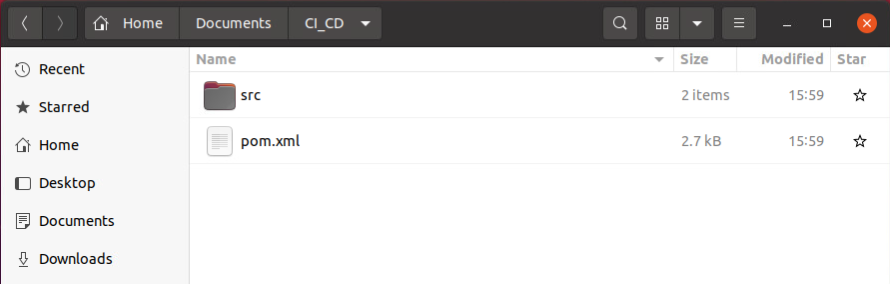
**-DinteractiveMode=false** argument will disable the interactive mode. If we don’t use this argument, then the interactive mode will be enabled, and it will ask us to confirm the project settings before creating our project. The project will only be created after we give our confirmation.



STEP-1: Create a HelloWorld Project using mvn command.



Project folder is successfully built.



Generated project folder has src and pom.xml file.

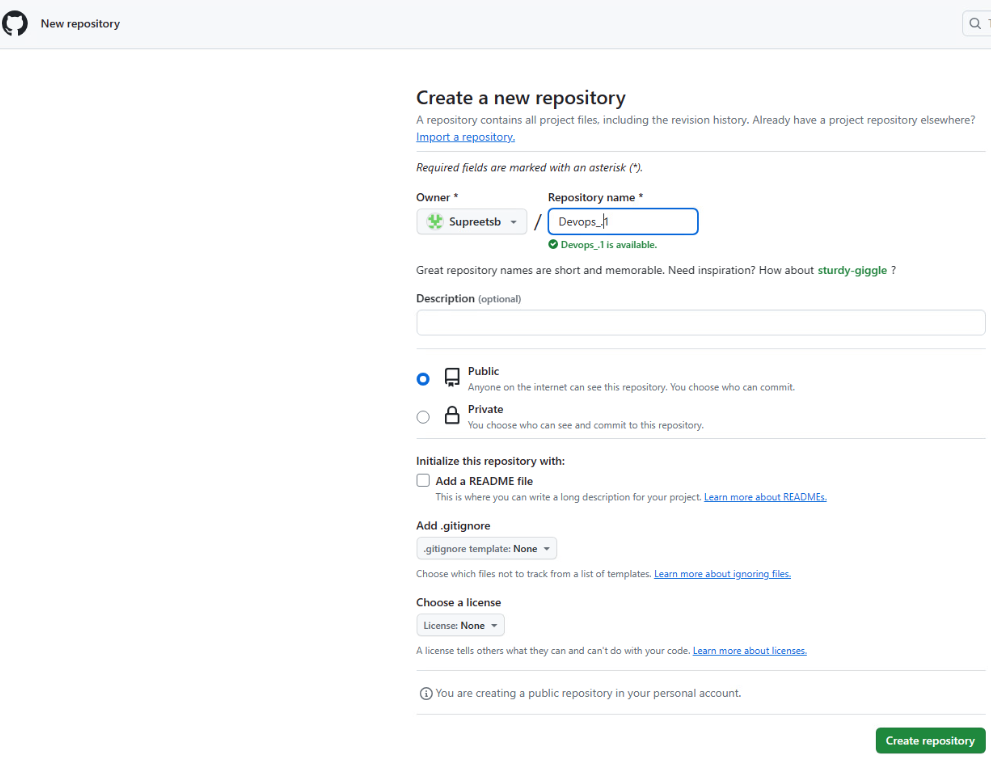


* We can see the HelloWorld code in App.java file.



STEP-2: Add dockerfile and jenkinsfile in the CI\_CD folder.

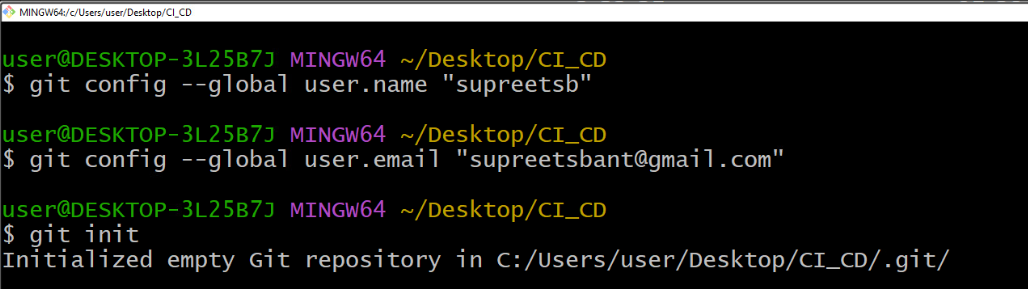
**Github**



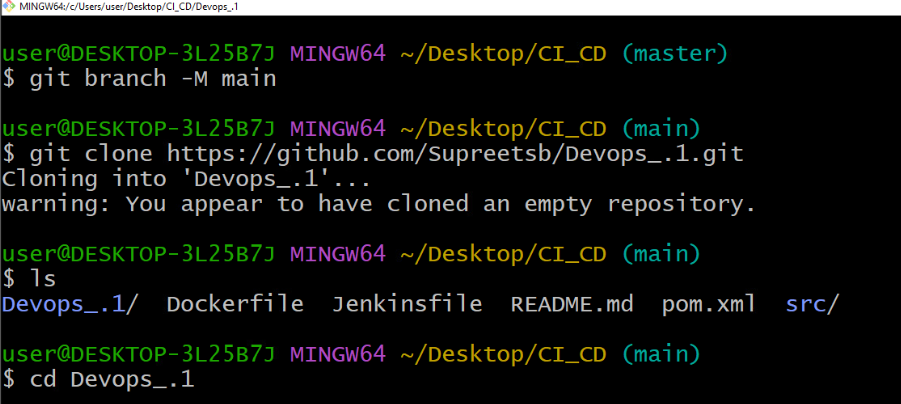
STEP-3: Create a new GITHUB Repository

Git

STEP-4: Got to CI\_CD folder, right click and open gitbash.



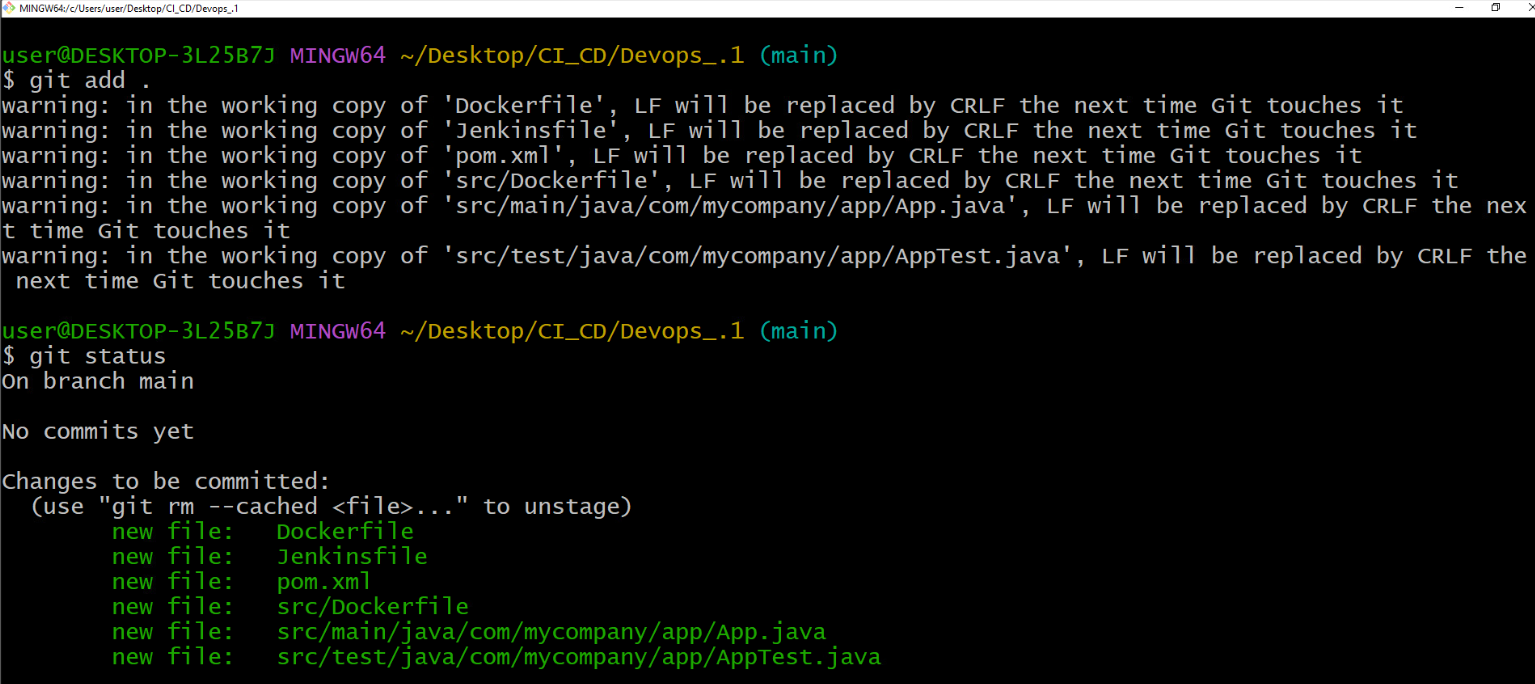
STEP-5: login to GitHub and initialize the repository.



STEP-6: Switch from master to main using “git branch”.

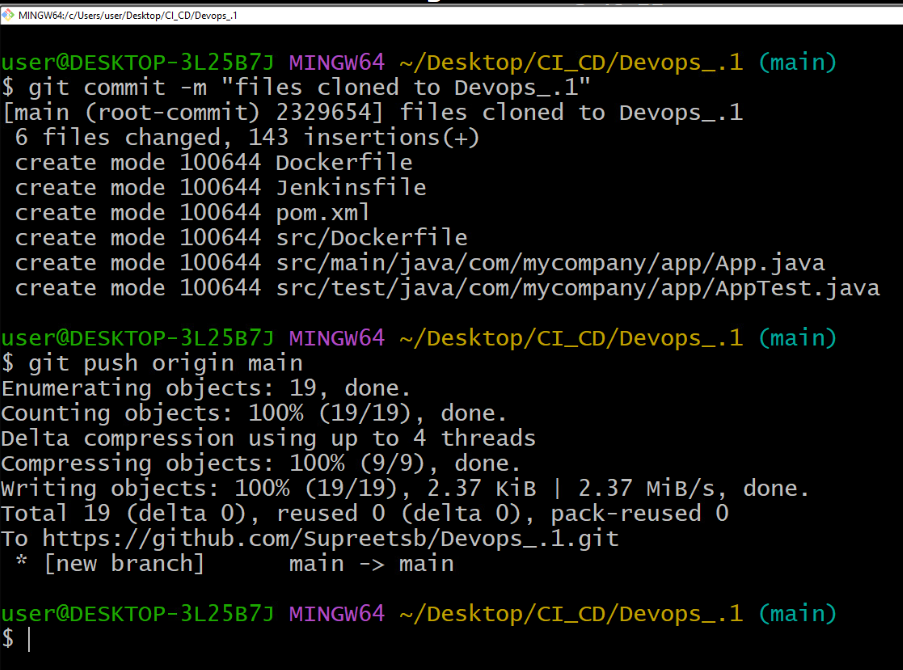
STEP -7: Clone the Github repository on to the local CICD folder.

STEP-8: cd to Devops1 folder

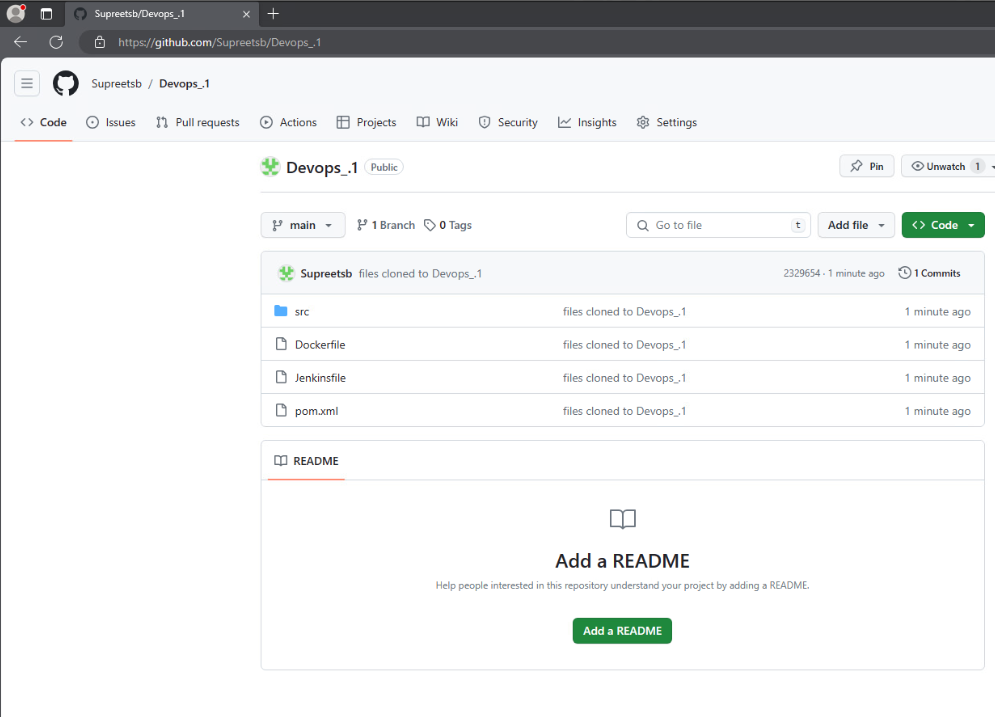


STEP-9: Add all the files using “git add”

STEP-10: check the commits using git status.



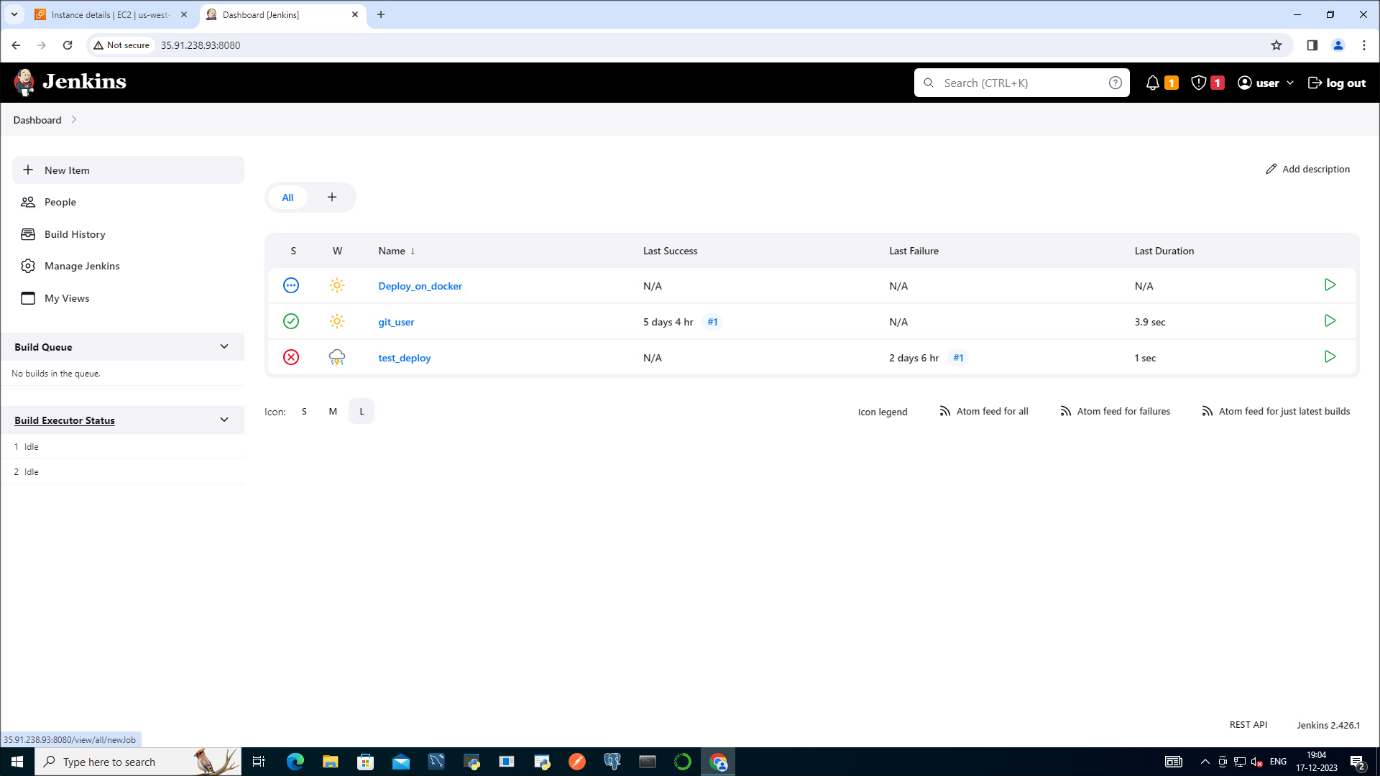
STEP-11: Save changes by performing “git commit”.



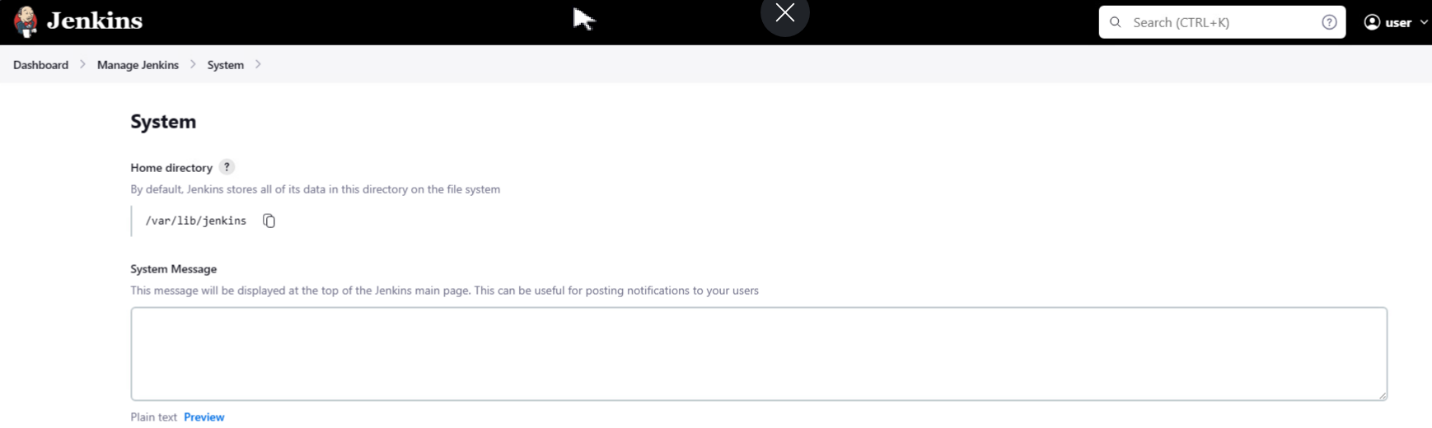
STEP-12: Check the repo for the added files.

**Stage-2**

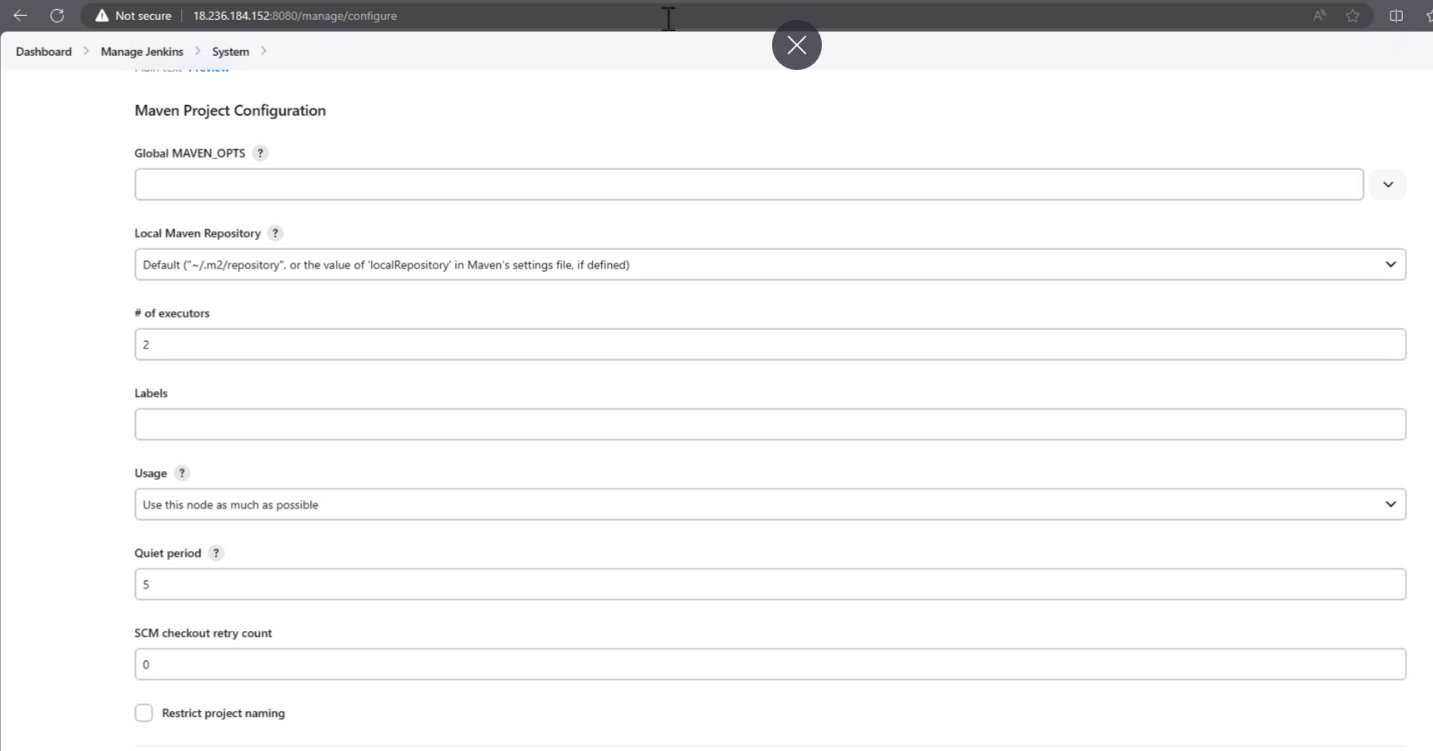
**Jenkins Setup**

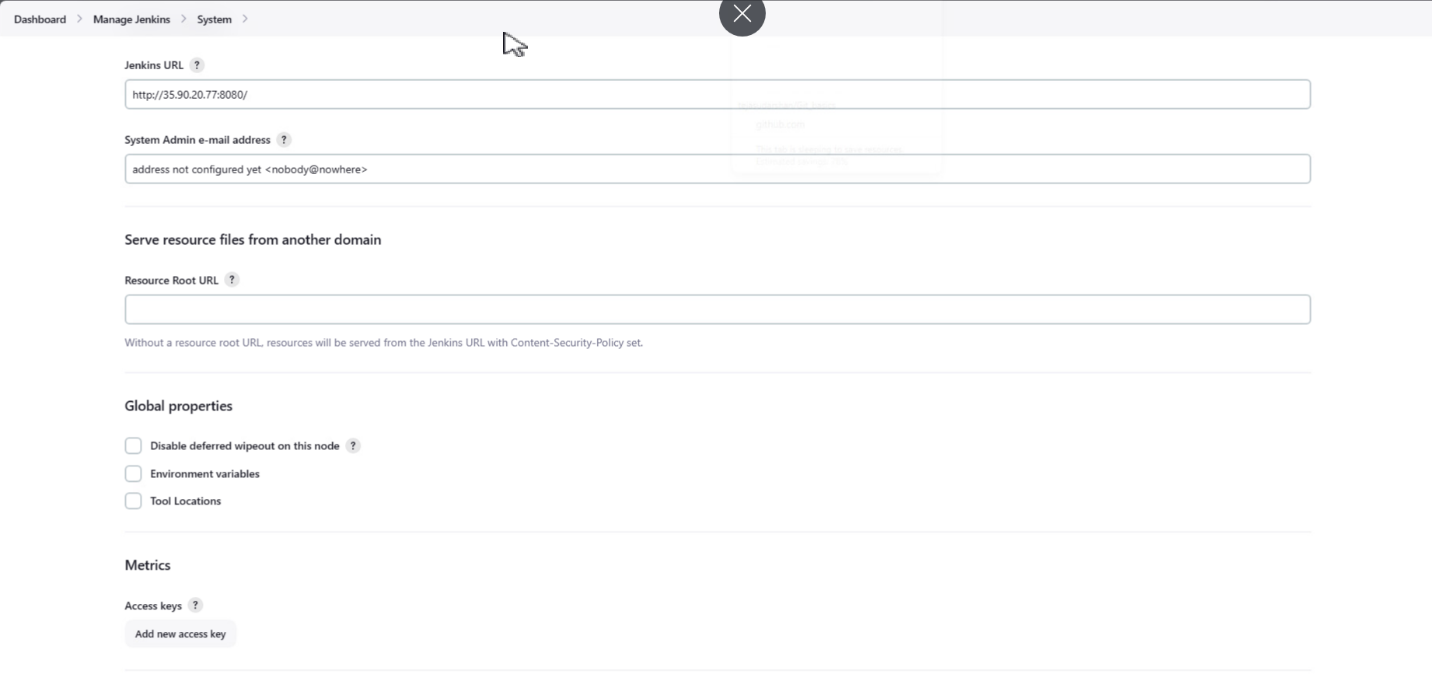


System settings in Jenkins

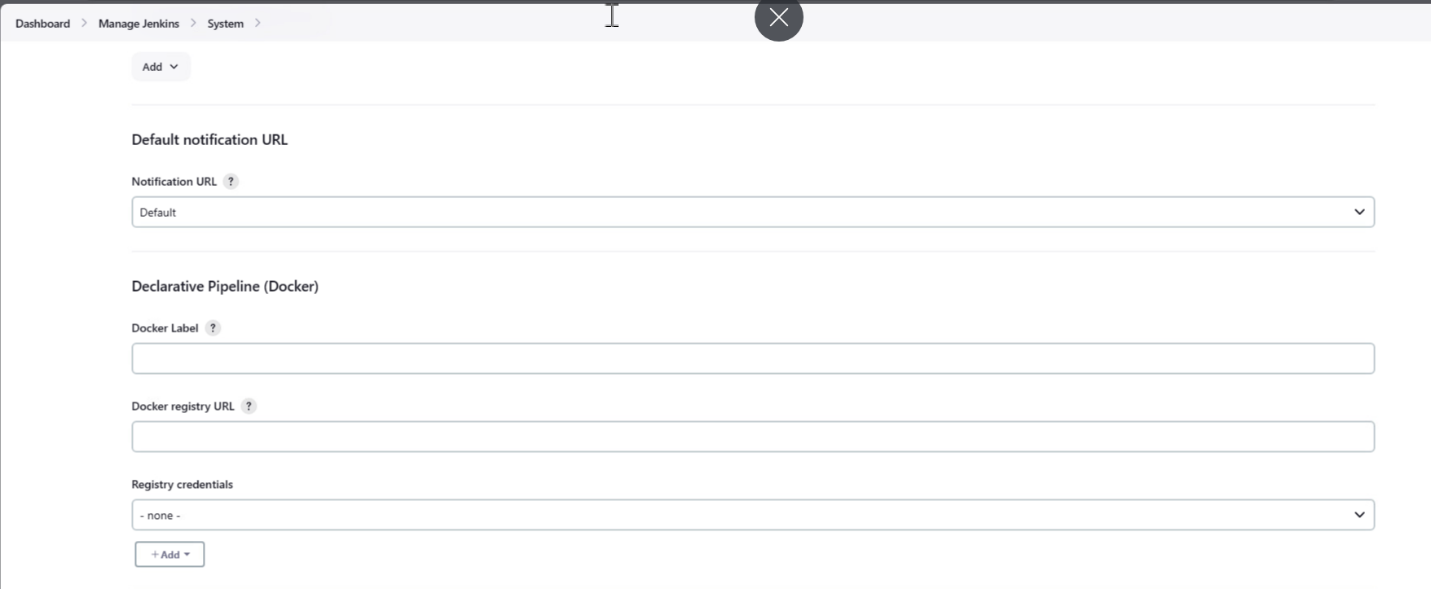


* **Maven project configuration:** no changes needed to be done here.



Jenkins URL: which is automatically created

Default notification URL: no changes

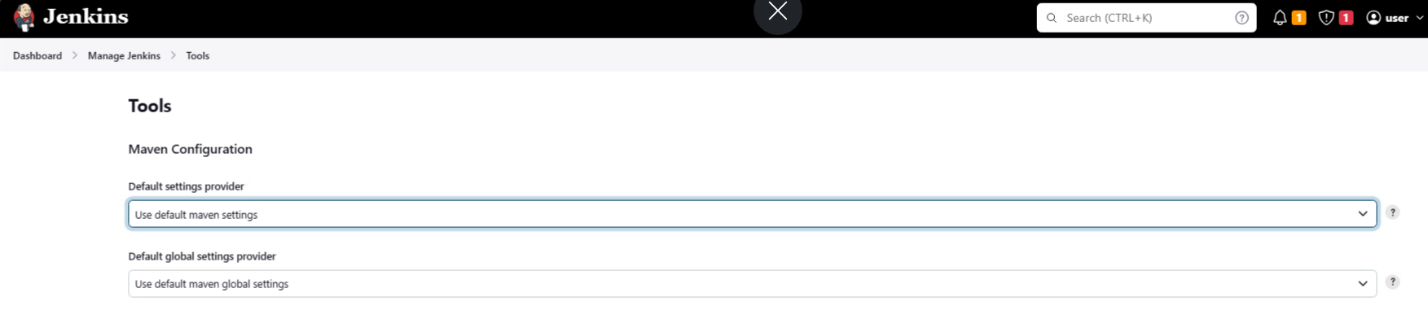


Publish over ssh: here we need to add the password of the aws jenkin instance.

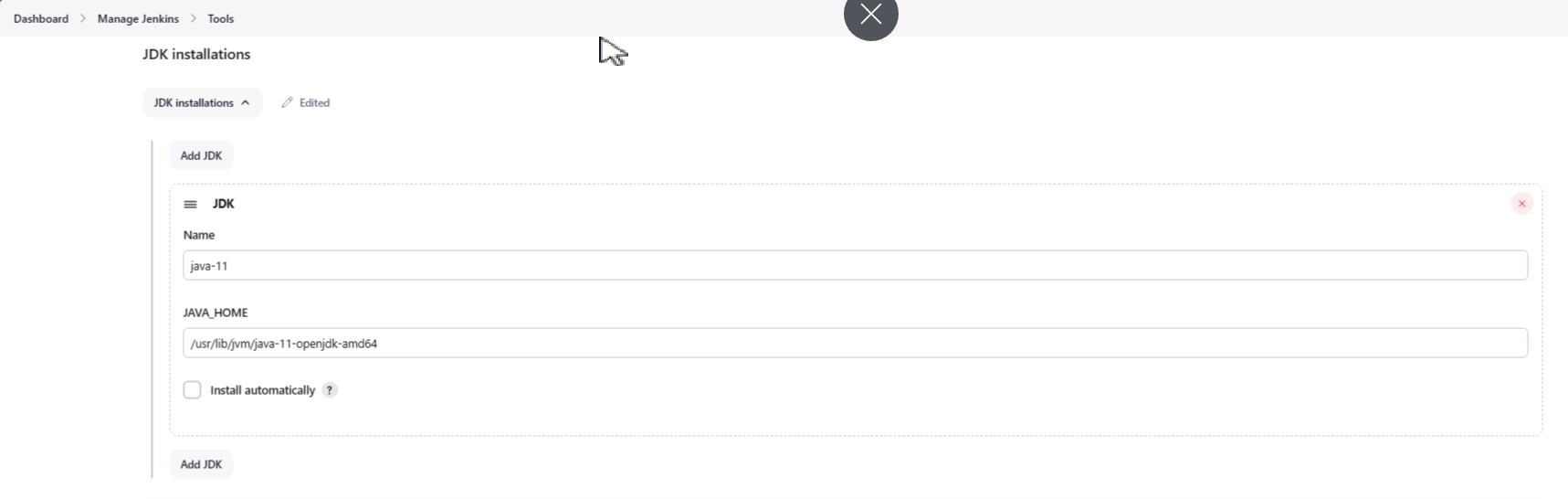
In ssh server we need to add docker and its credentials.



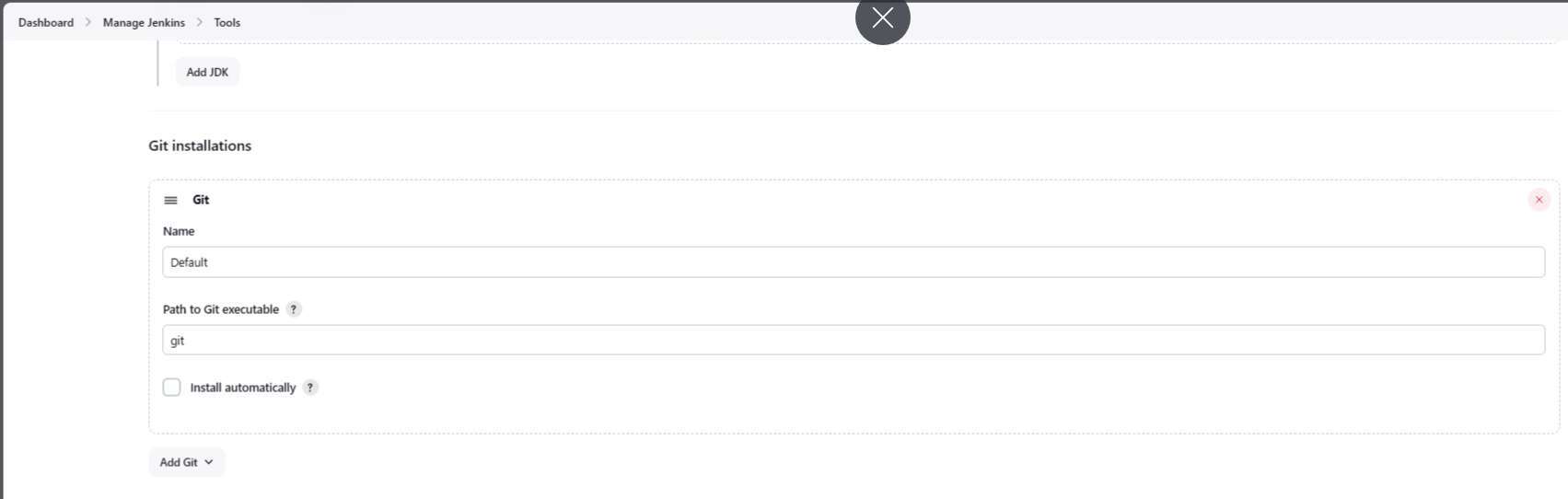
Tools on Jenkins



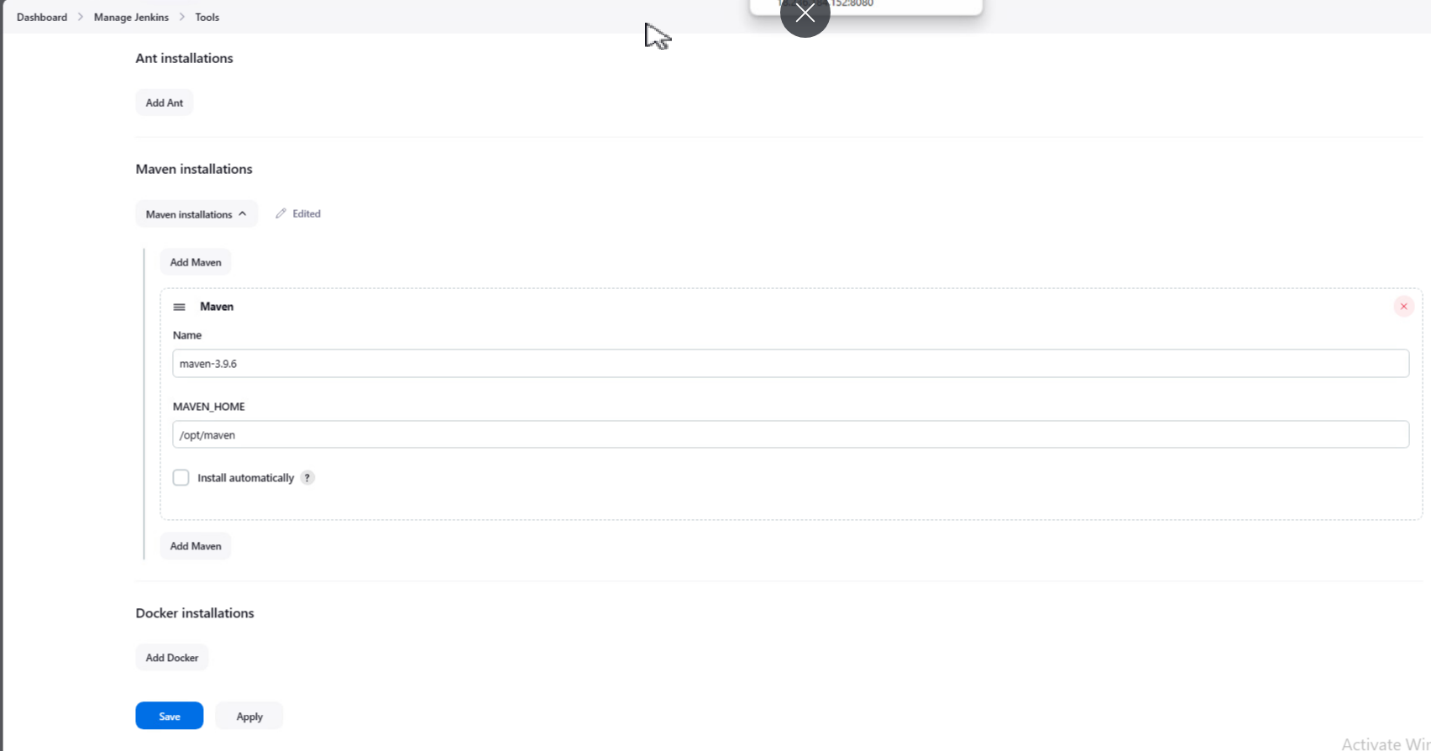
* JDK installations: we need to add java version and path where java is installed in the aws instance.



* Git installations: no changes need to be done here



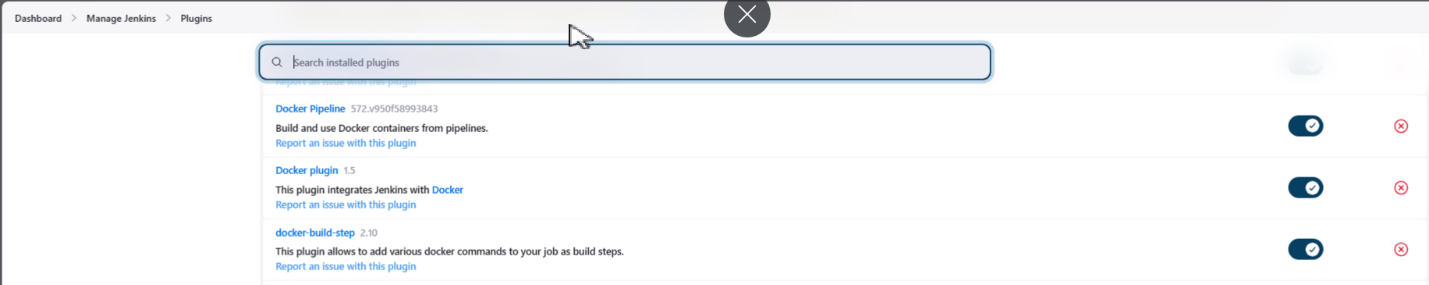
* Maven tool addition: maven version and path need to be added in the respective fields.

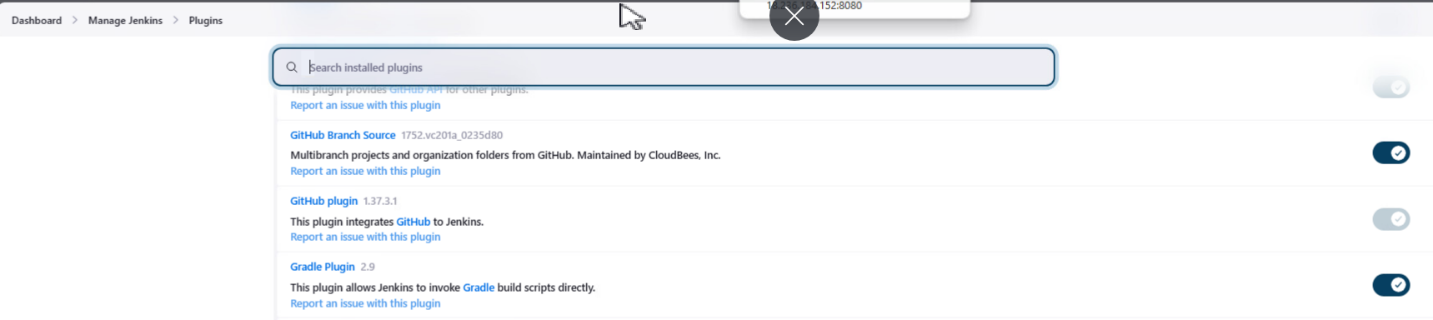


* And then apply and save it.

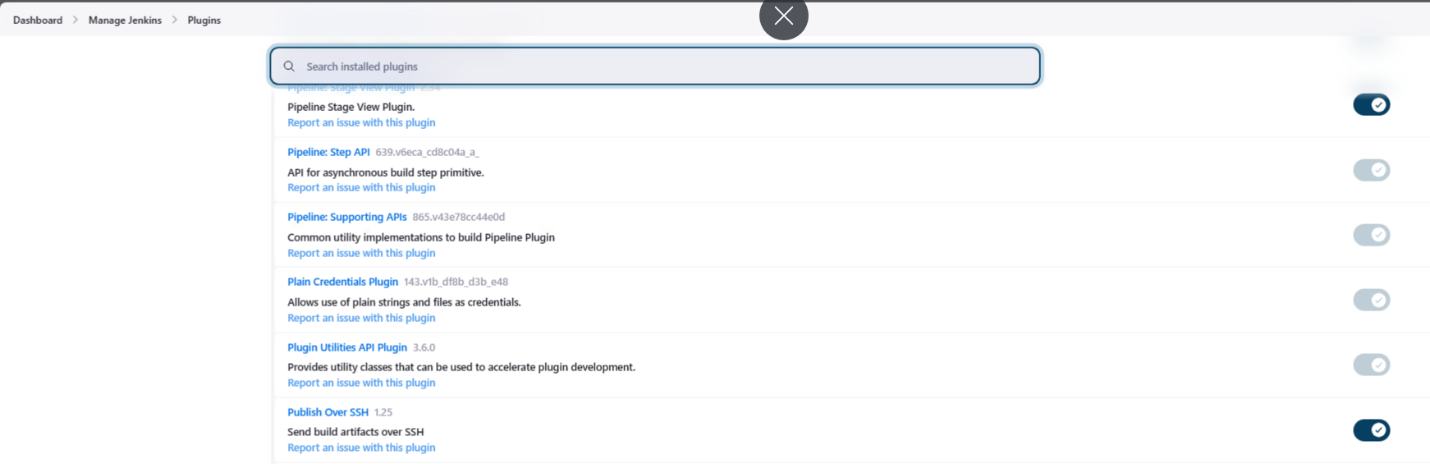
**Plugins needed for the project.**

Docker plugin

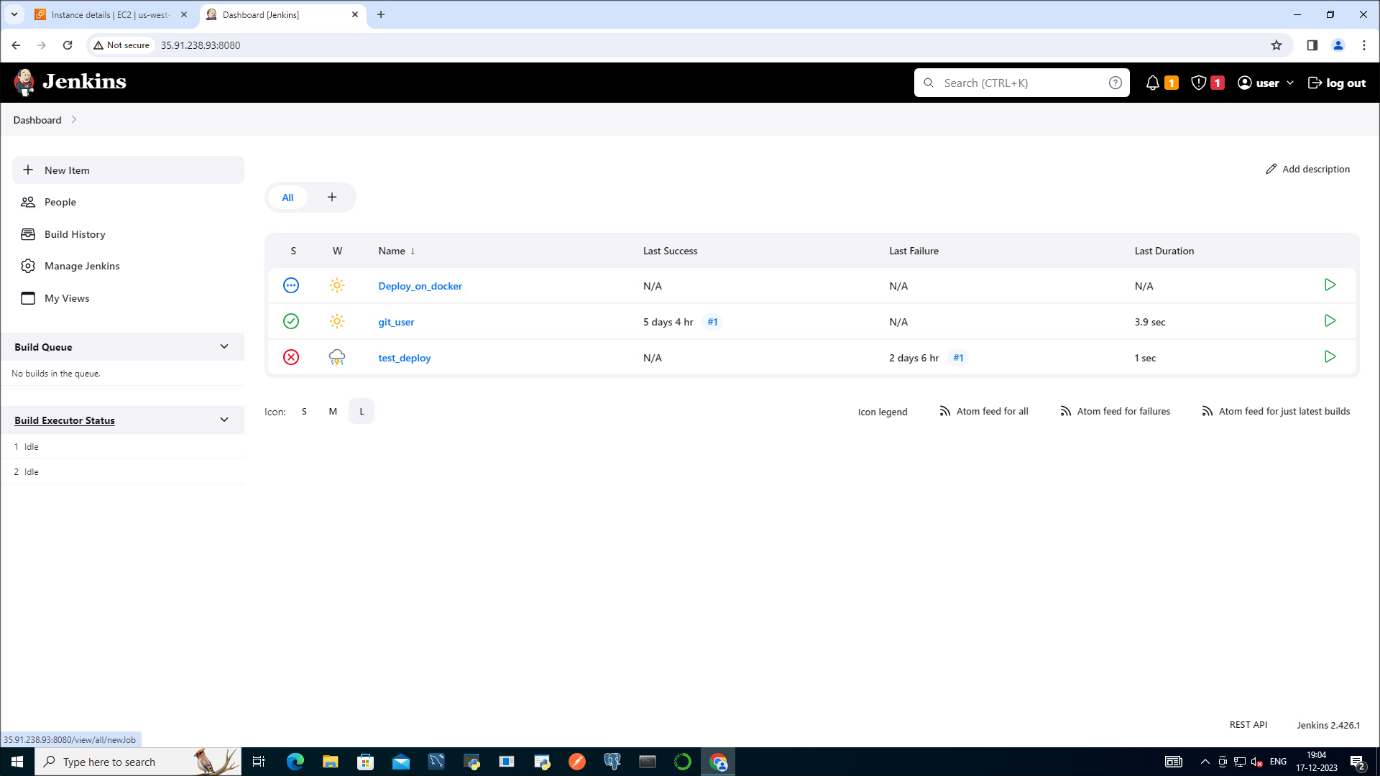


Git plugin

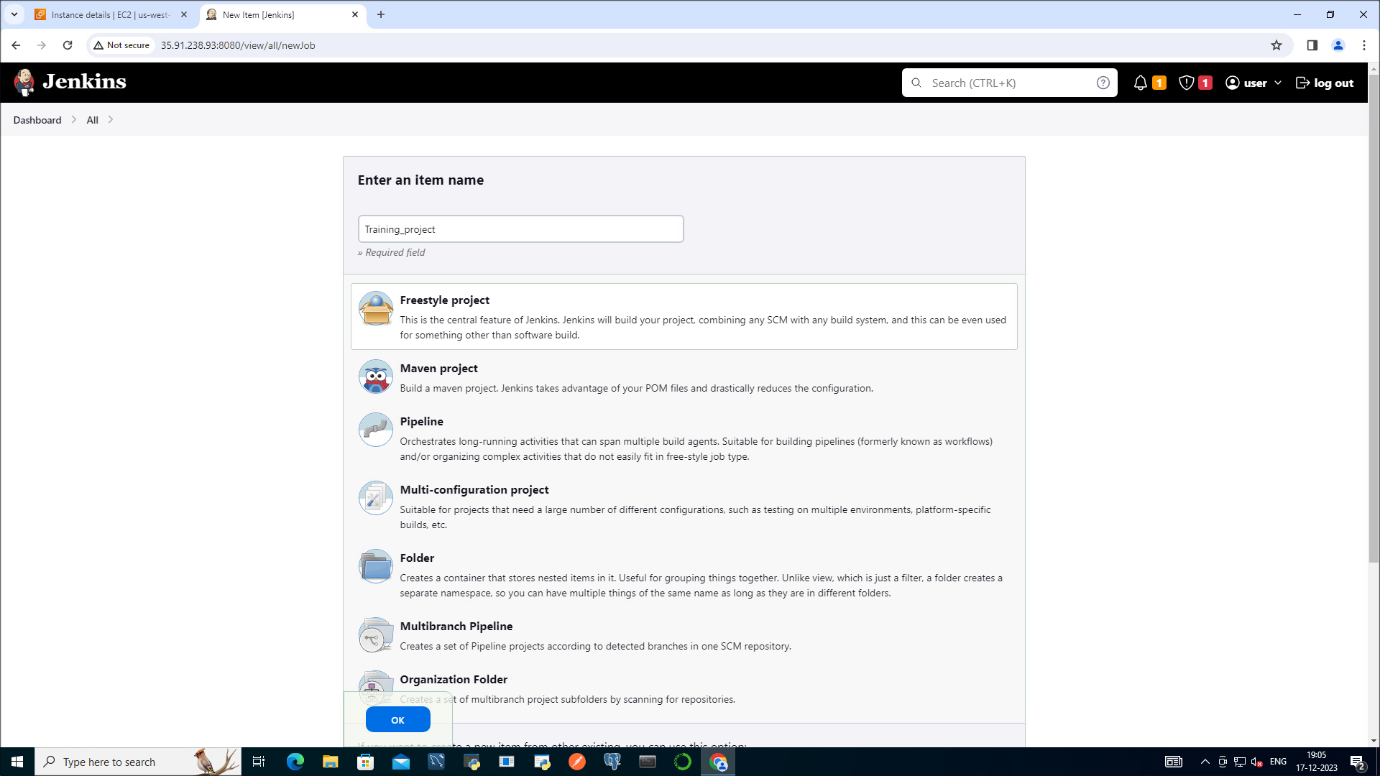
Pipeline and ssh plugins



STEP-1: Create a new item

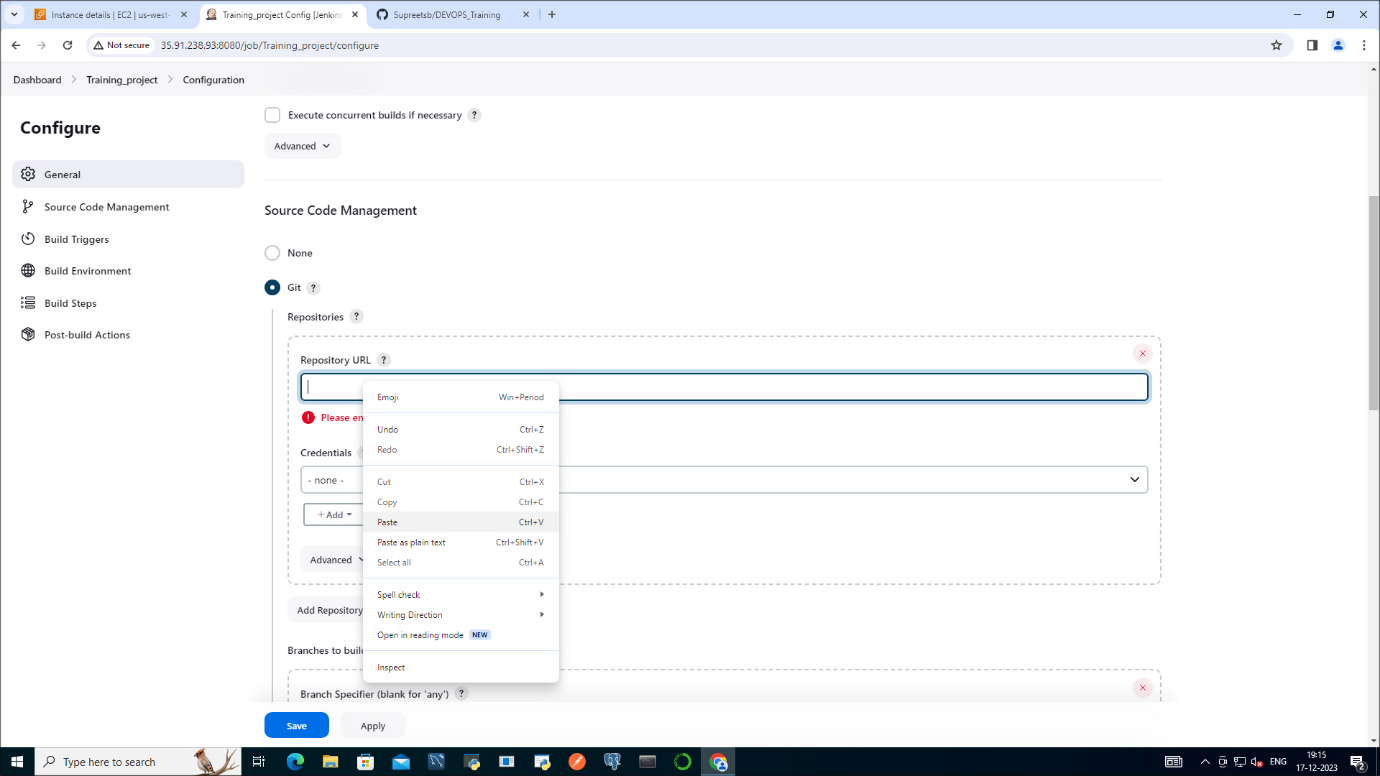


STEP-2: Create a freestyle project for git-jenkin integration.

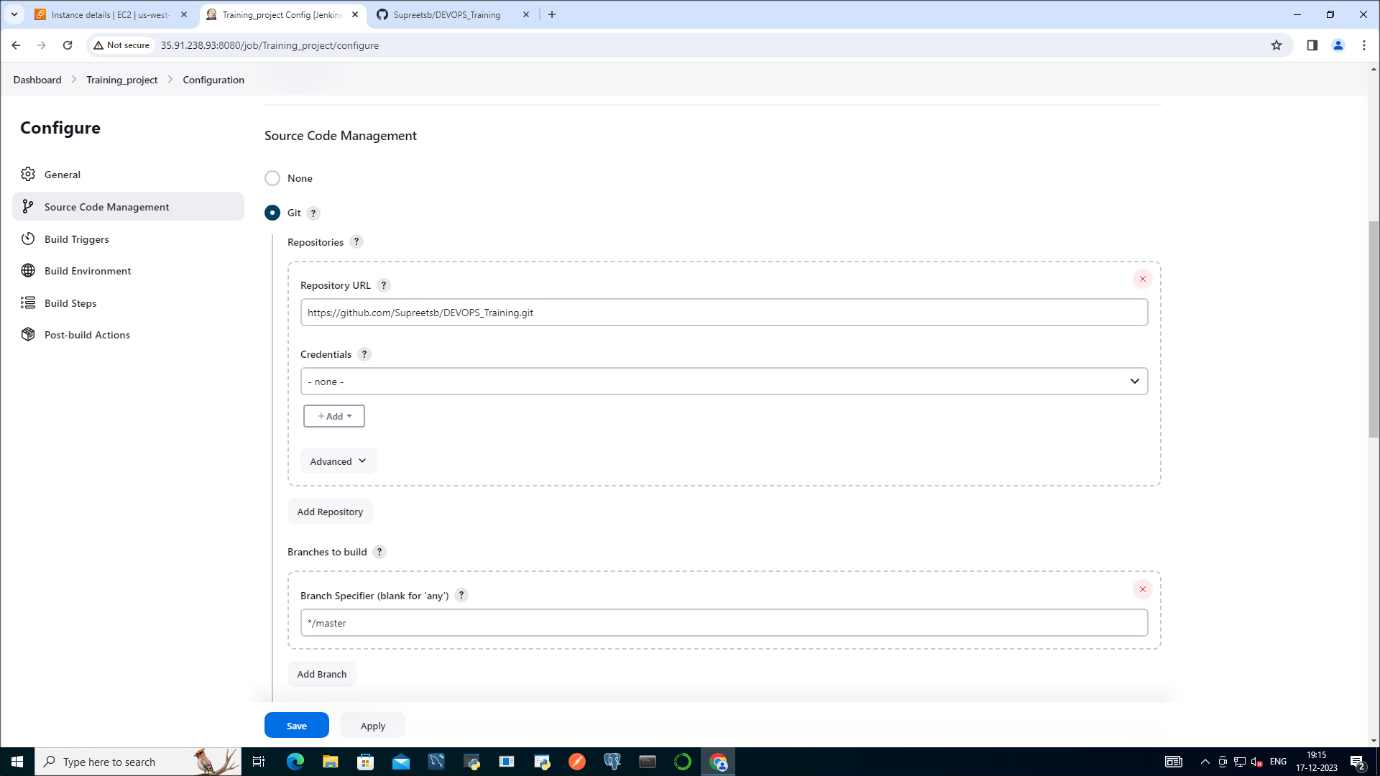


* Creating a project will navigate to configure page.

STEP-3: Copy the GitHub project repository link and paste it in the source code management.



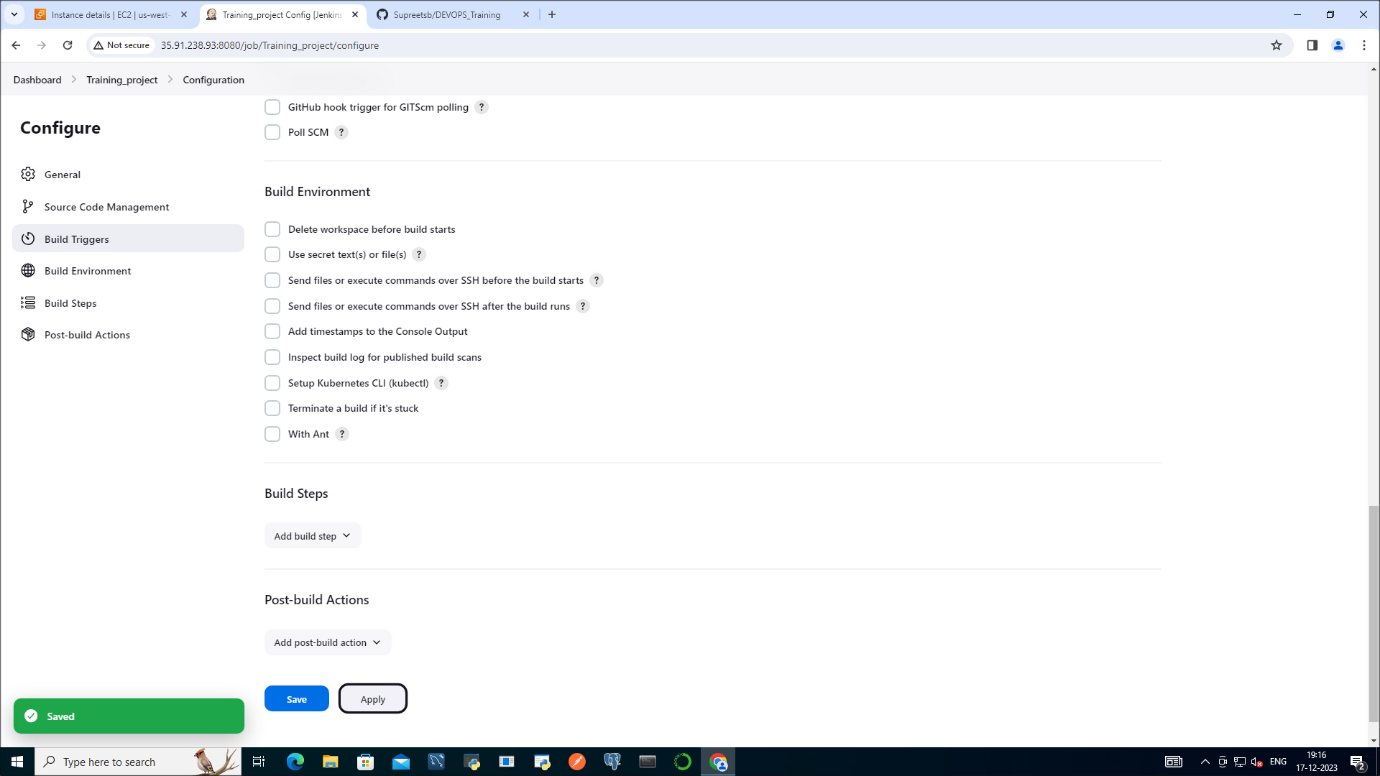
* No credentials be given in the credentials field.



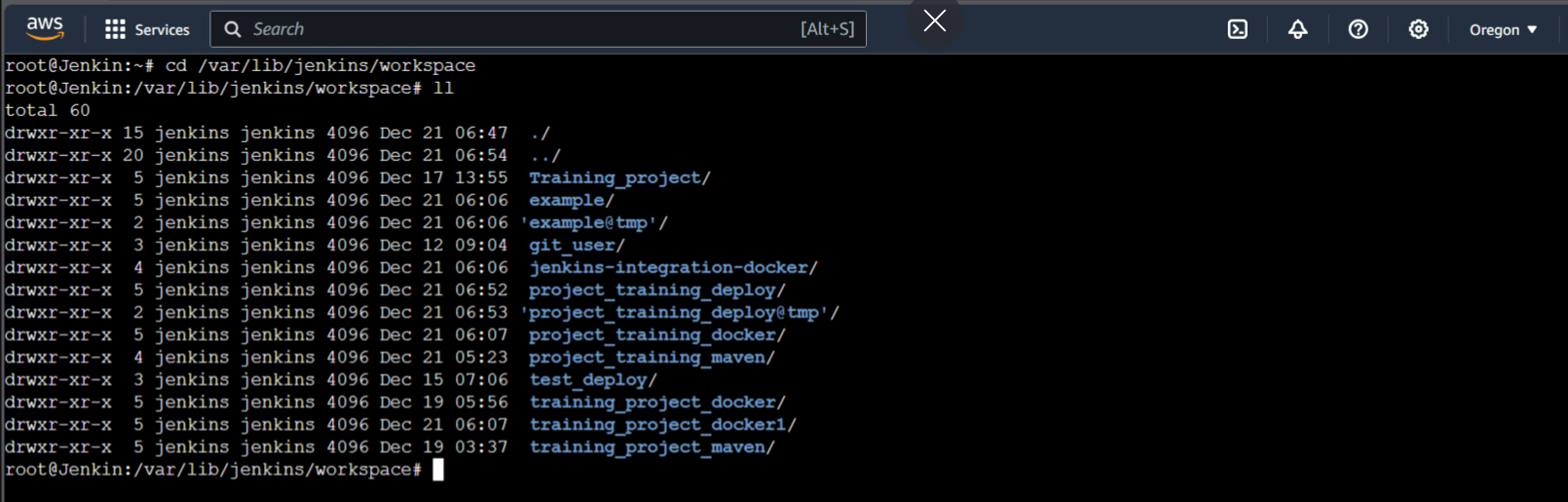
STEP-4: change the branch to main.



STEP-5: Click on apply and save it.



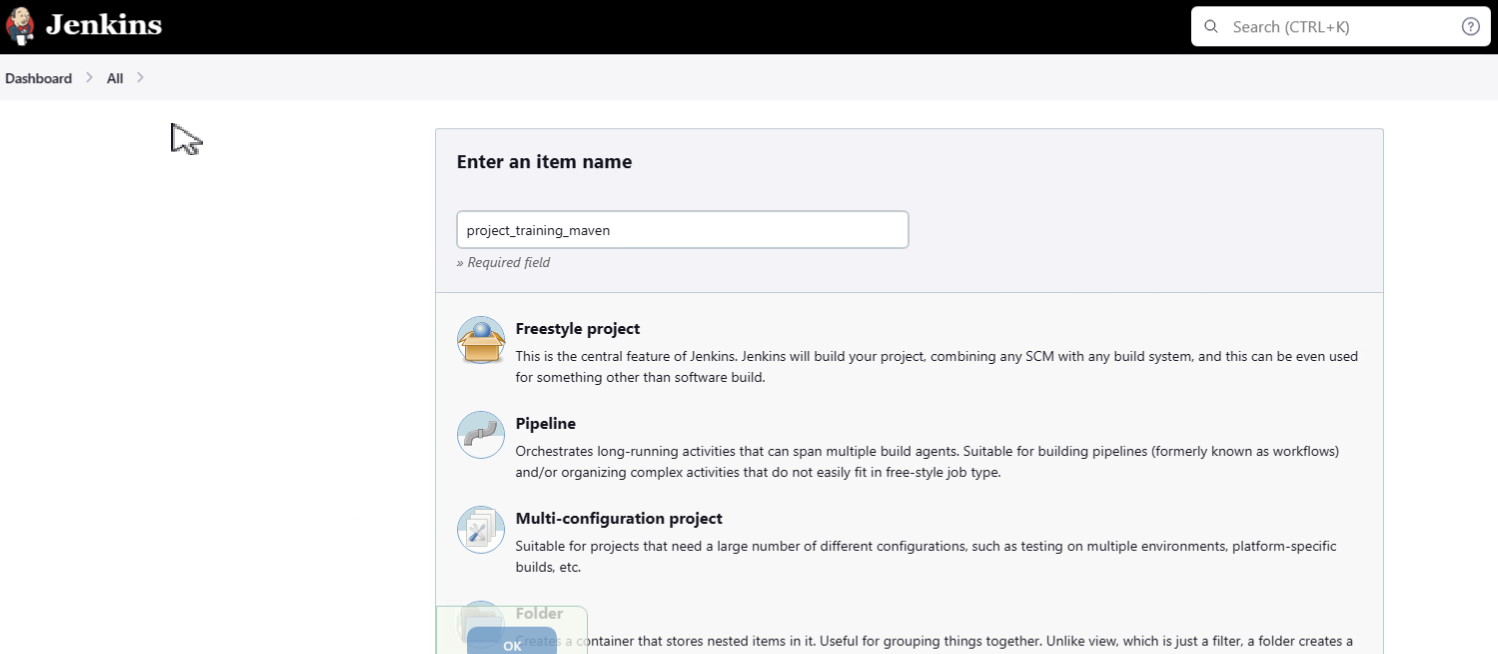
STEP-6: After Building the project Training\_project folder is created on Jenkins workspace.

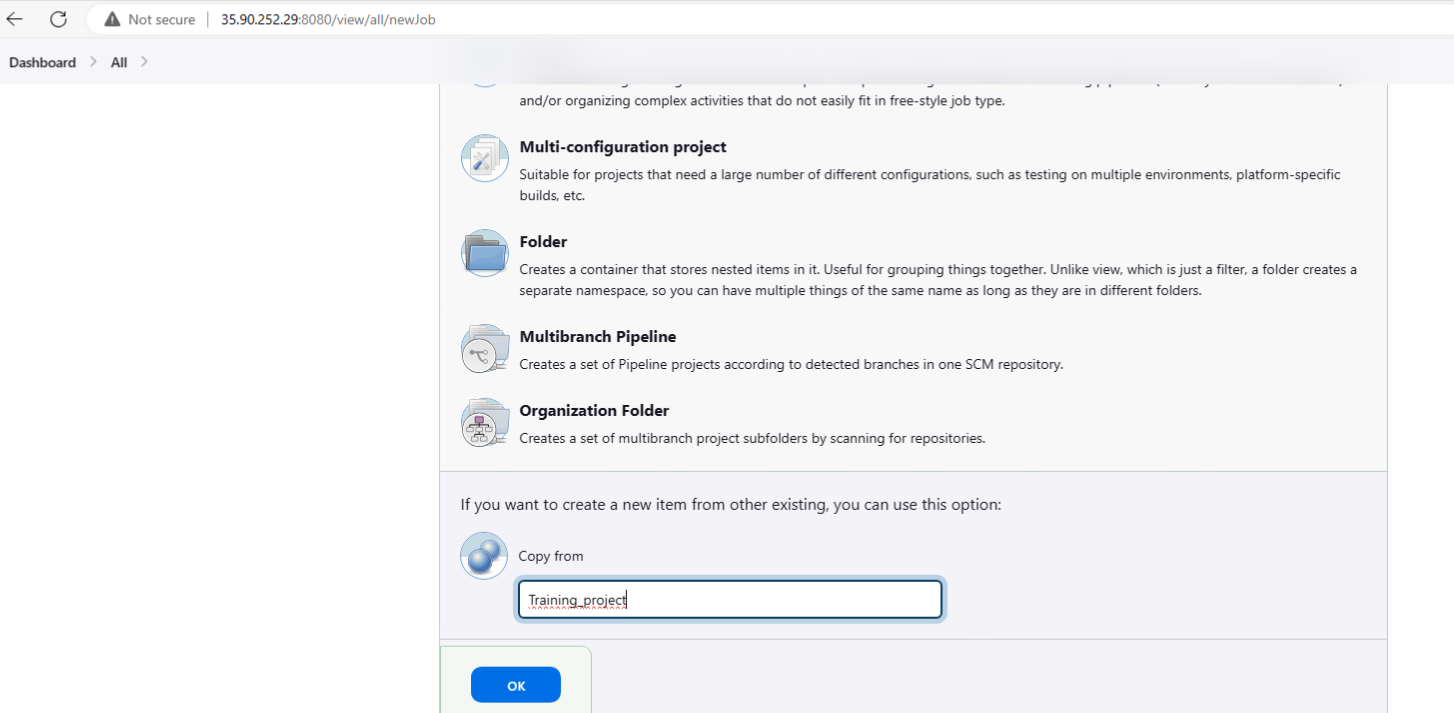


**Stage-3 Build the Project**

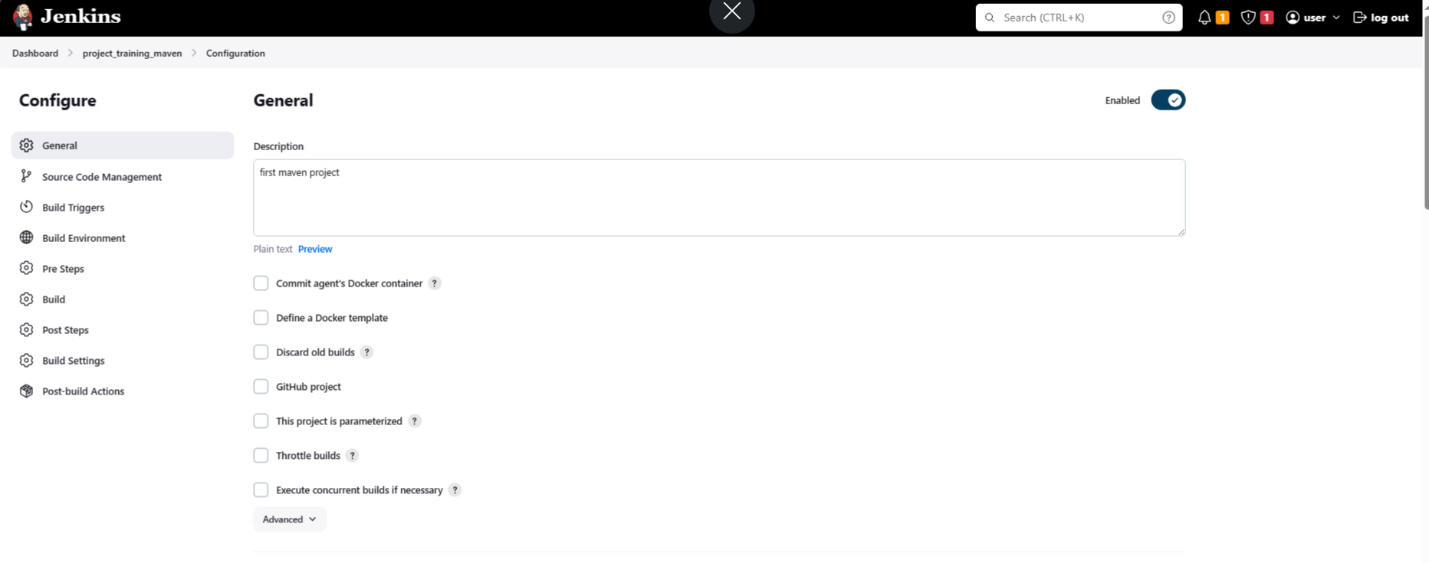
STEP-1: Create maven project.

* Instead of freestyle project click on copy from to use configuration of existing “git-jenkin” project.

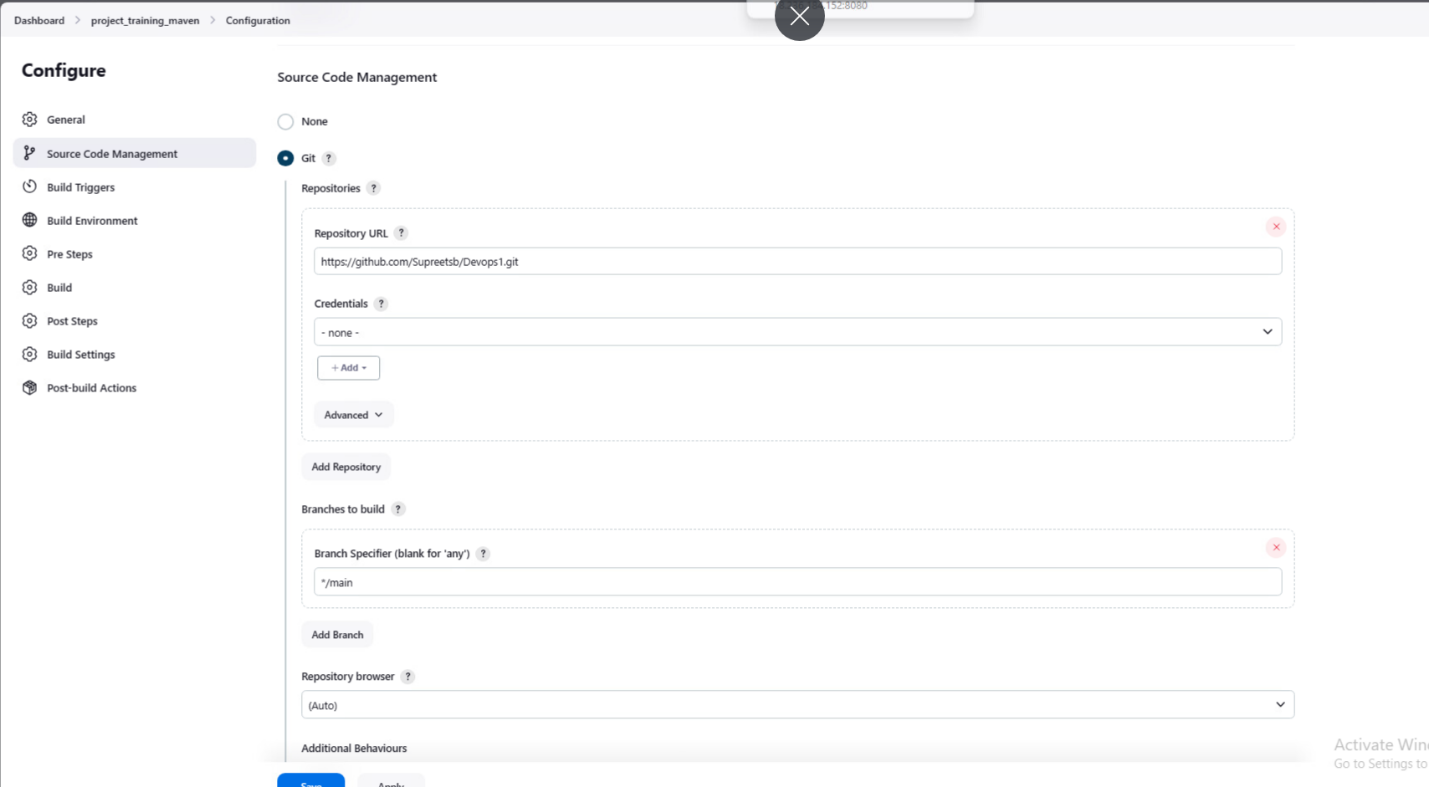




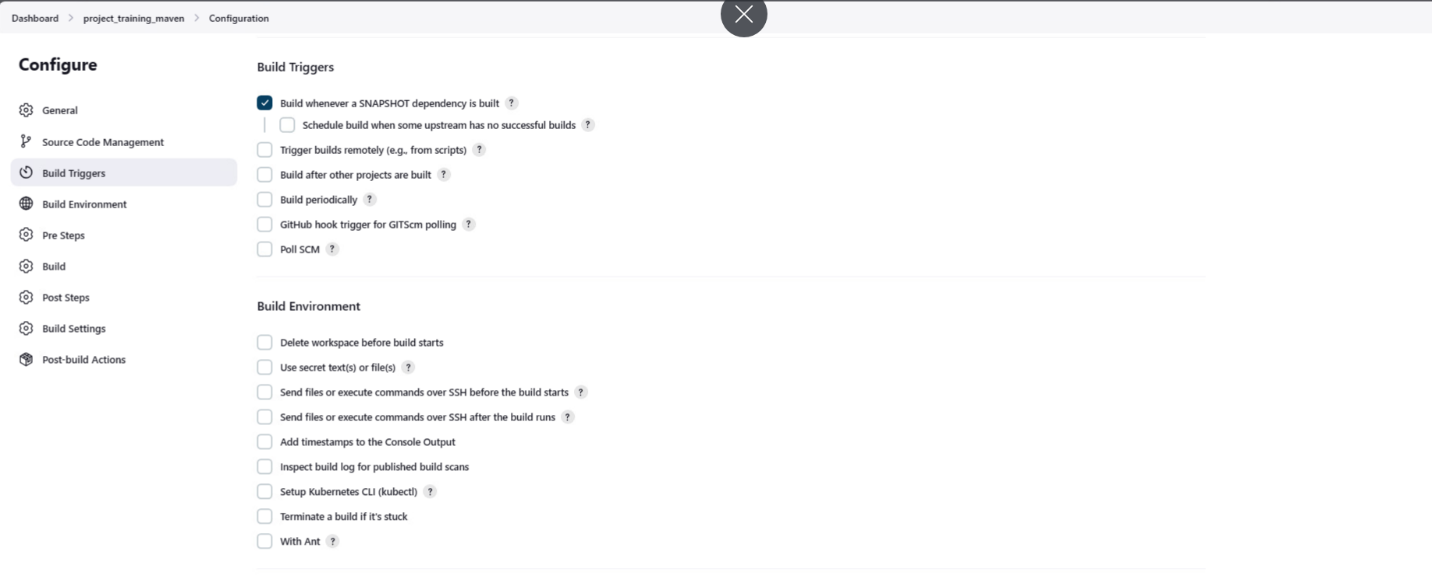
**Project configuration**



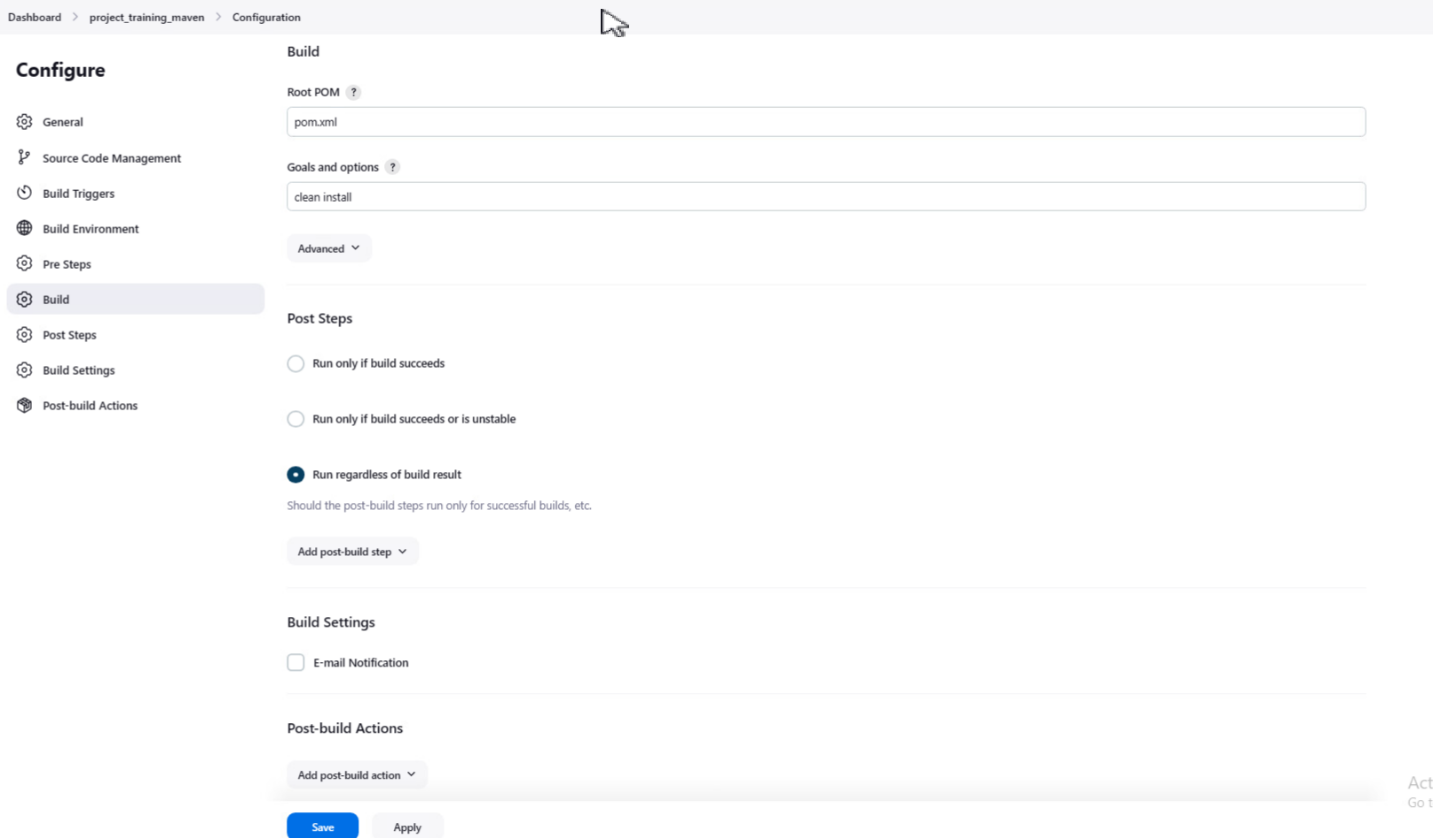
STEP-2: Git configuration: add the git hub repository link and no need to add the credentials in the branch field set it as “main”.



STEP-3: build triggers: 1st field “build whenever SNAPSHOT dependency is built” will be automatically ticked if we copy the project from docker project initially while creating the project.



STEP-4: Build: add pom.xml file and add clean install in goals and options.



**Contents of pom file**

**<?xml version="1.0" encoding="UTF-8"?>**

**<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<groupId>com.mycompany.app</groupId>**

**<artifactId>my-app</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<name>my-app</name>**

**<!-- FIXME change it to the project's website -->**

**<url>http://www.example.com</url>**

**<properties>**

**<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**

**<maven.compiler.source>1.7</maven.compiler.source>**

**<maven.compiler.target>1.7</maven.compiler.target>**

**</properties>**

**<dependencies>**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.11</version>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->**

**<plugins>**

**<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->**

**<plugin>**

**<artifactId>maven-clean-plugin</artifactId>**

**<version>3.1.0</version>**

**</plugin>**

**<!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->**

**<plugin>**

**<artifactId>maven-resources-plugin</artifactId>**

**<version>3.0.2</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-compiler-plugin</artifactId>**

**<version>3.8.0</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-surefire-plugin</artifactId>**

**<version>2.22.1</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-jar-plugin</artifactId>**

**<version>3.0.2</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-install-plugin</artifactId>**

**<version>2.5.2</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-deploy-plugin</artifactId>**

**<version>2.8.2</version>**

**</plugin>**

**<!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->**

**<plugin>**

**<artifactId>maven-site-plugin</artifactId>**

**<version>3.7.1</version>**

**</plugin>**

**<plugin>**

**<artifactId>maven-project-info-reports-plugin</artifactId>**

**<version>3.0.0</version>**

**</plugin>**

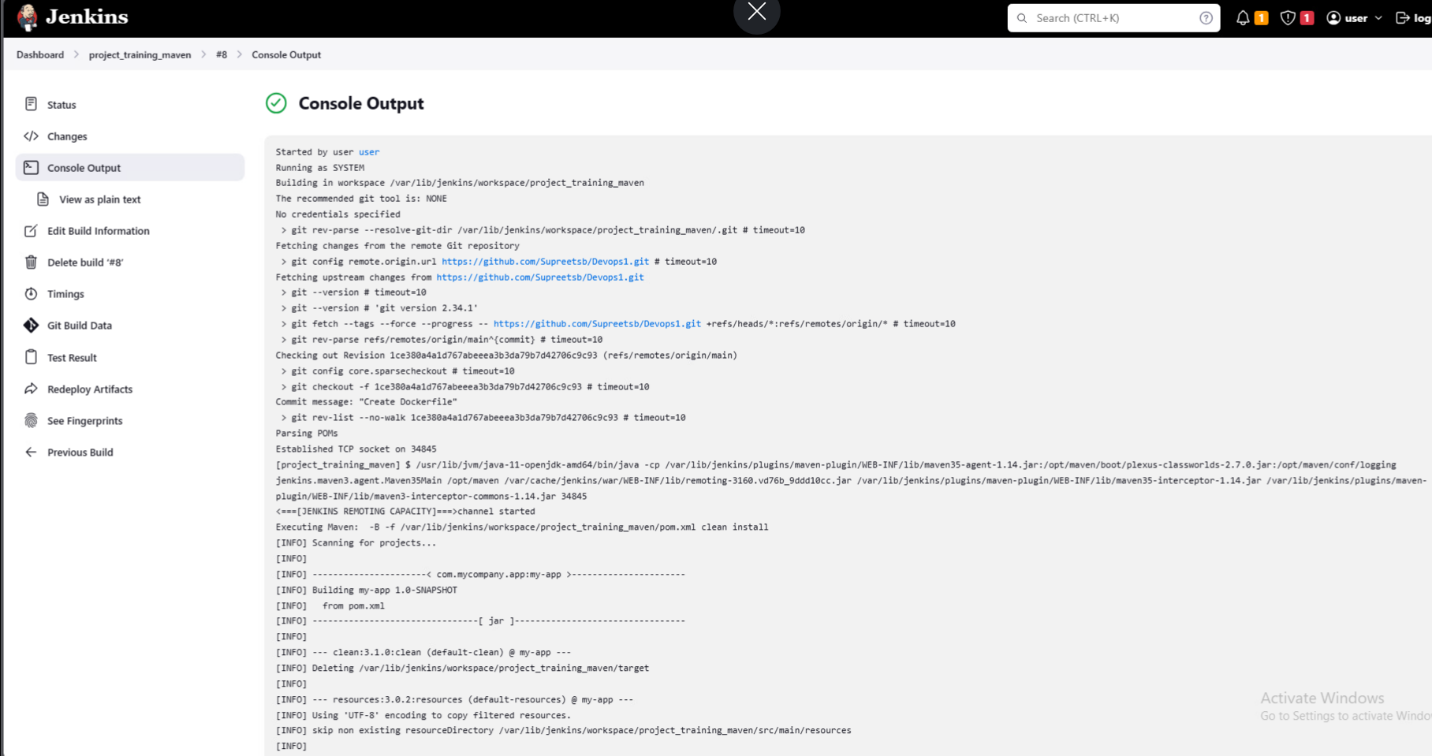
**</plugins>**

**</pluginManagement>**

**</build>**

**</project>**

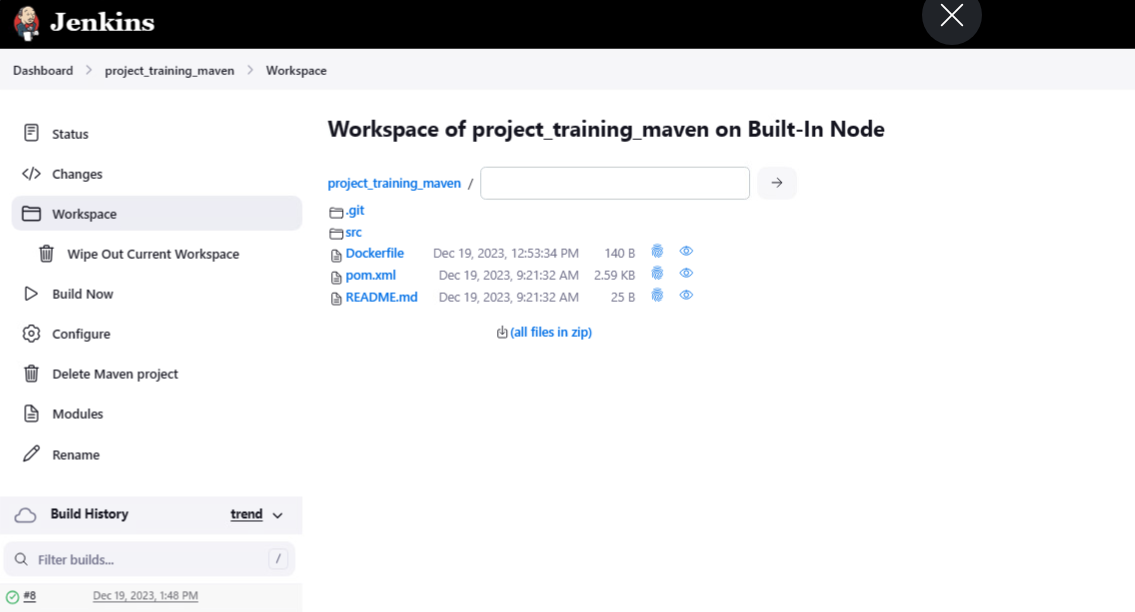
**STEP-5:** Check console output of the maven project



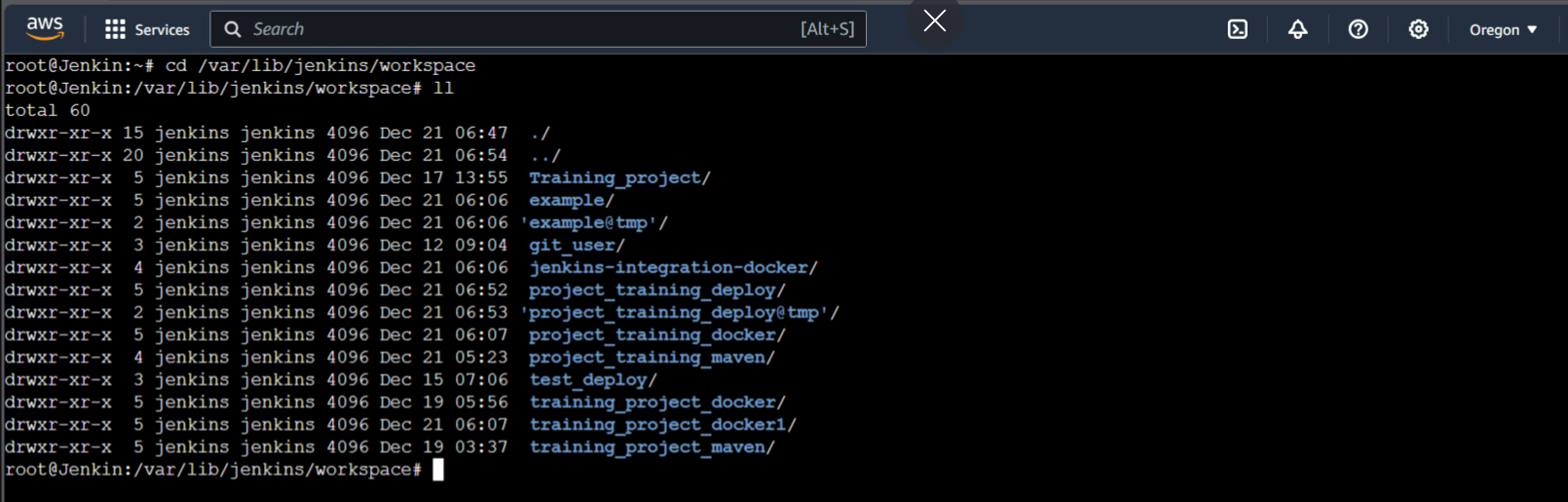
* Project showing output where jar file is created.



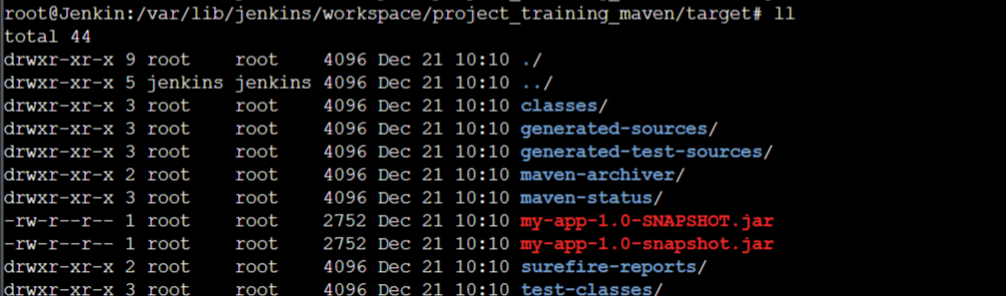
* Maven workspace: for reference



* Jenkins projects for reference



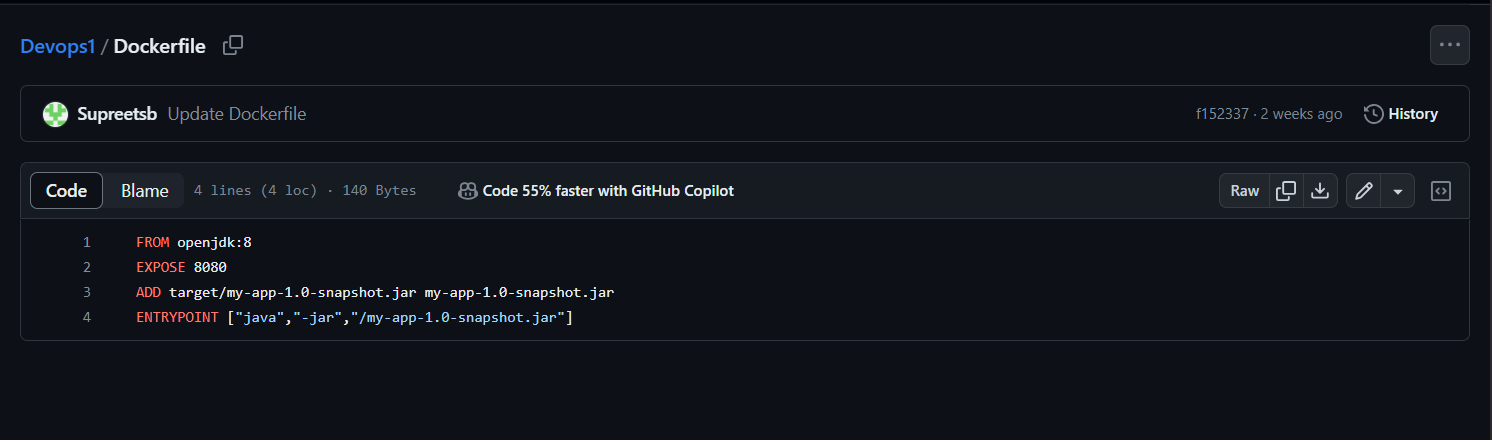
* Jar file location in the instance



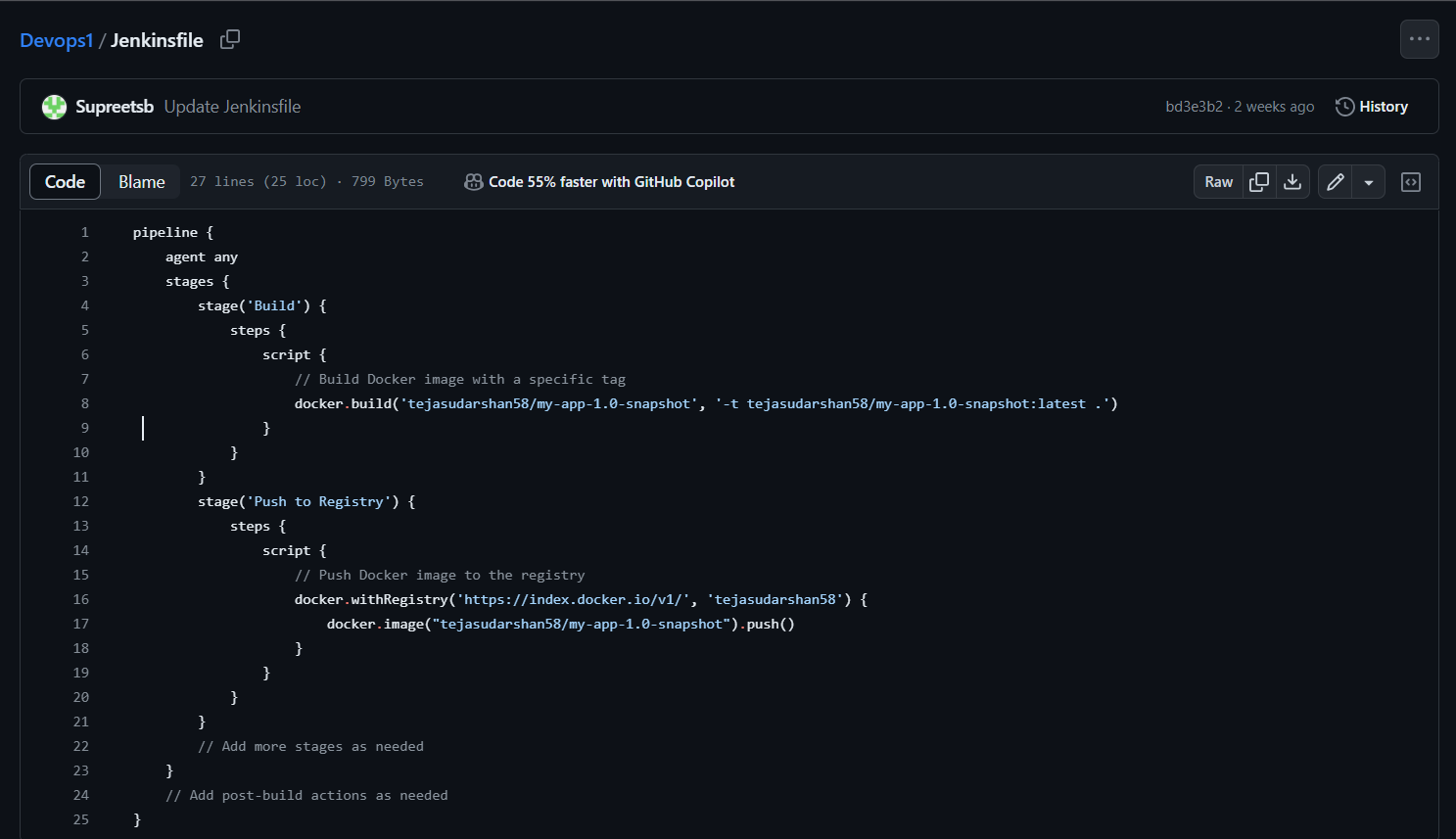
**Stage-4**

**Docker deployment**

Dockerfile



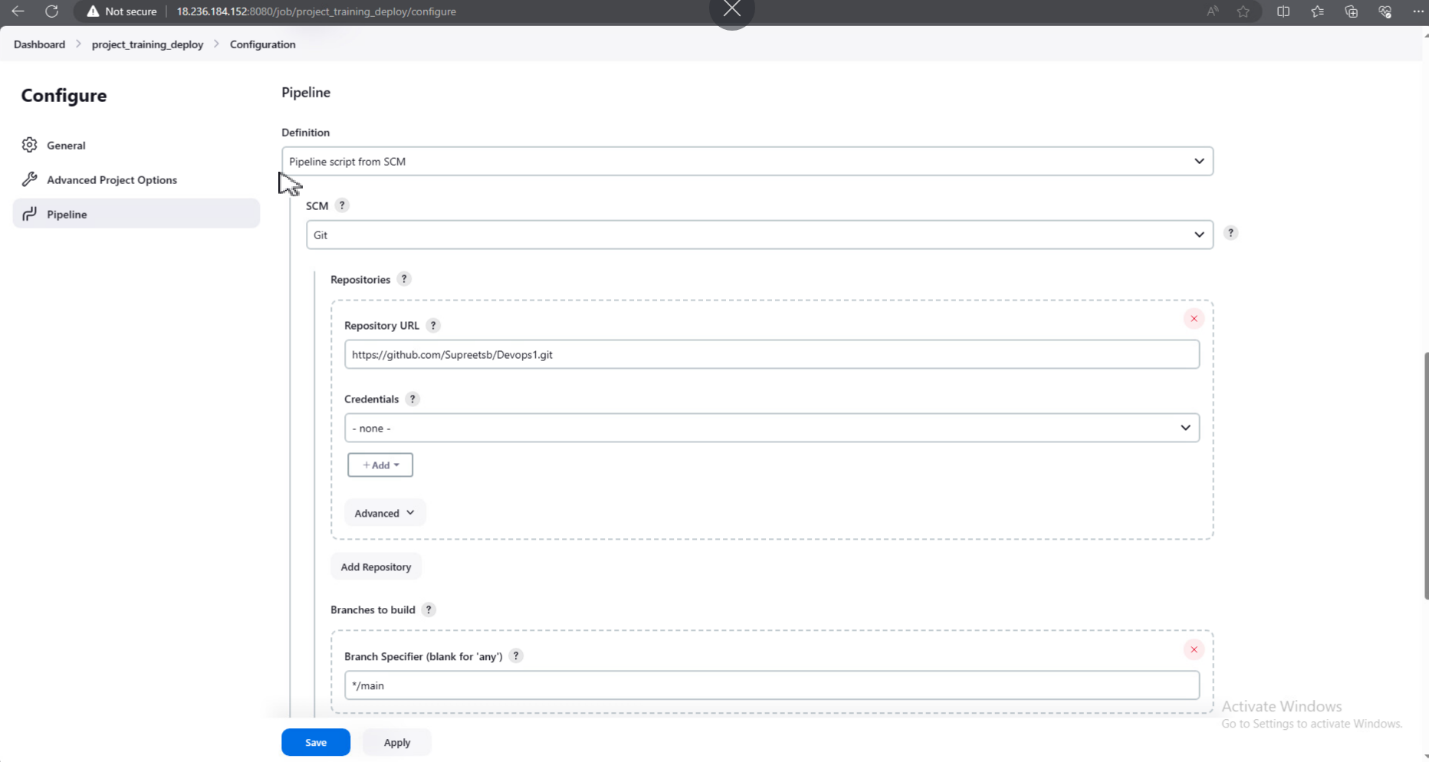
Jenkinfile



Configure pipeline:

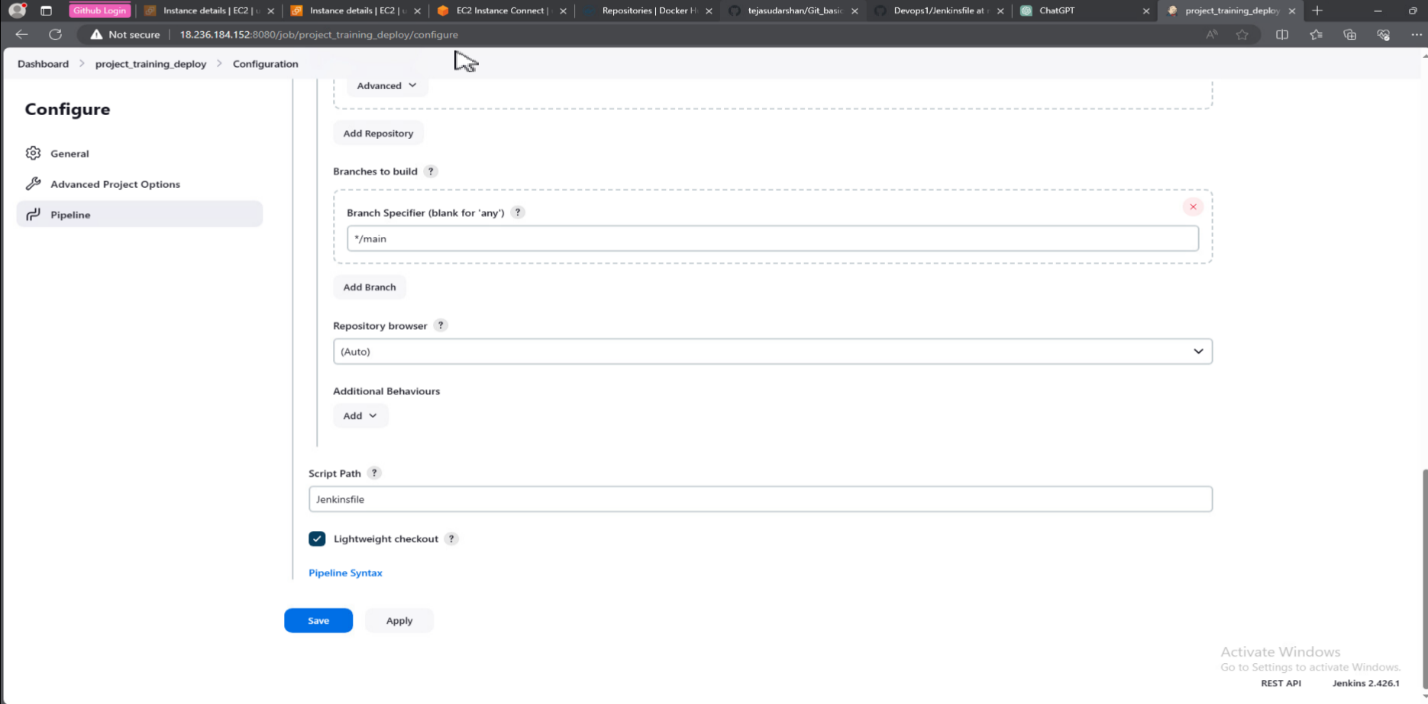
STEP-1: Select “SCM” – Jenkins will obtain pipeline from locally cloned git repository.

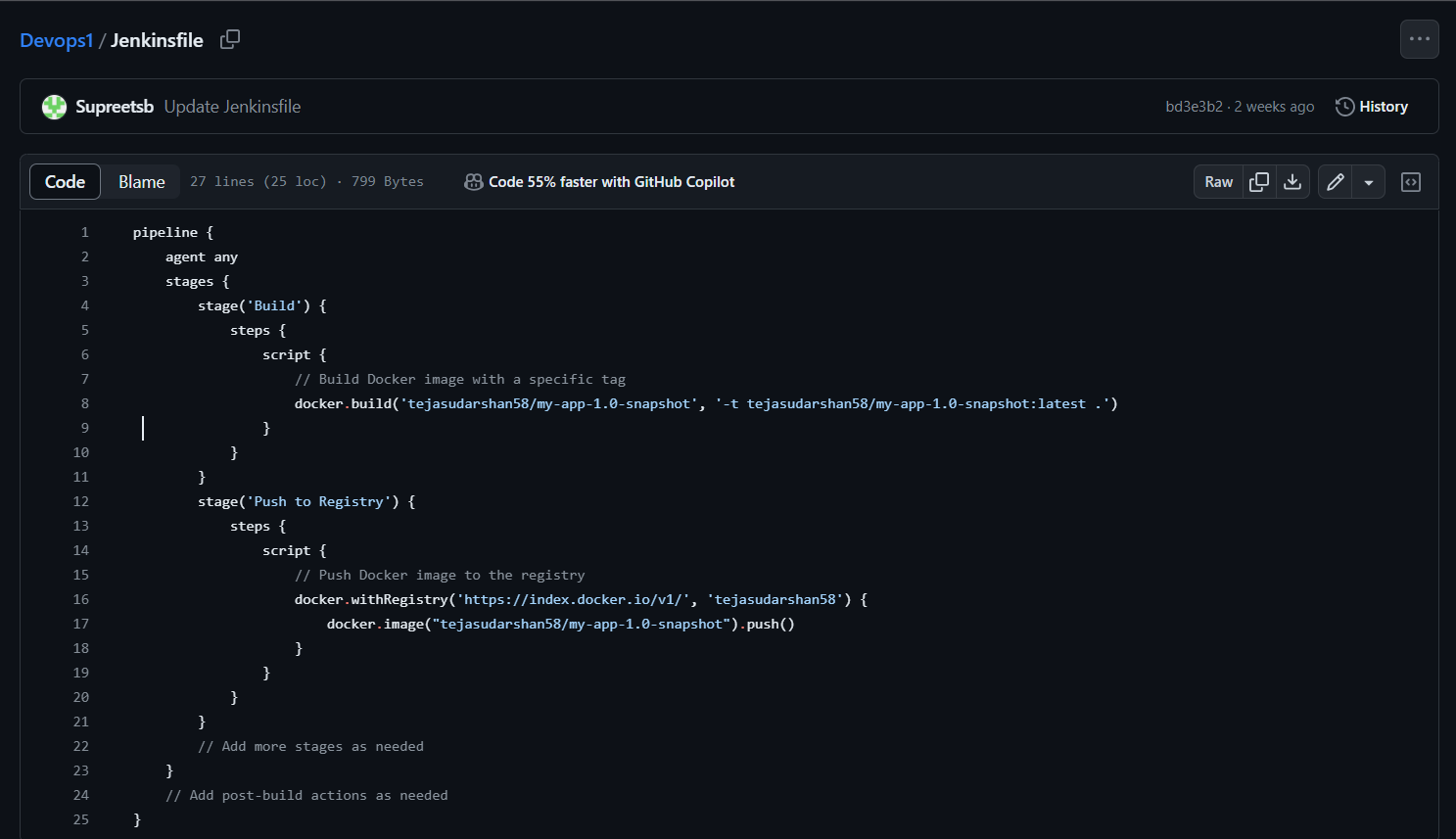
STEP-2: Add repository URL.



STEP-3: Specify branch “ \*/main ”

STEP-4: Apply and Save

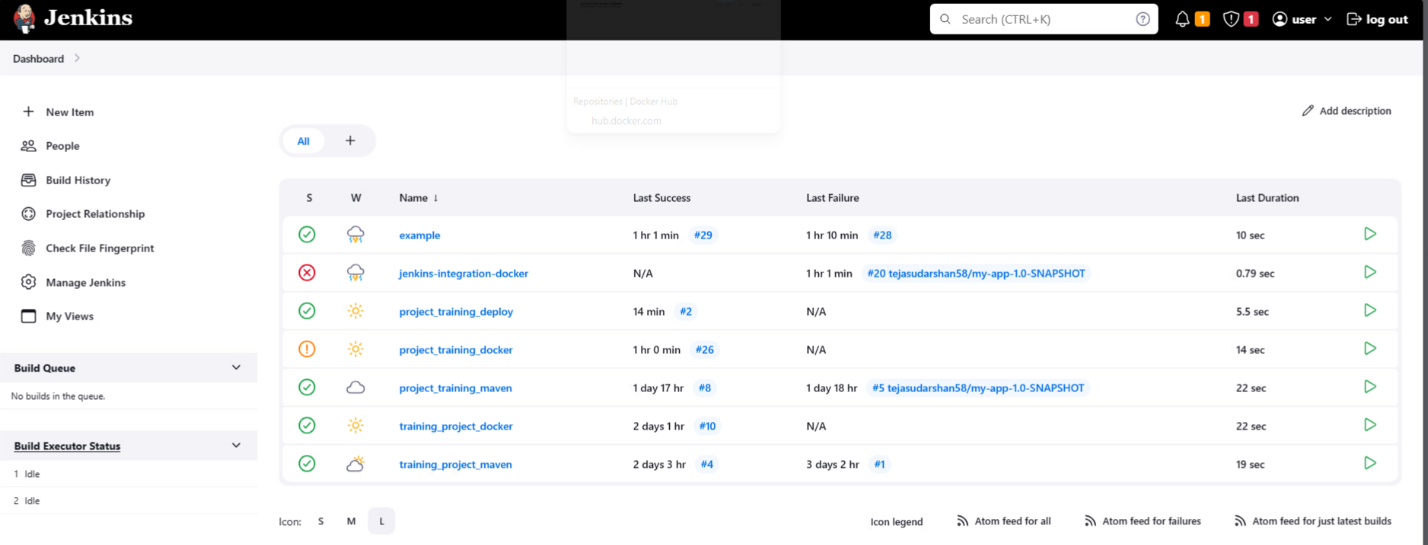




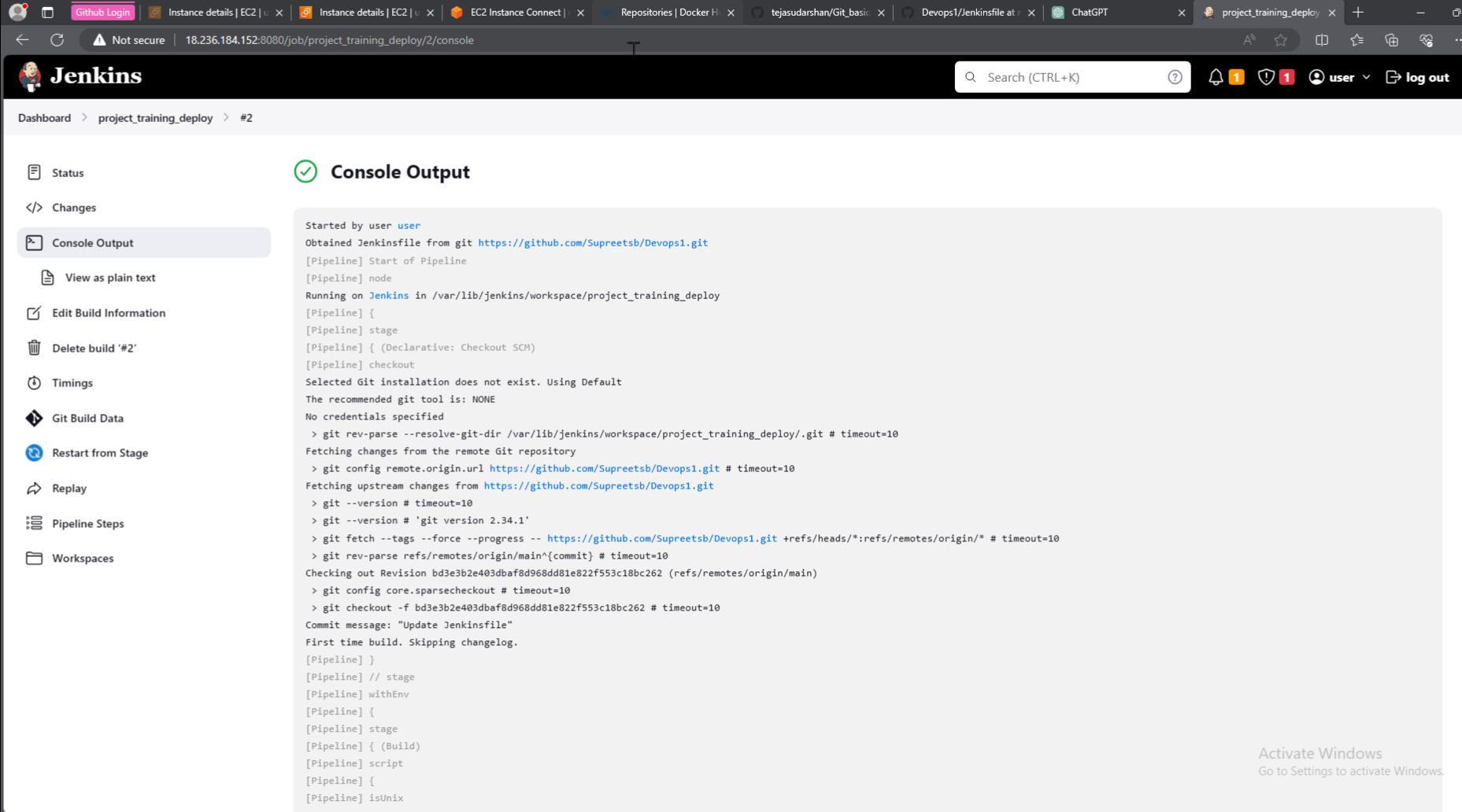
STEP-5: Pipeline for docker deployment is written in Jenkinsfile.

STEP-6: Select the item and click on build.

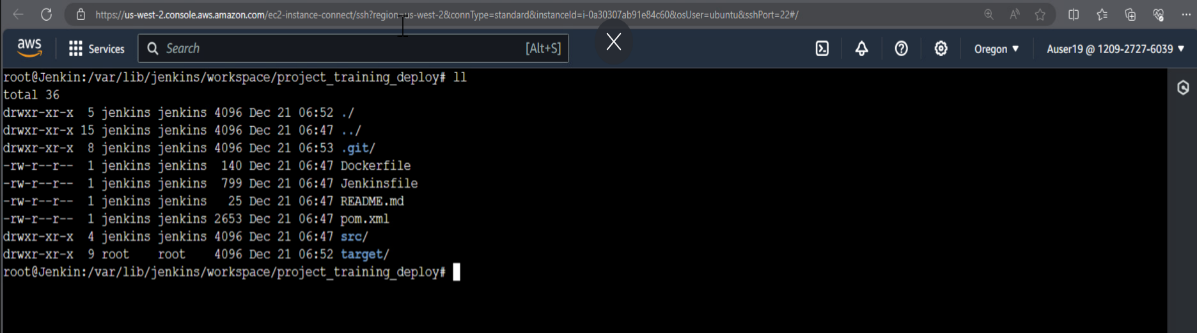
STEP-7: Item named “Project\_training\_deploy” is created.



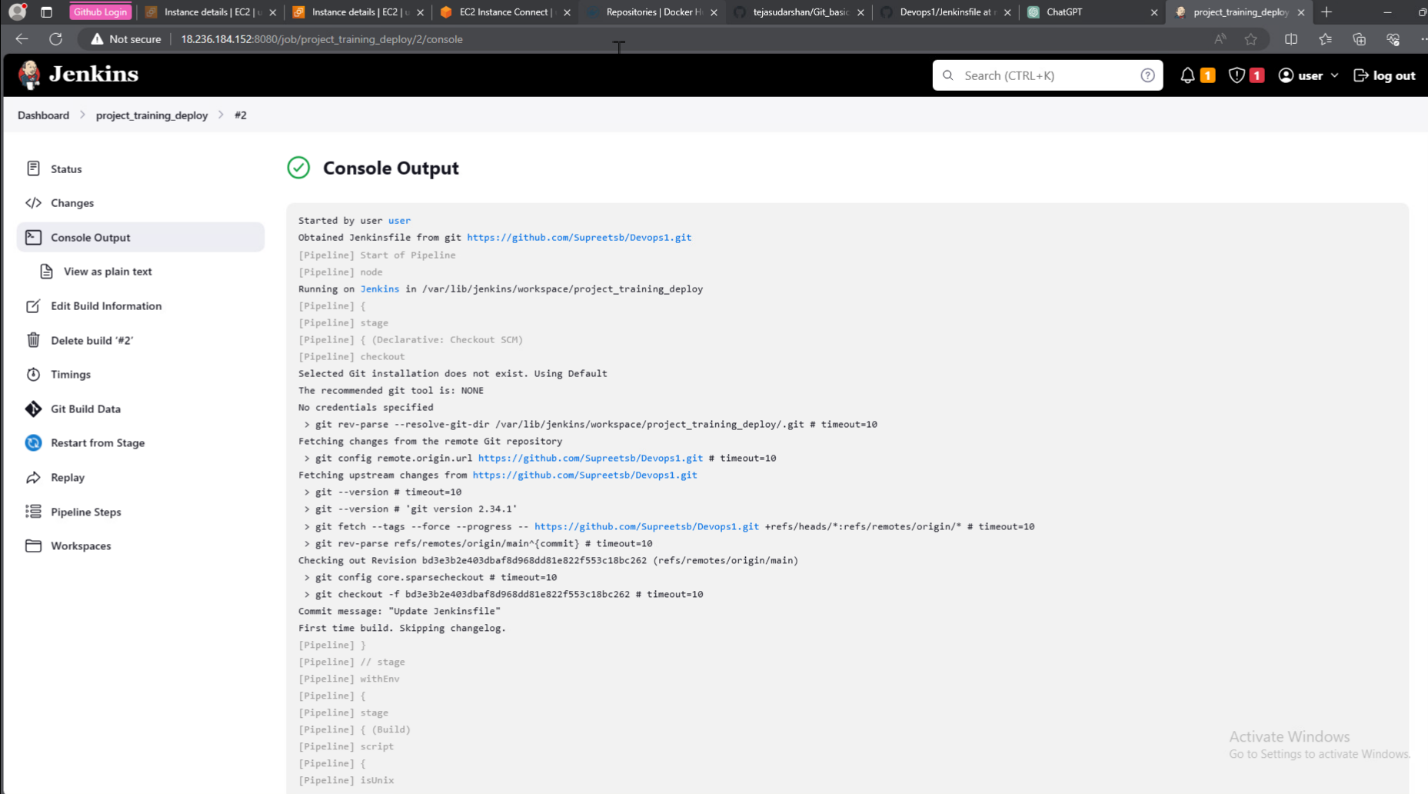
STEP-8: Check the console output for build details.



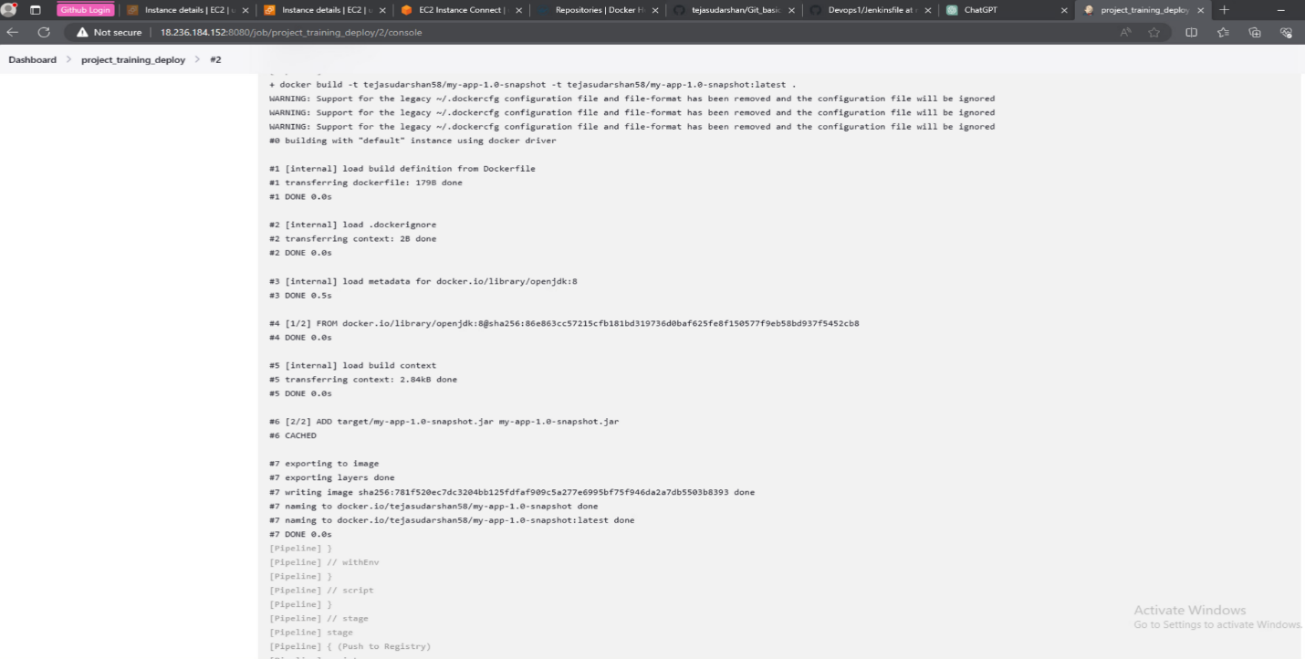
STEP-9: In Jenkins server we can find the folder “project\_training\_deploy” containing all the files.



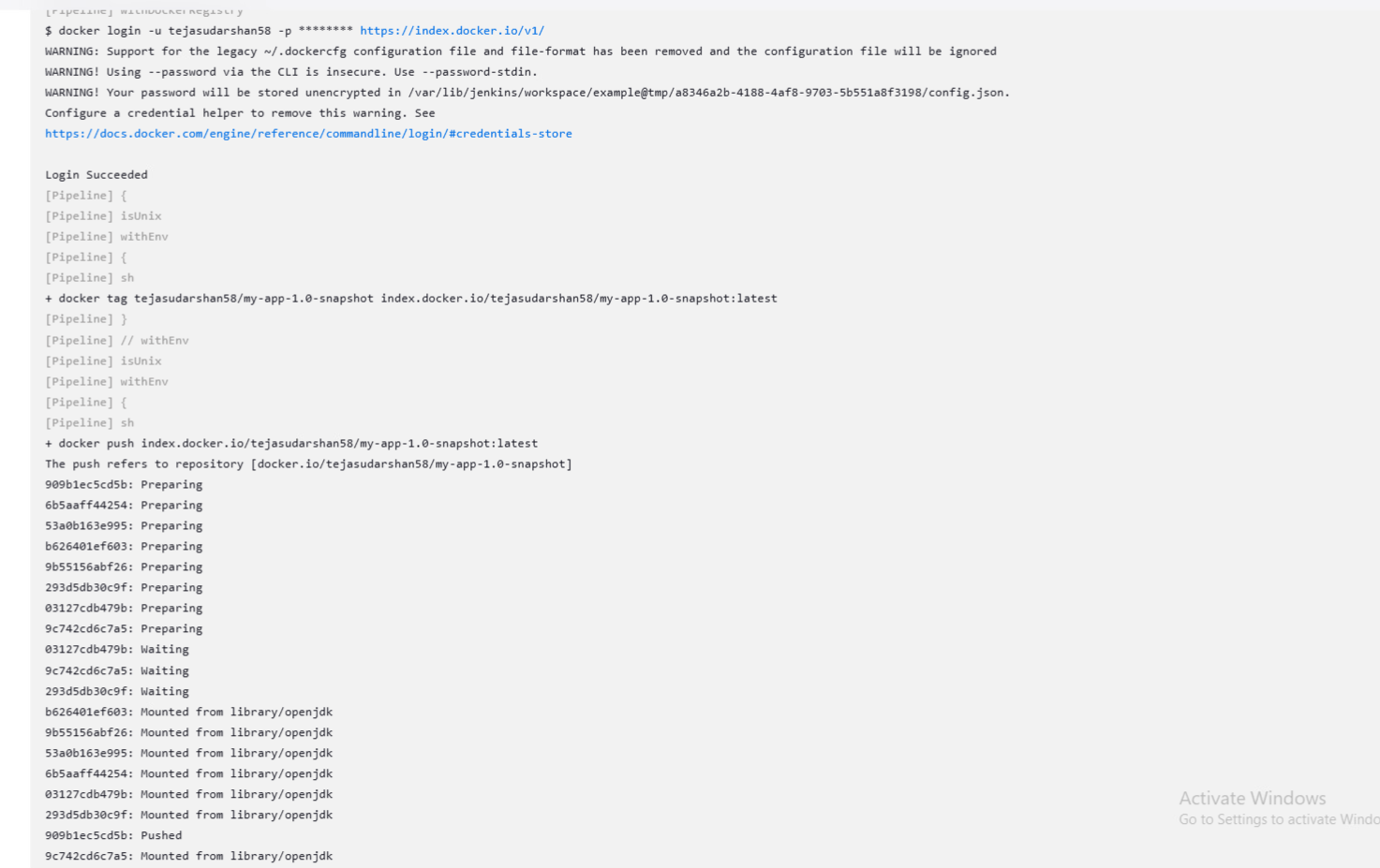
STEP-10: Check for docker file built and pushed on to the docker hub.



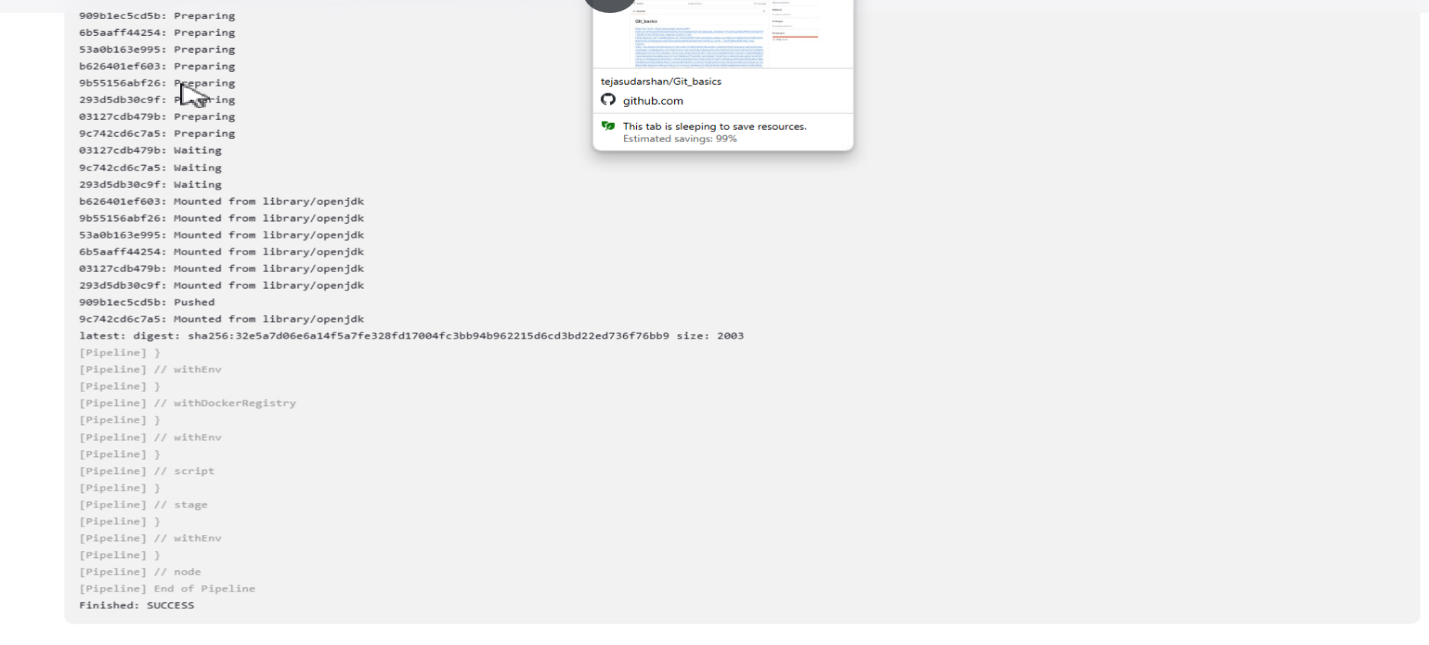
* Docker build command is executed which detects the dockerfile in the workspace and image name “my-app-1.0-snapshot: latest”.



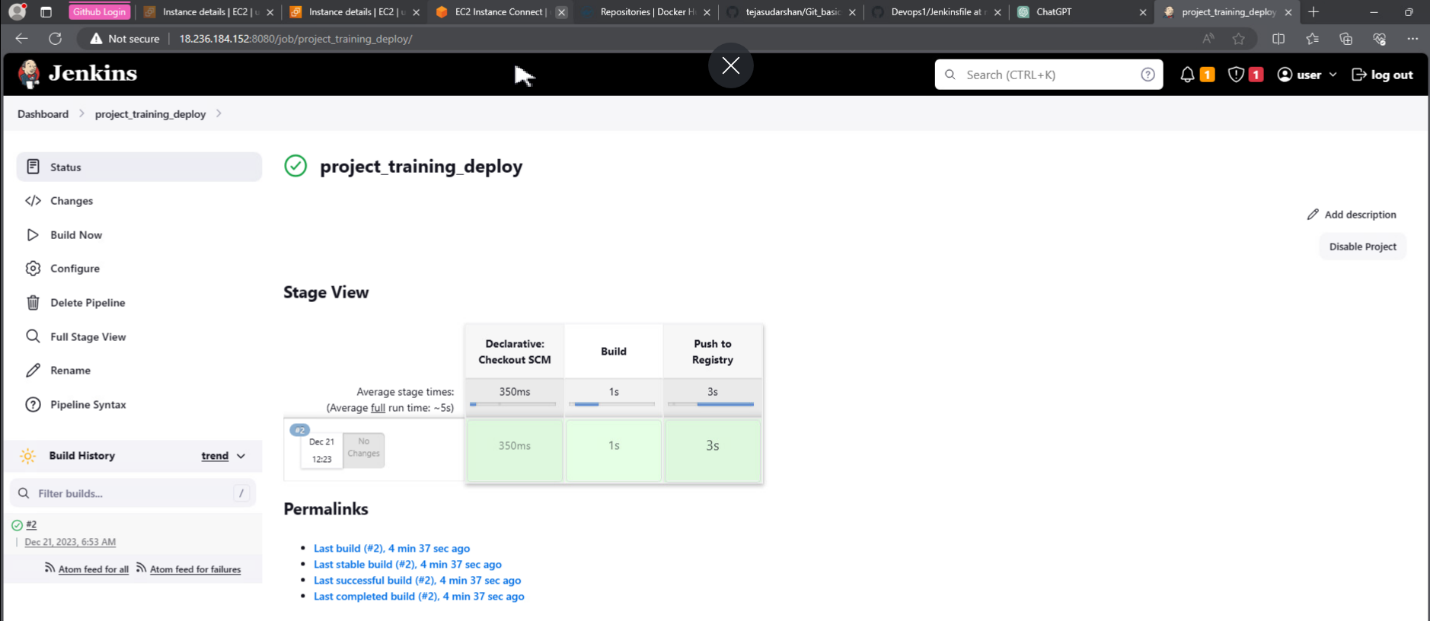
* Docker login command is executed to login to docker hub.
* Docker tag is used to tag the image that has to pushed i.e “docker registry id/my-app-1.0-snapshot:latest” .
* Docker push is executed to push the tagged image to the docker image to the docker hub.



* Below Image shows that docker image is pushed.
* Finished: SUCCESS shows that the build is complete.



* Build queue shows the build complete with green tick.
* Stage view of the process include Checkout SCM, Build, Push to registry.



* Docker image(my-app-1.0-snapshot) can be seen in Repositories section in the docker hub.

