# **Experiment-6: Maven Automation Using Jenkins**

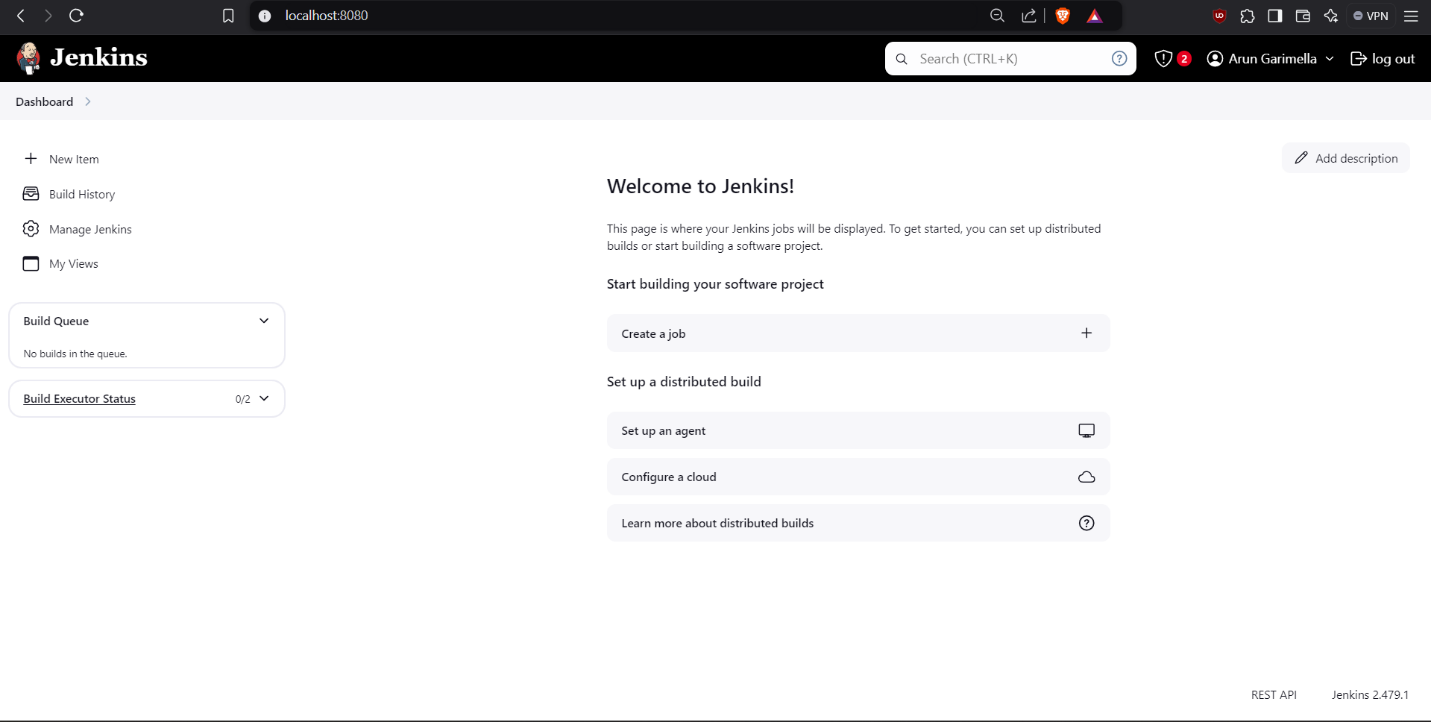
**Aim:** To build CI/CD pipelines using Jenkins for Maven Java and web projects, including both freestyle and scripted pipelines with Poll SCM.

**Introduction:**

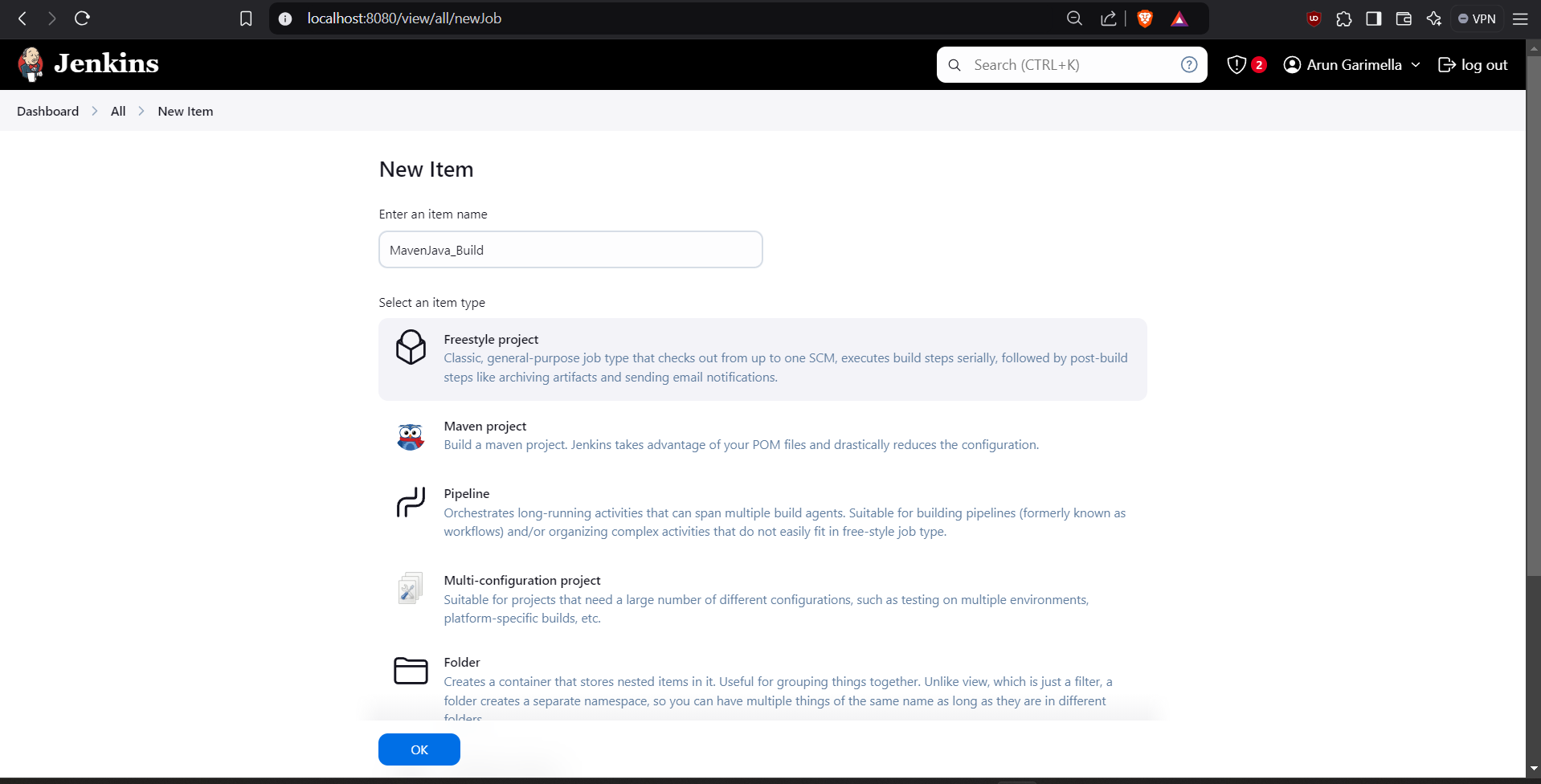
1. **Building the CI/CD Freestyle Pipeline Using Jenkins for Maven Java Project:**
   * **CI/CD:** Continuous Integration (CI) and Continuous Delivery (CD) are practices that involve automating the integration and deployment of code changes to improve software quality and delivery speed.
   * **Freestyle Pipeline:** A freestyle pipeline in Jenkins is a flexible and easy-to-configure pipeline that allows you to build, test, and deploy applications.
   * **Maven Java Project:** A Maven Java project uses Maven for managing dependencies and building Java applications.
2. **Building the CI/CD Freestyle Pipeline Using Jenkins for Maven Web Project with Poll SCM:**
   * **Maven Web Project:** A web project built using Maven, typically involving Java servlets, JSP, and other web technologies.
   * **Poll SCM:** Poll SCM is a Jenkins feature that periodically checks the source code repository for changes and triggers a build if changes are detected.
   * **Freestyle Pipeline for Web Project:** Configuring a freestyle pipeline to automate the build and deployment of a Maven web project, with the additional step of polling the SCM for changes.
3. **Building the CI/CD Scripted Pipeline Using Jenkins for Maven Java Project with Poll SCM:**
   * **Scripted Pipeline:** A scripted pipeline in Jenkins uses Groovy-based scripts to define the stages and steps of the CI/CD process, offering more control and flexibility compared to freestyle pipelines.
   * **Poll SCM:** As with the freestyle pipeline, Poll SCM is used to trigger builds based on changes in the source code repository.
   * **Scripted Pipeline for Java Project:** Creating a scripted pipeline for a Maven Java project, integrating Poll SCM to ensure the pipeline runs automatically when changes are detected.

**Procedure:**

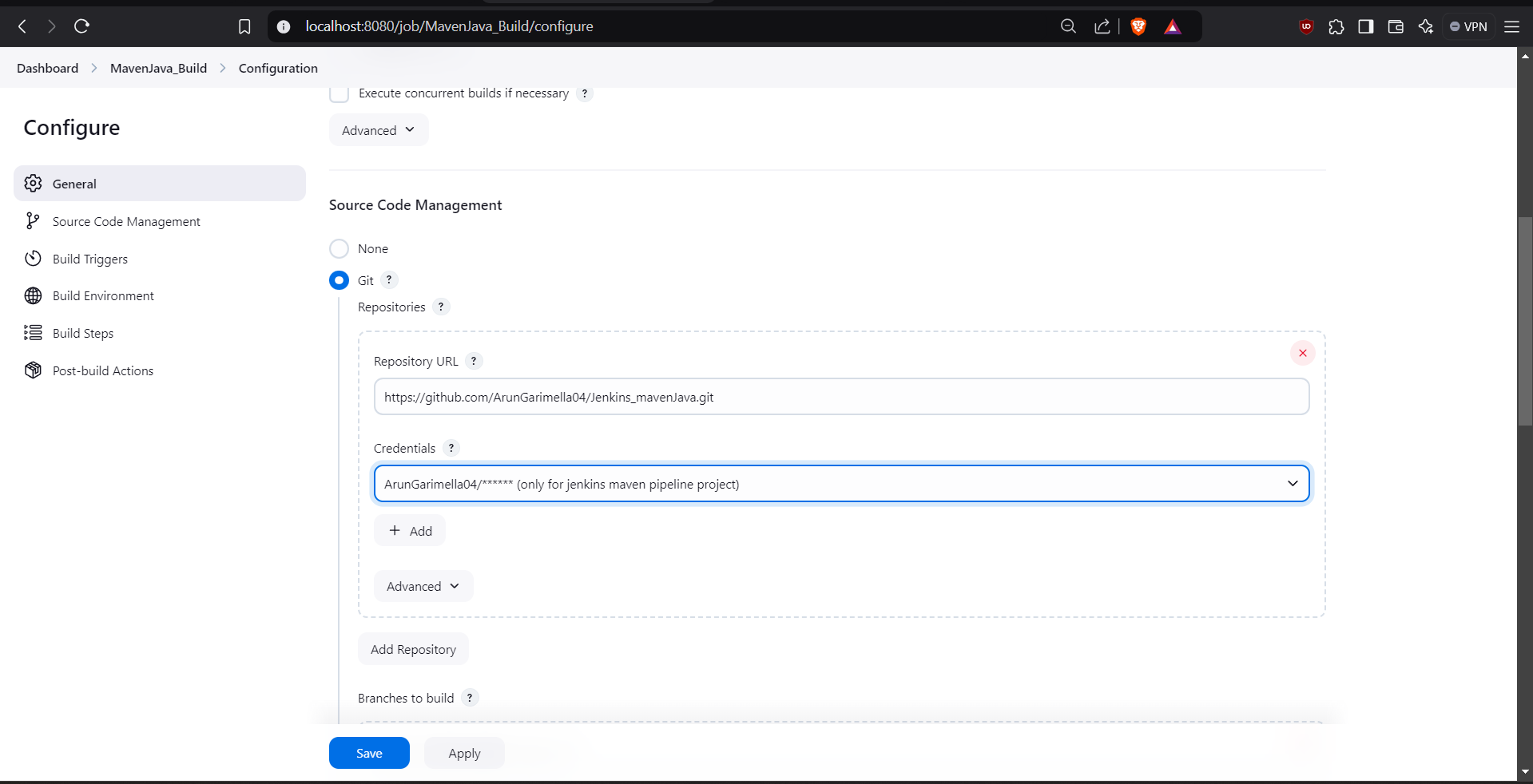
**Steps for Maven Java Automation:**

Step 1: Open Jenkins in local host: 8080/ and click on a new item present in the left side.

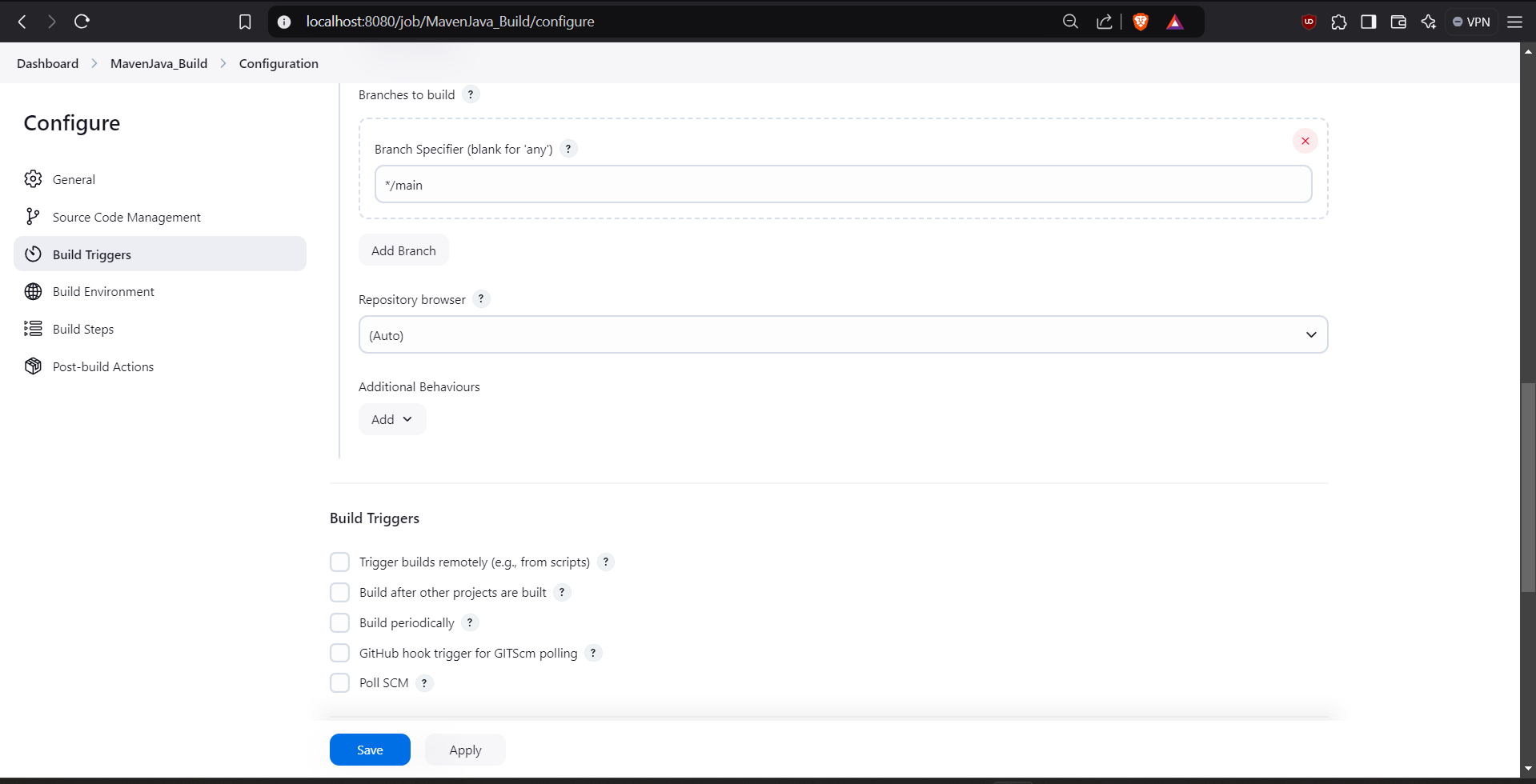
Step 2: Select a new Freestyle Project give name (eg. MavenJava\_Build) and then click ok.



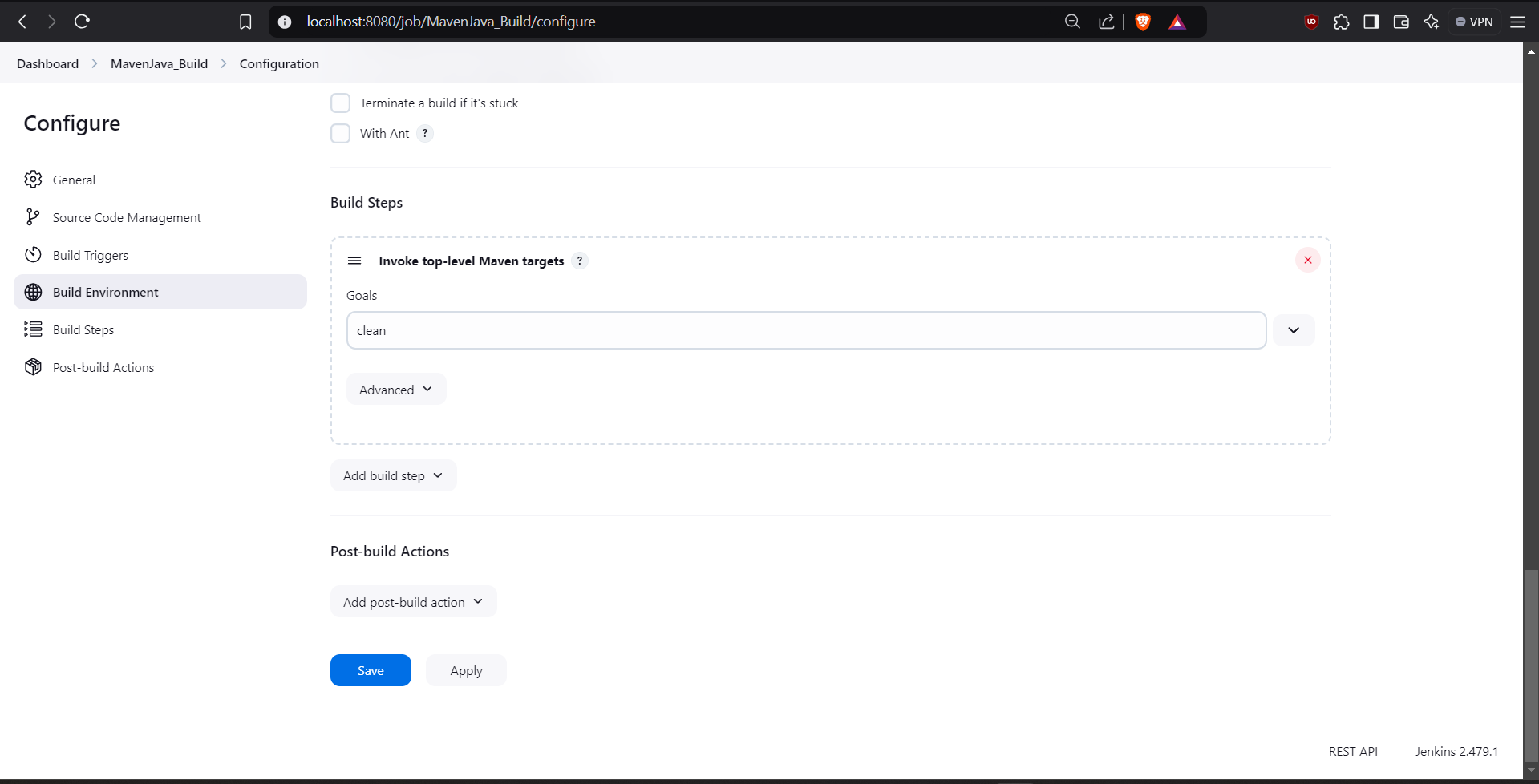
Step 3: In description type e.g., Java Build demo, Scroll down and in Source code management give the Git MavenJava repository URL of the project to be built.

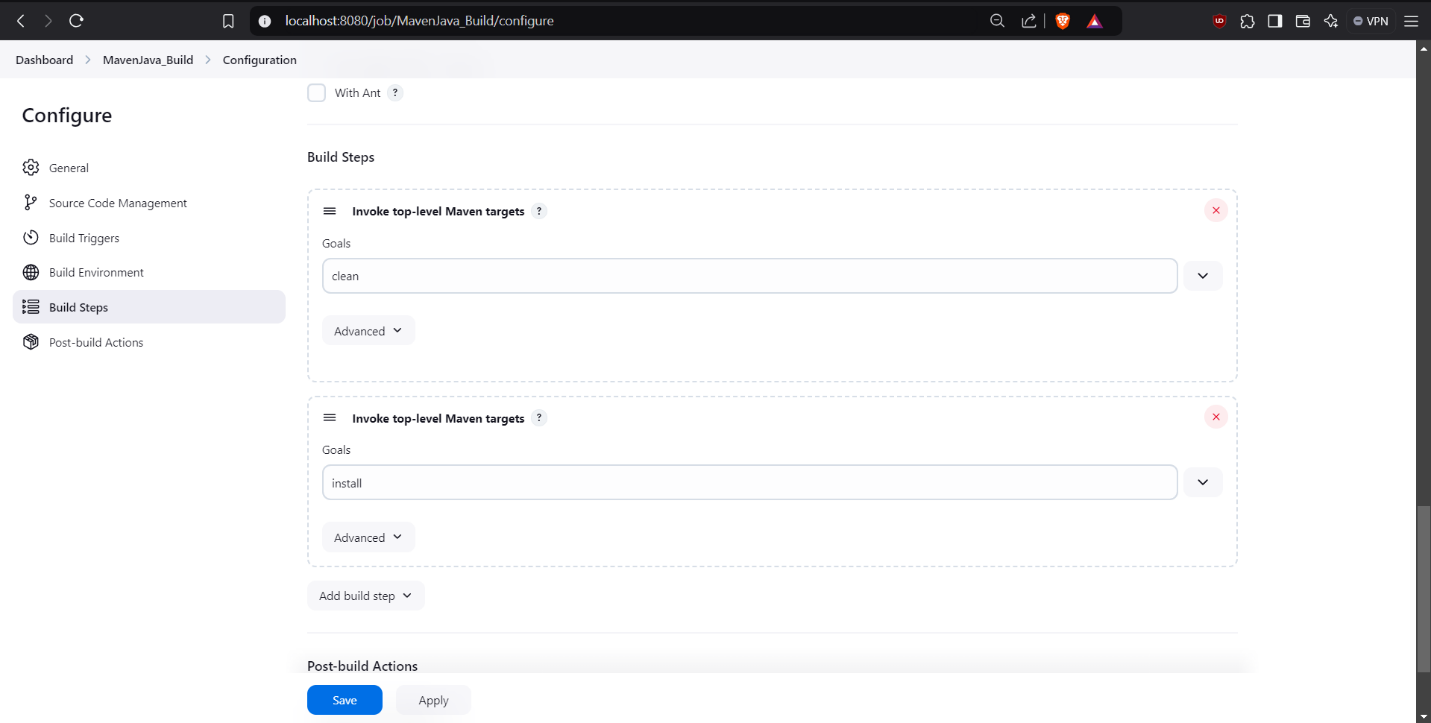


Step 4: Scroll down in Branches to build, Specify the Branch as \*/Main as it is in the GitHub.

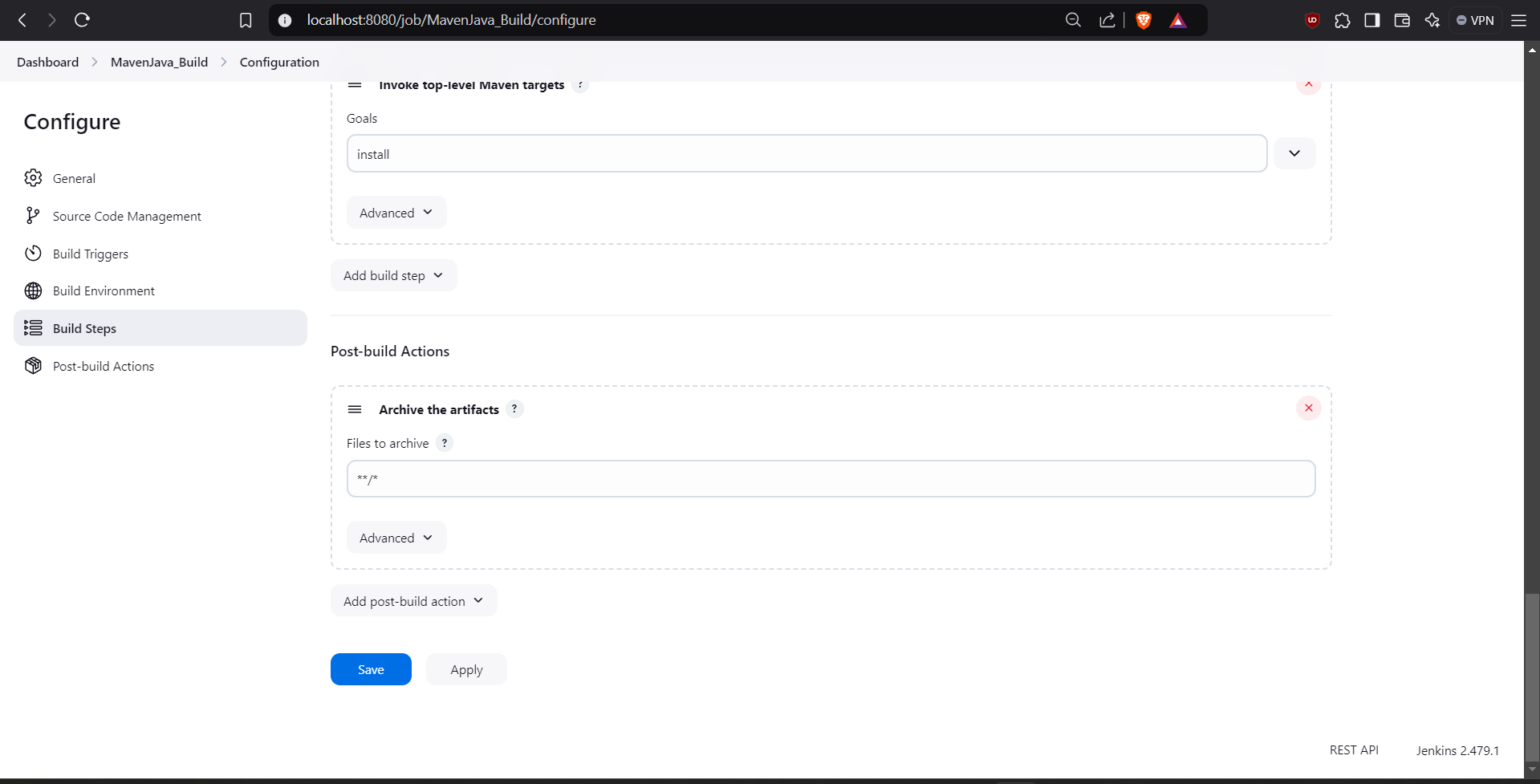


Step 5: Goto Build Steps -> Add Build Step-> Invoke top-level Maven targets, Type MAVEN\_HOME in Maven version, Goals: Clean.

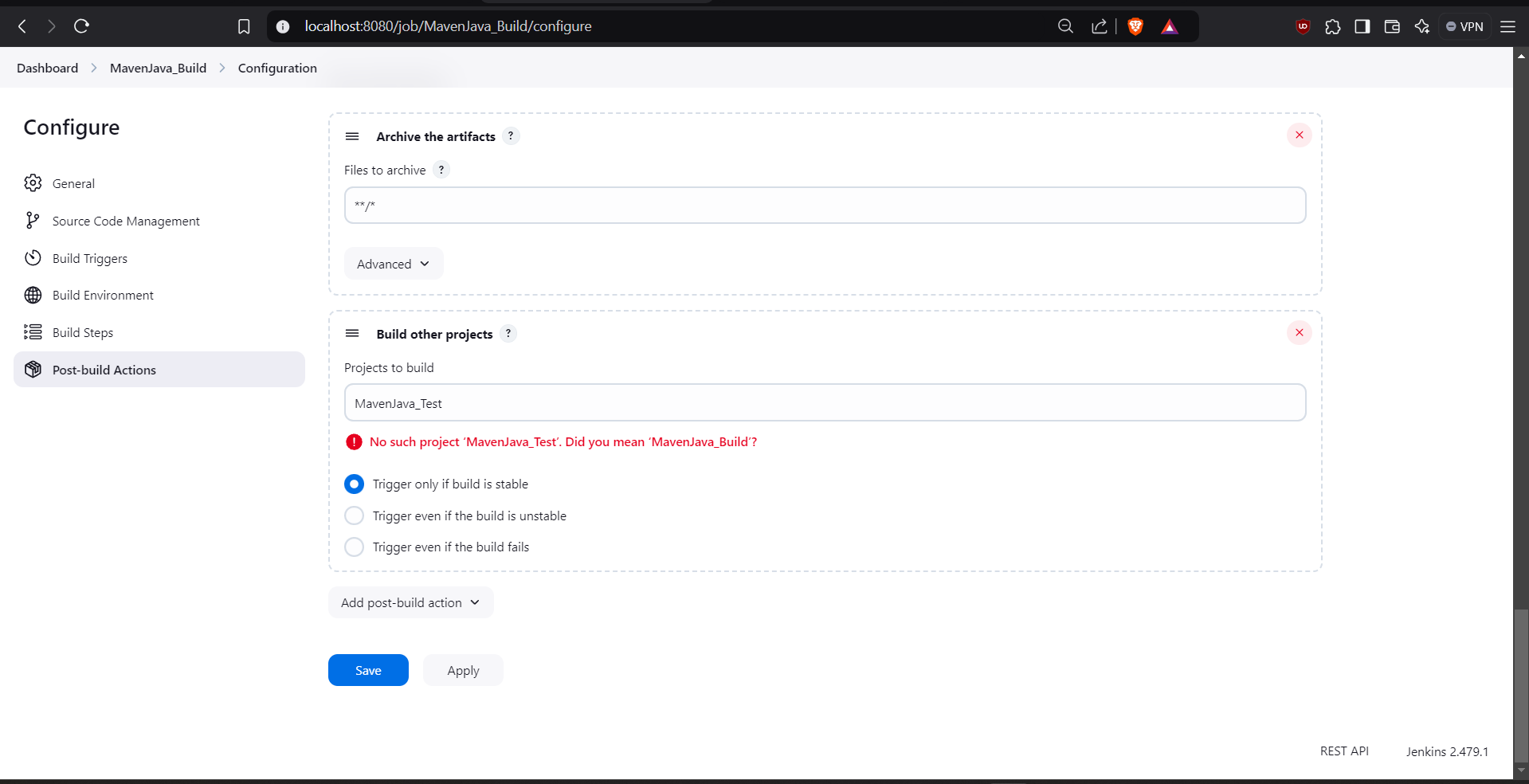


Step 6: Again, click Add Build Step-> Invoke top-level Maven targets-> Type MAVEN\_HOME in Maven version, Goals: install.

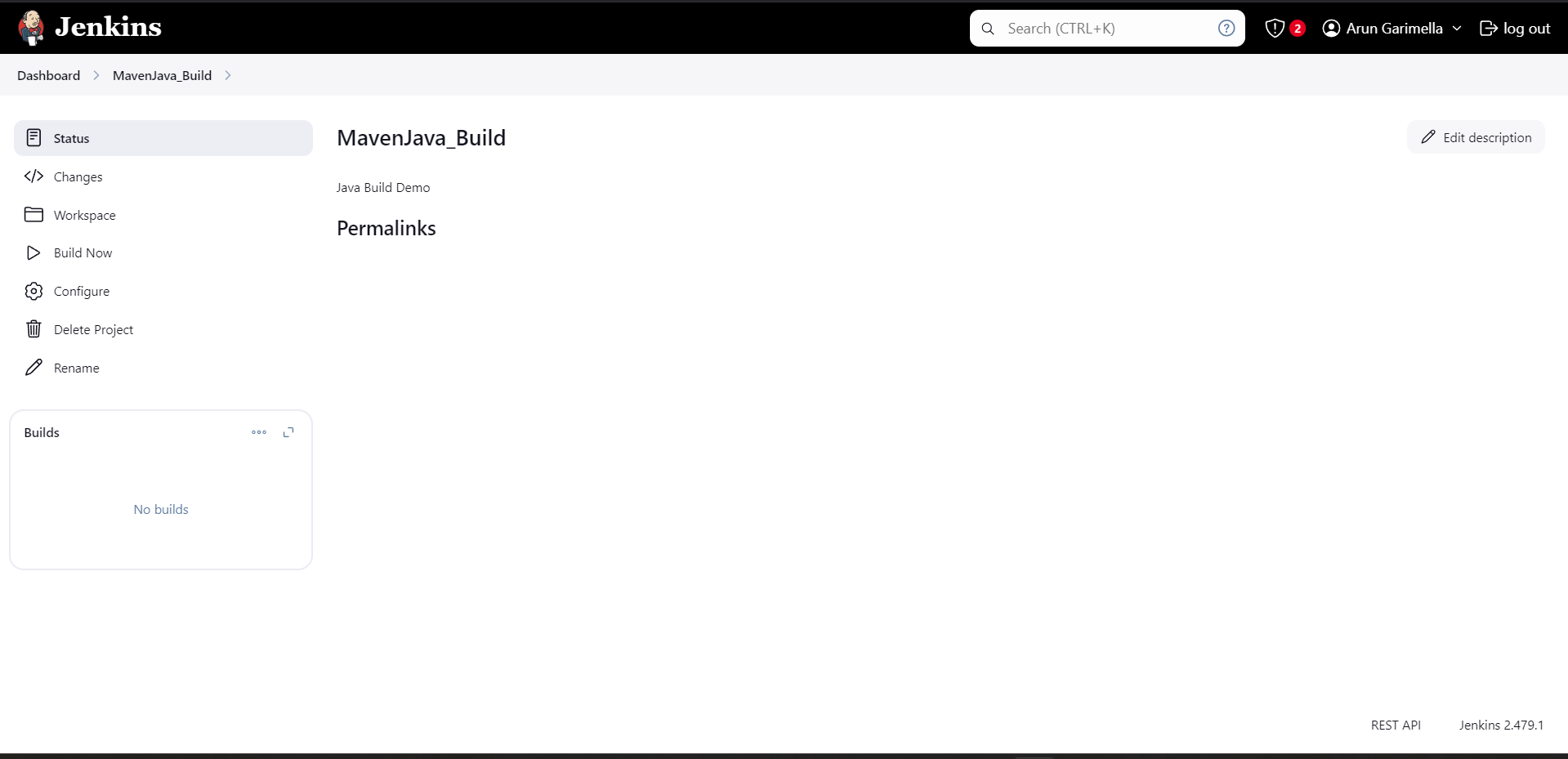
Step 7: Click add post build actions-> select Archive the artifacts, Type \*\*/\* in Files to Archive.



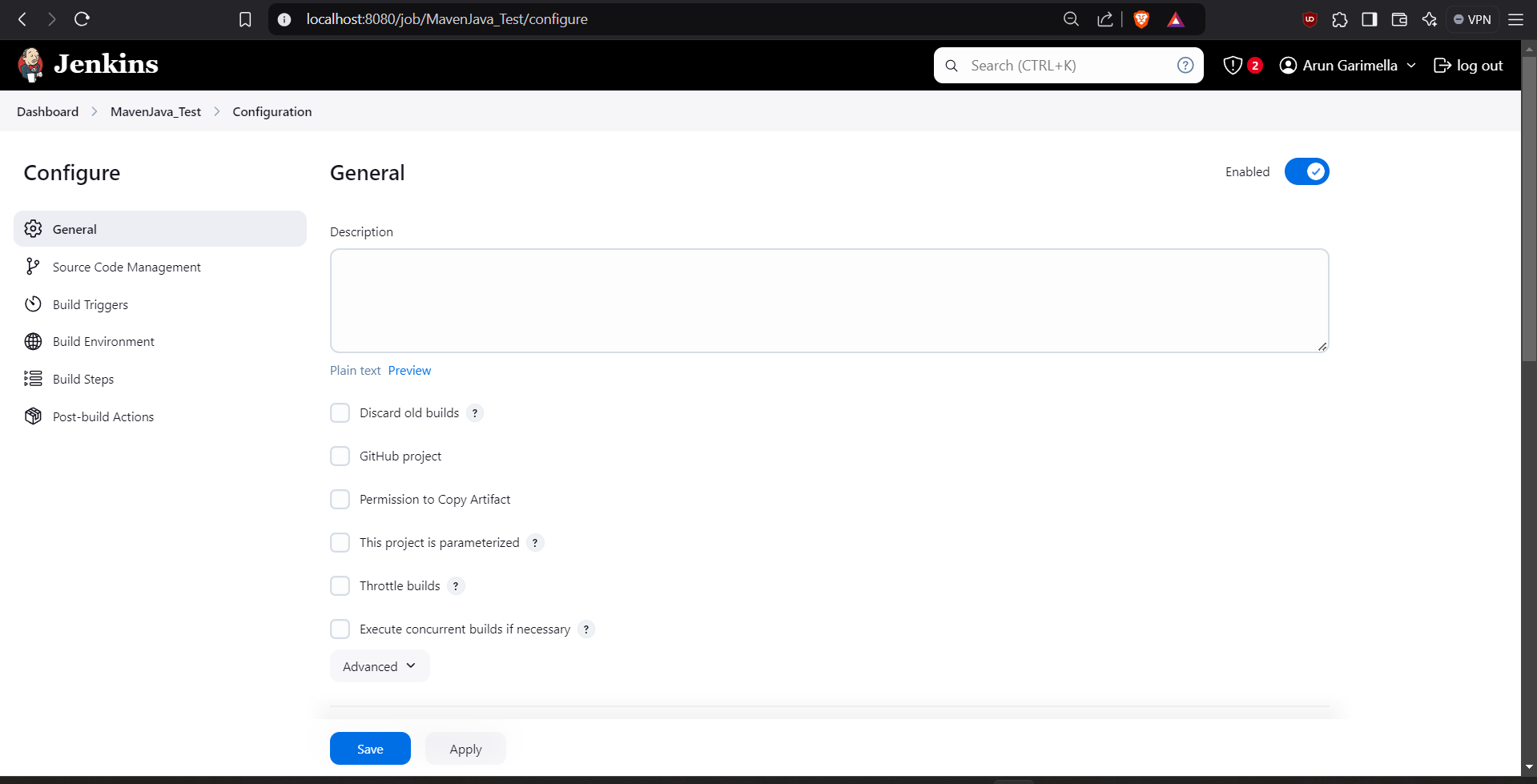
Step 8: Now click Add Post Build Action-> Build other projects**.**  In Projects to build, write MavenJava\_Test and select Trigger only if build is stable.



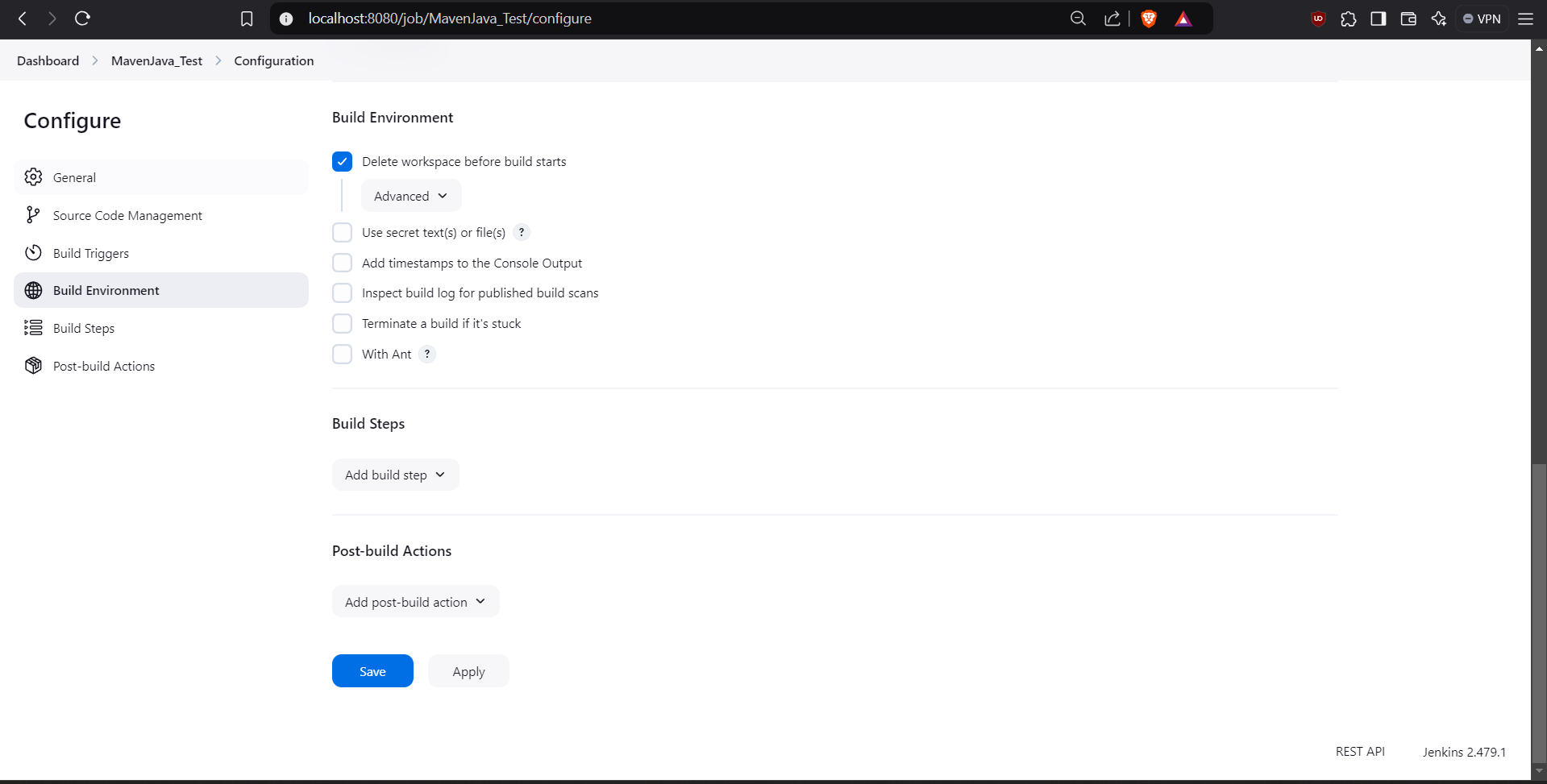
Step 9: Click on Apply and save.



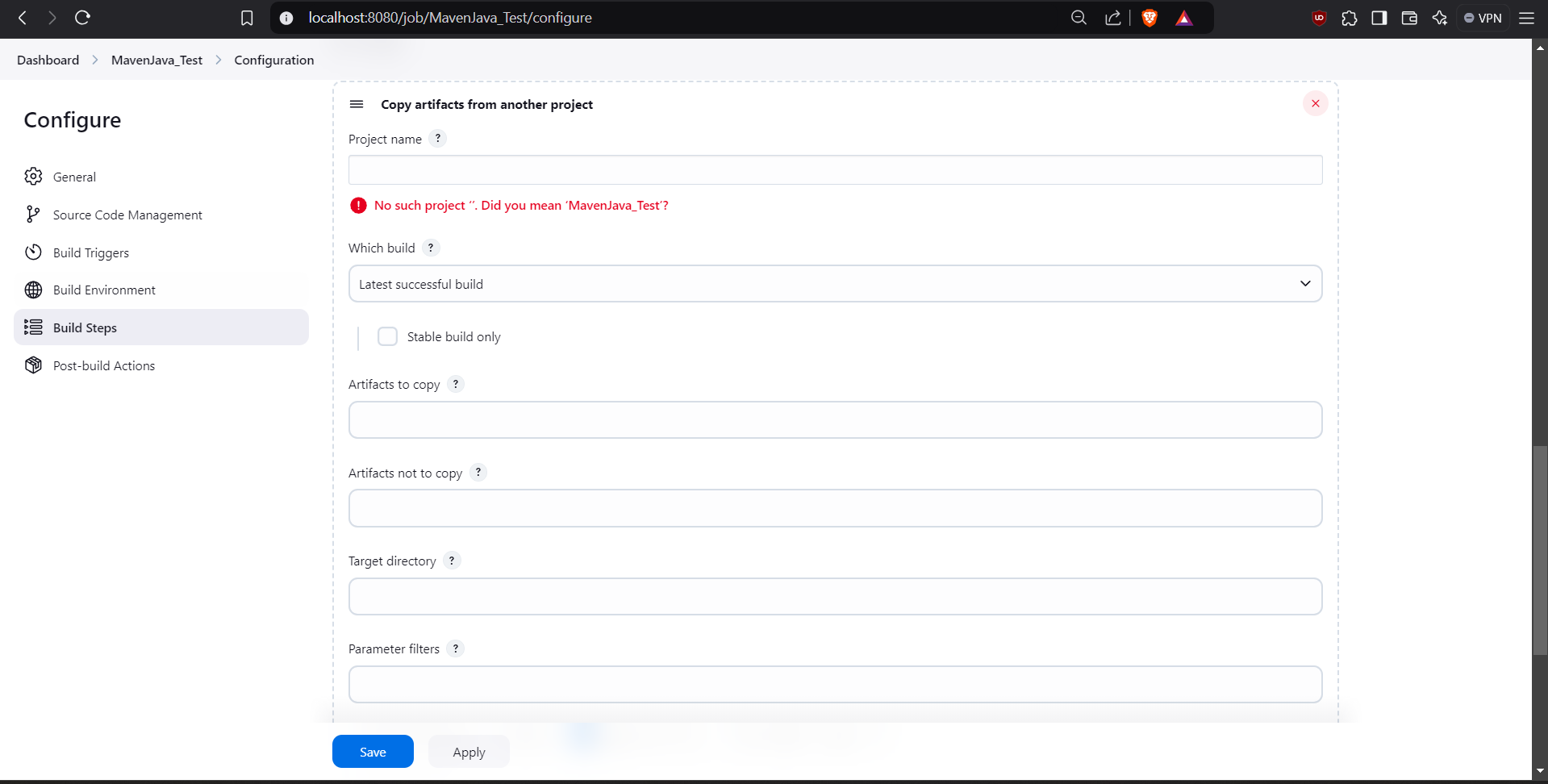
Step 10: Go to dashboard -> New item-> Freestyle Project, give project name as **MavenJava\_Test**, then press on OK.



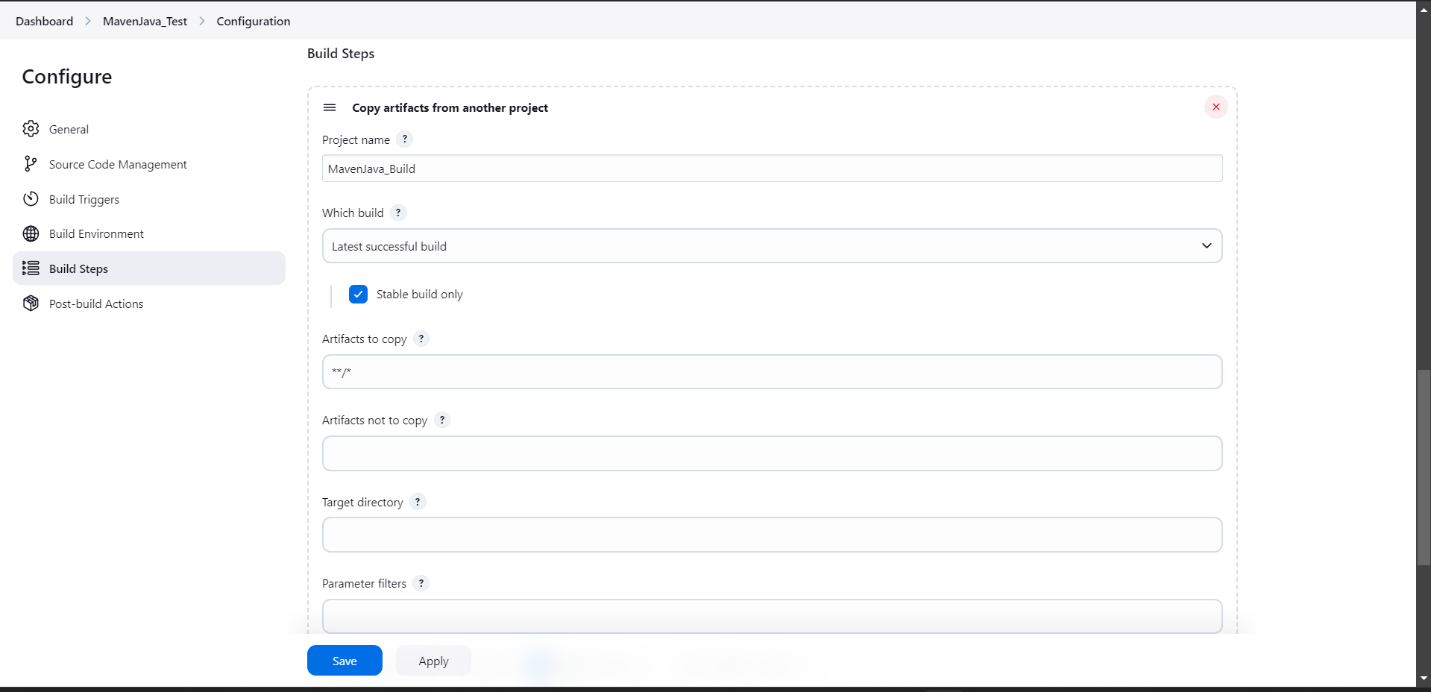
Step 11: In description type Test demo and in Build environment, check the box with name “Delete the workspace before build starts”.

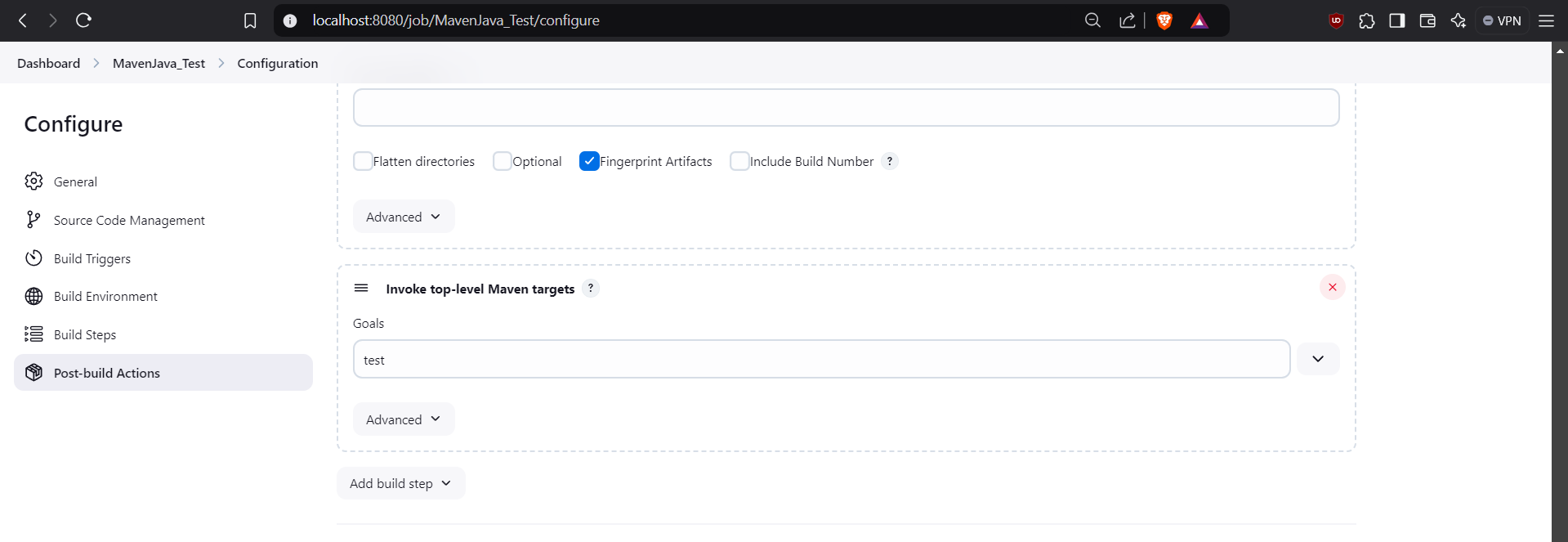


Step 12: Click on Add Build Step-> copy the artifacts from another project.

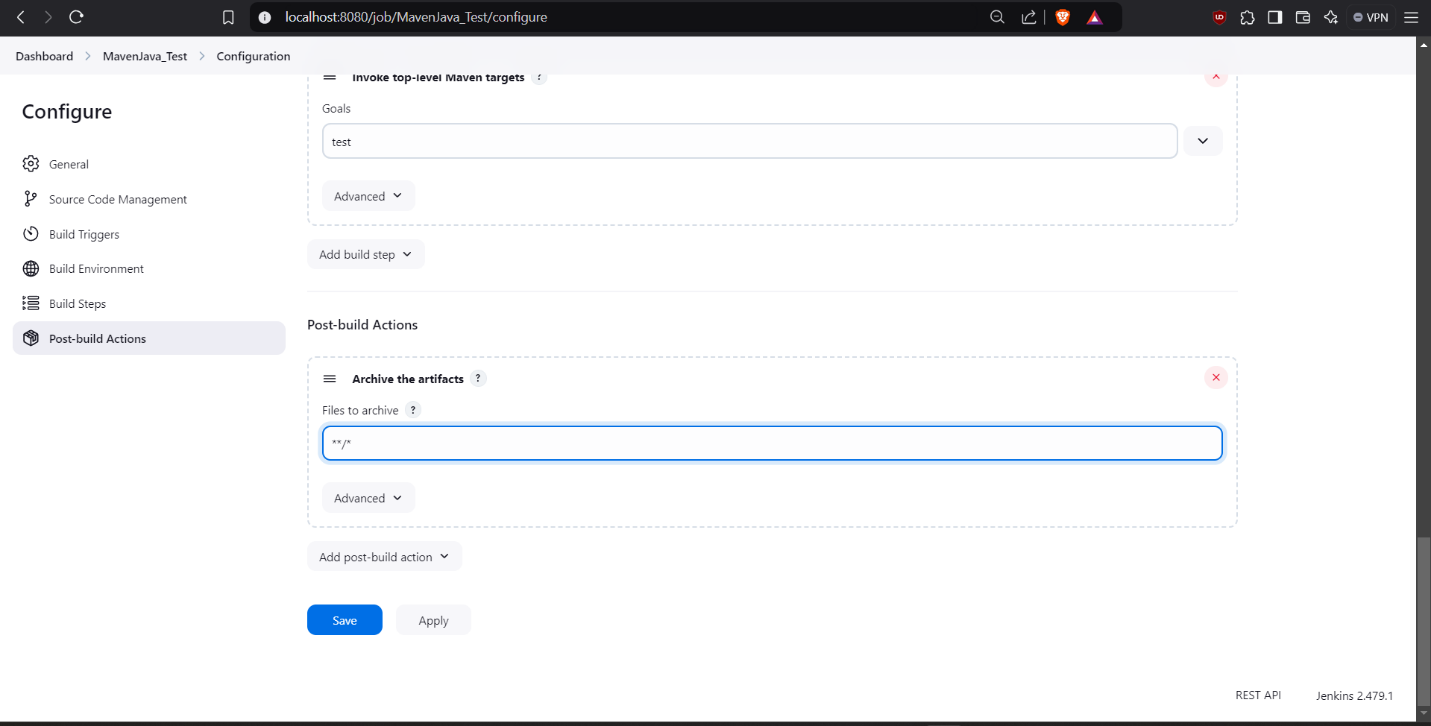


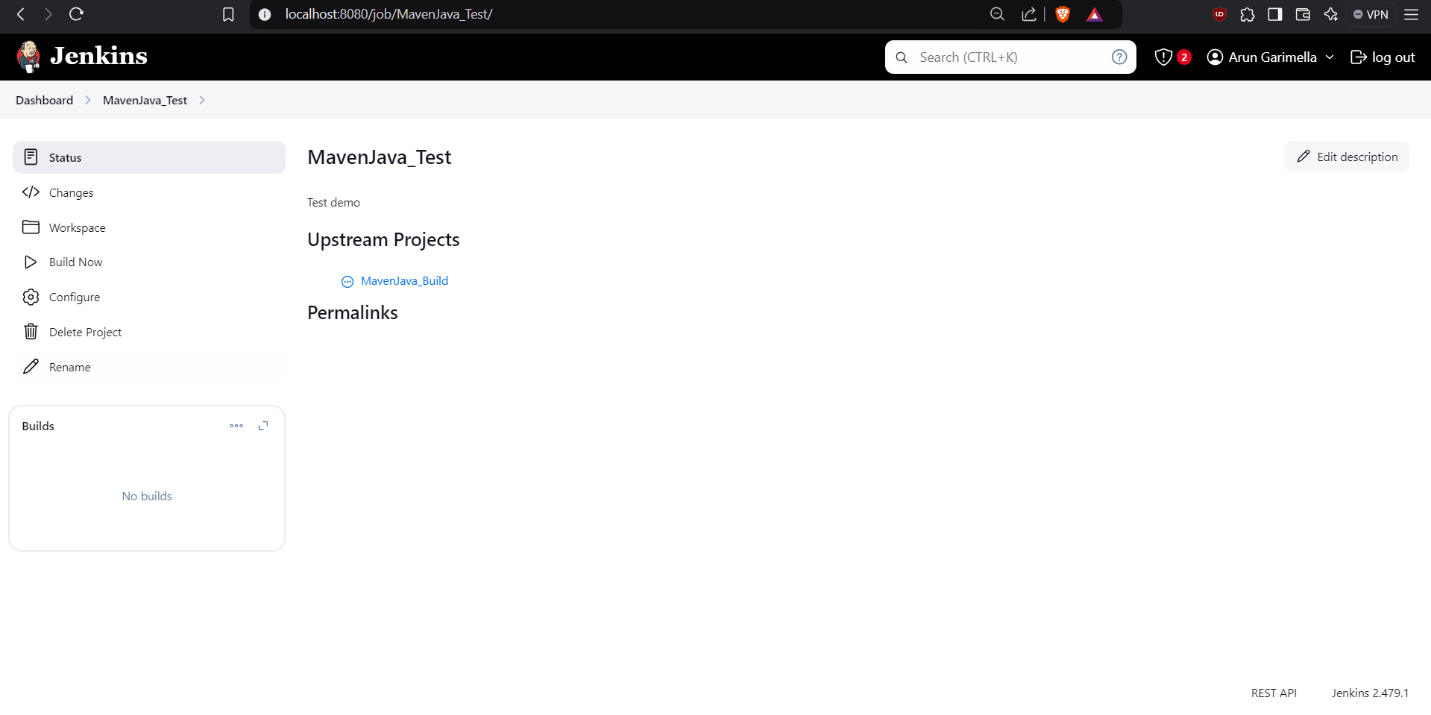
Step 13: Now Give project name: MavenJava\_Build, Which build: Check the boxstable build only, Artifacts to copy: \*\*/\*.

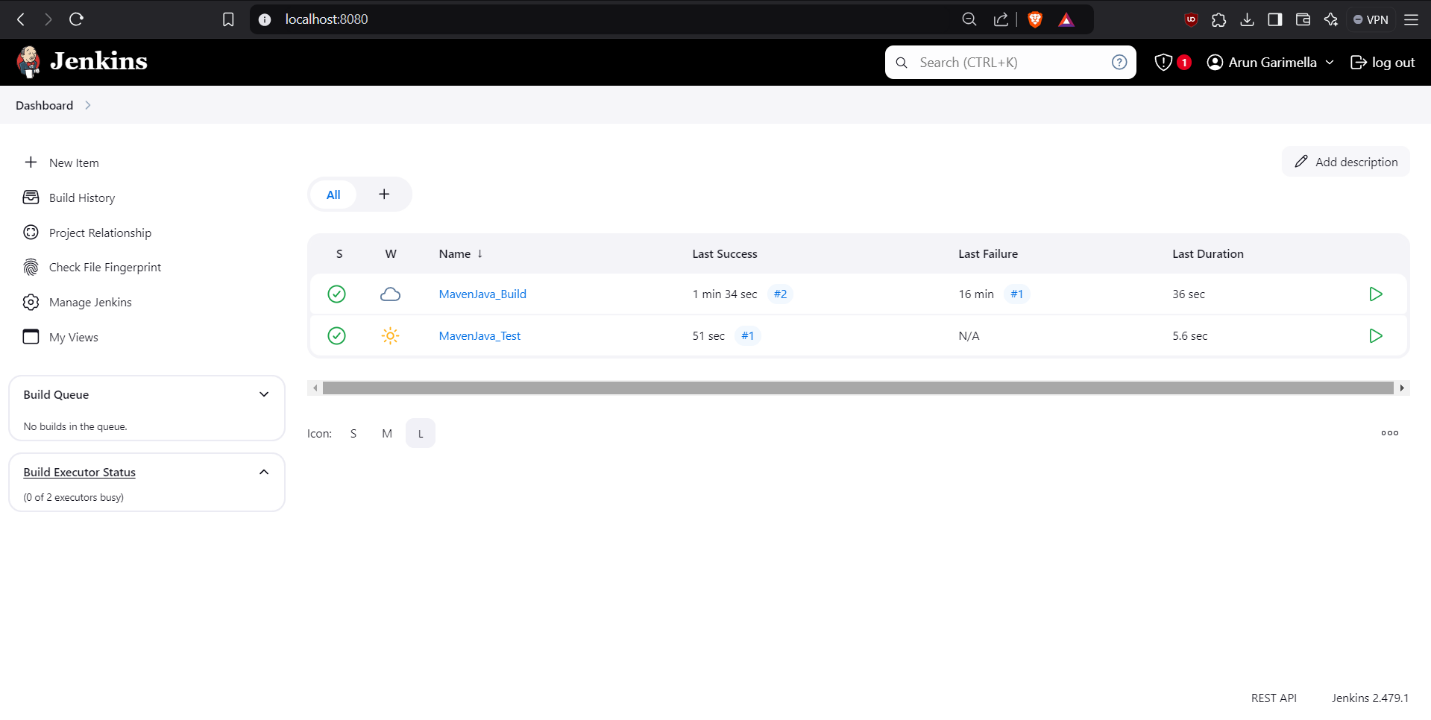


Step 14: Click Add Build Step-> Invoke top-level Maven targets-> Type MAVEN\_HOME in Maven version, Goals: test.

Step 15: Click on Post build actions->Archive the artifacts, in Files to archive type \*\*/\*.



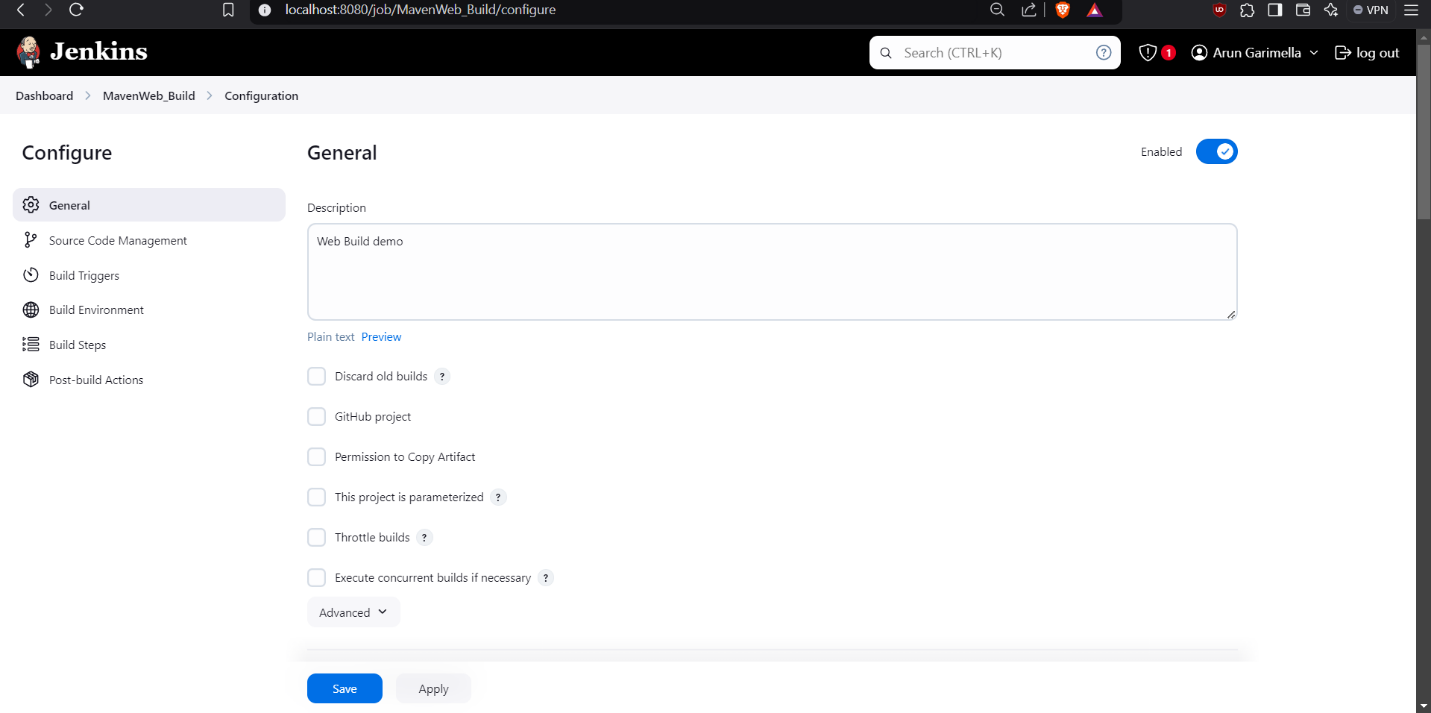
Step 16: Now click apply and save. Now run the project.

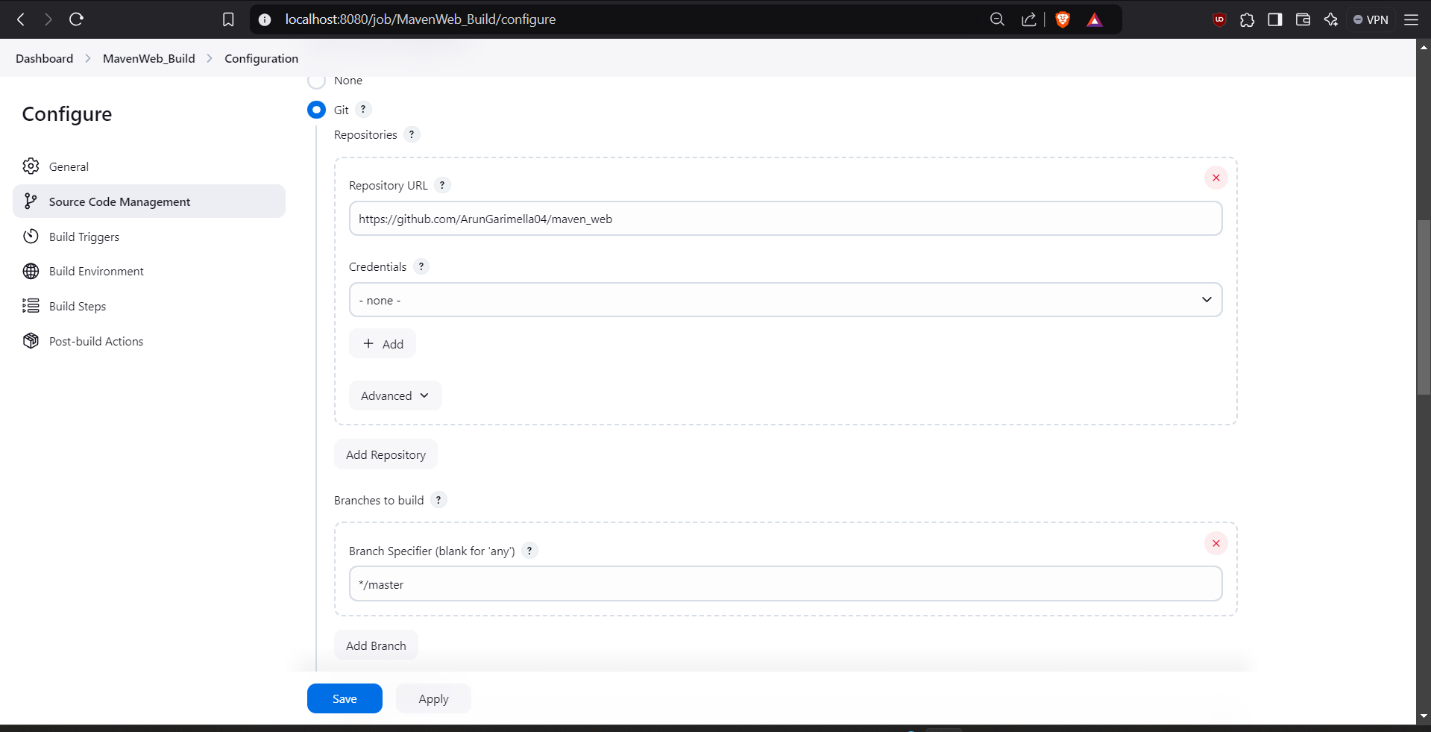


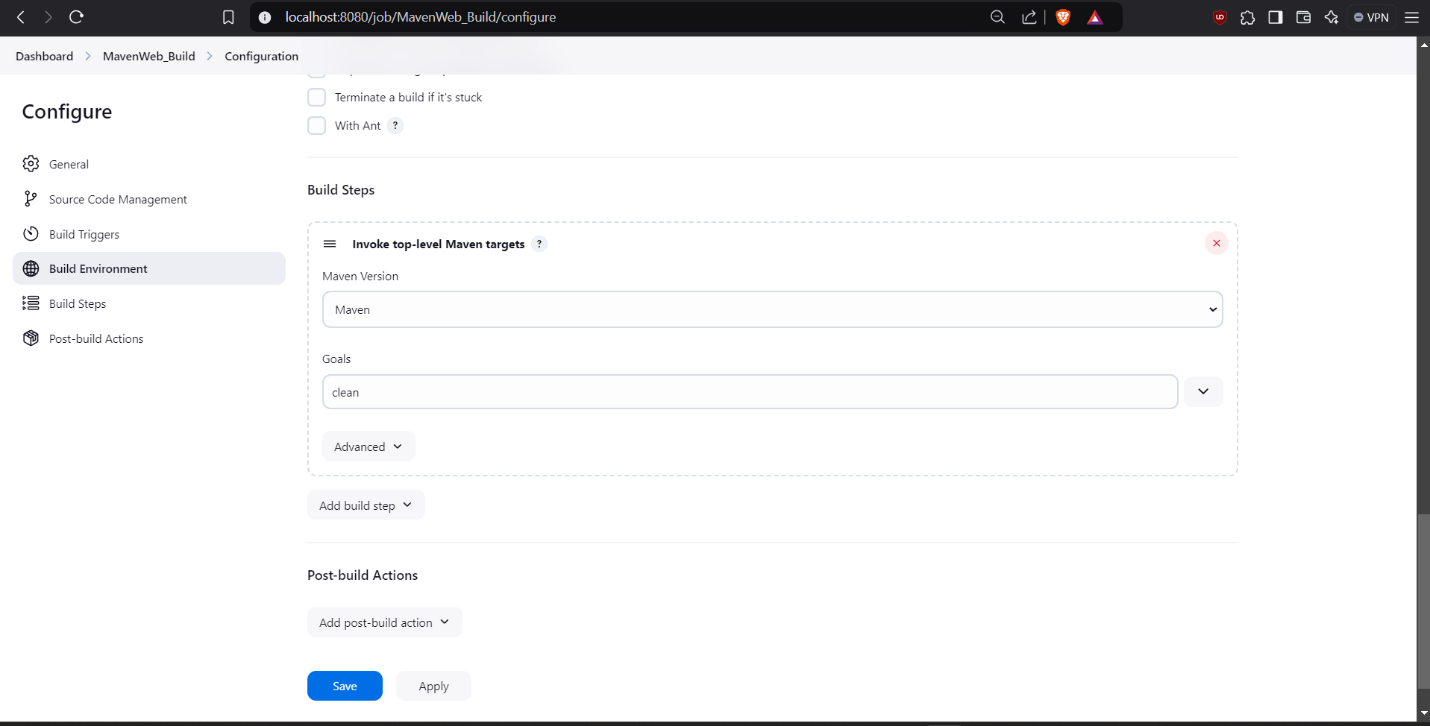
**Steps for Maven Web Automation:**

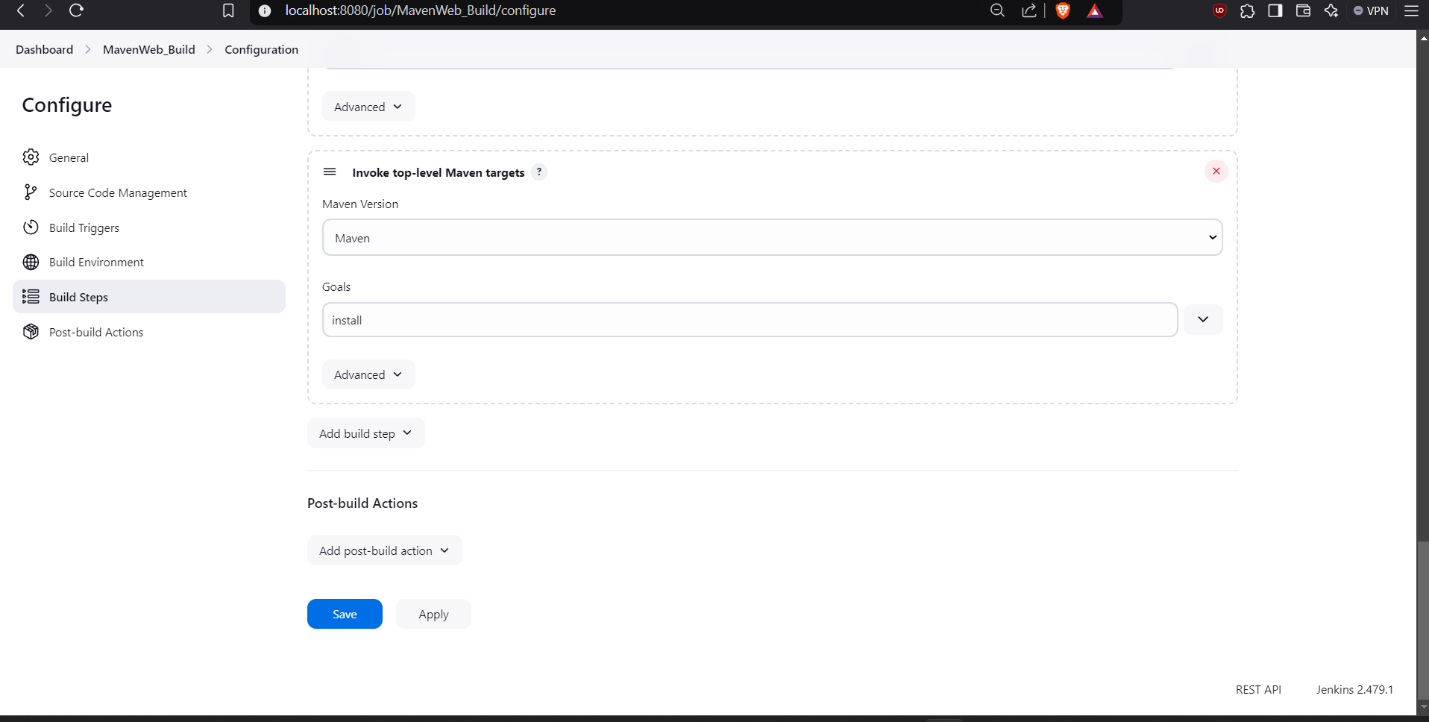
Step 1: Open Jenkins in local host: 8080/ and click on a new item present in the left side.

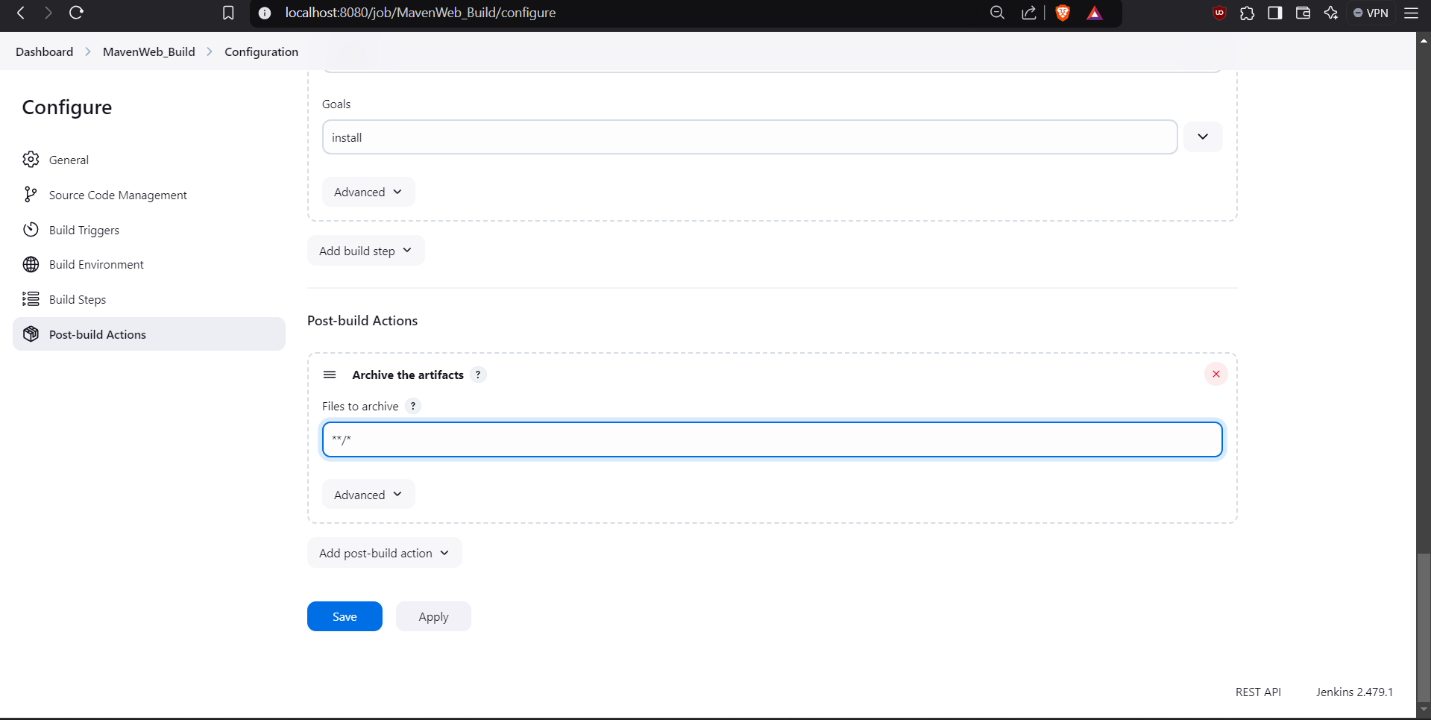
Step 2: Select a new Freestyle Project give name **MavenWeb\_Build** and then click ok.

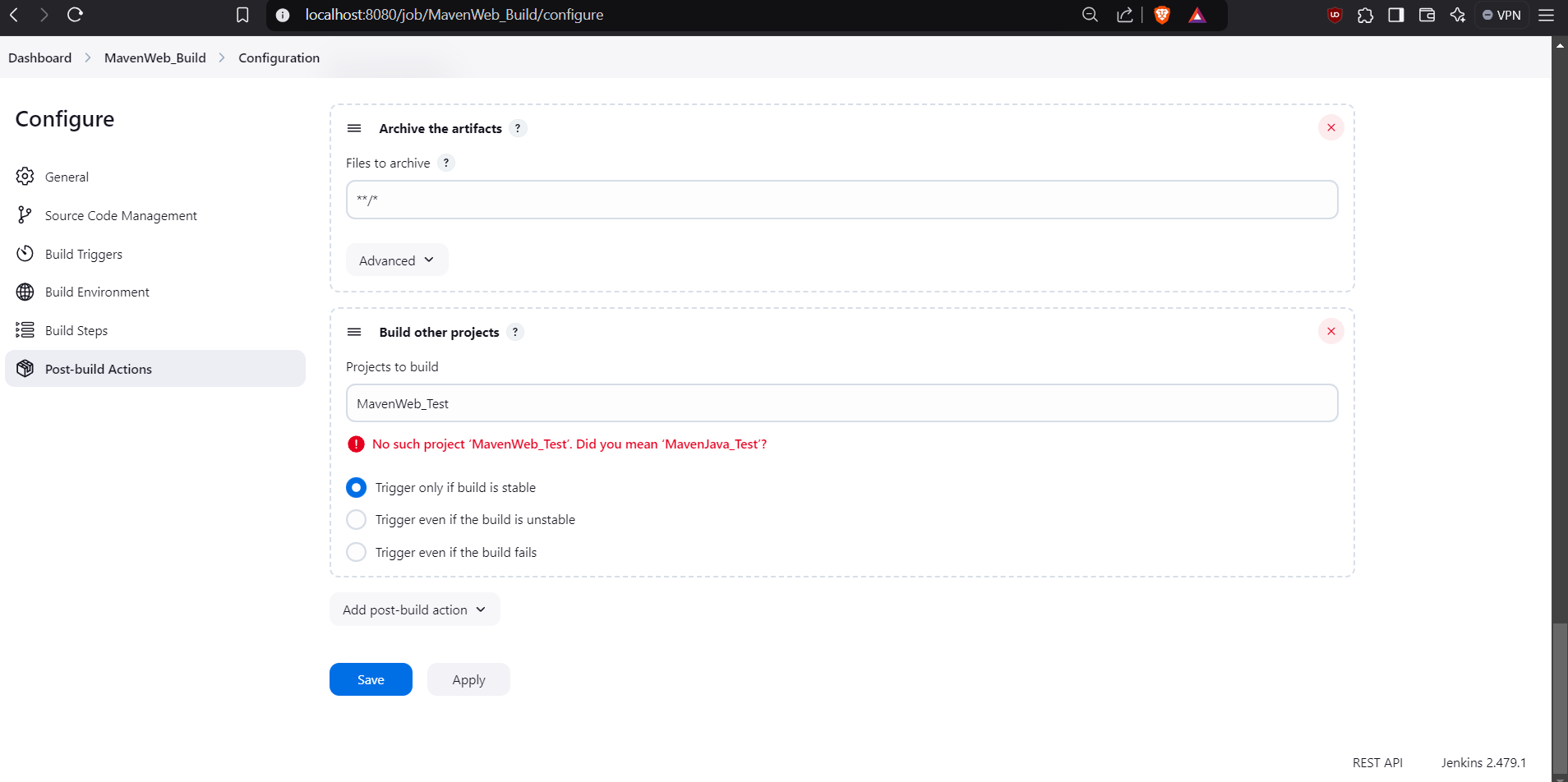
Step 3: In description type e.g., Web Build demo, Scroll sown and in Source code management give the Git MavenWeb repository URL of the project to be built.

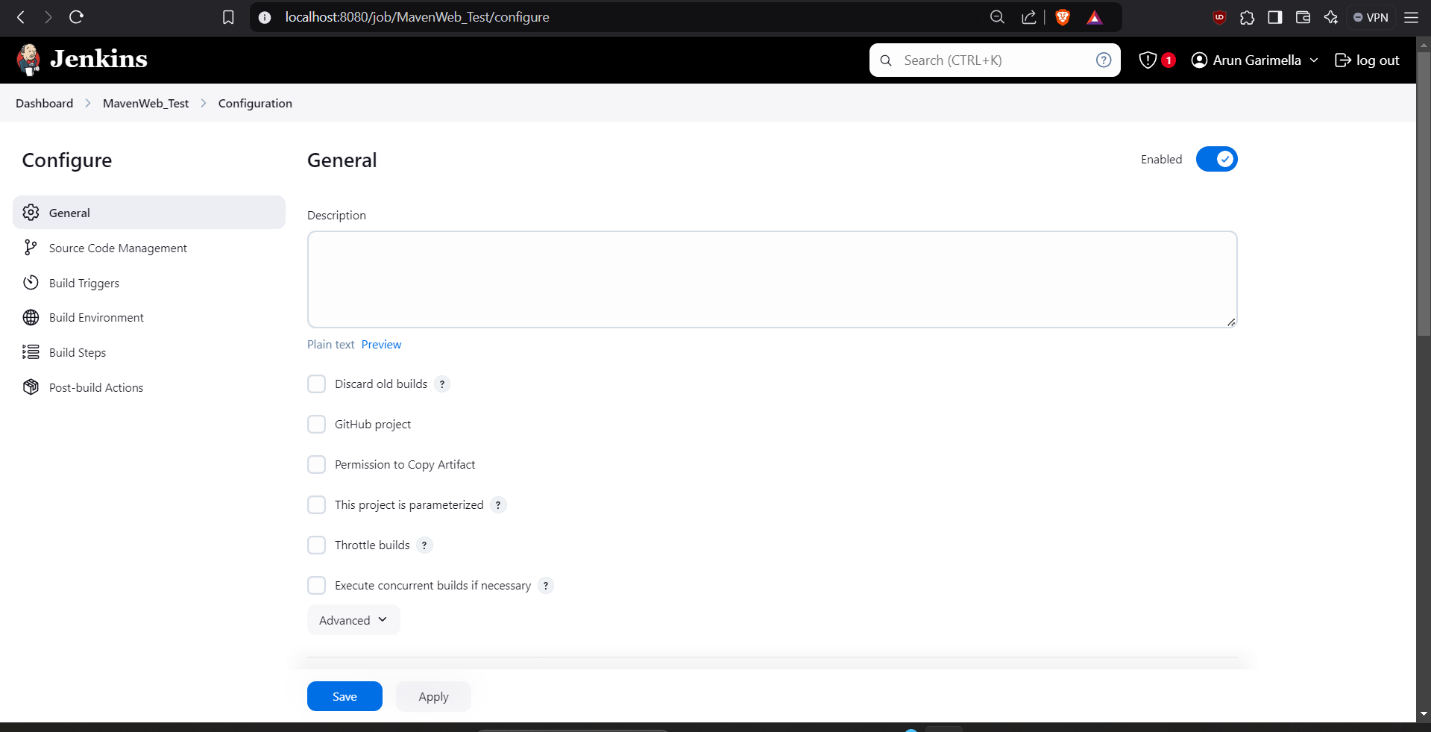
Step 4: Scroll down in Branches to build, Specify the Branch as \*/Main as it is in the GitHub

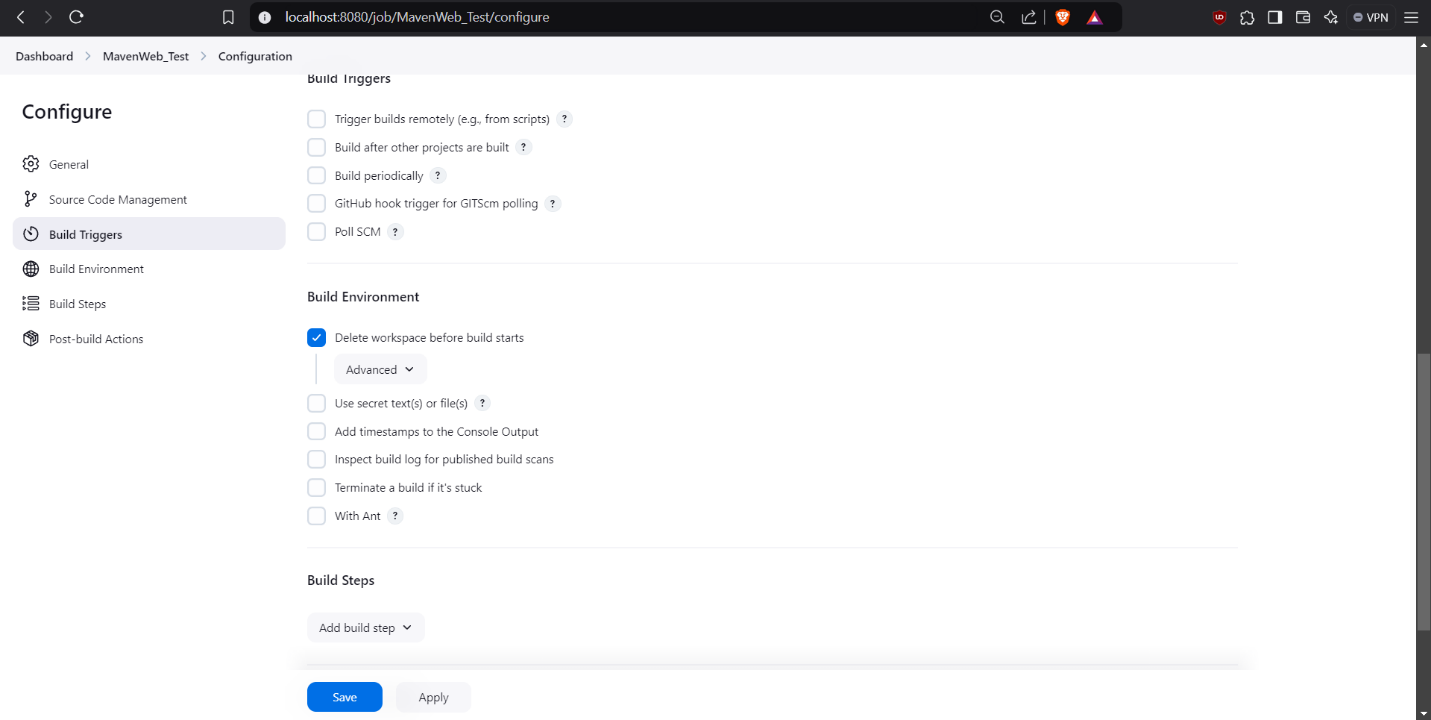
Step 5: Goto Build Steps -> Add Build Step-> Invoke top-level Maven targets, Type MAVEN\_HOME in Maven version, Goals: Clean.

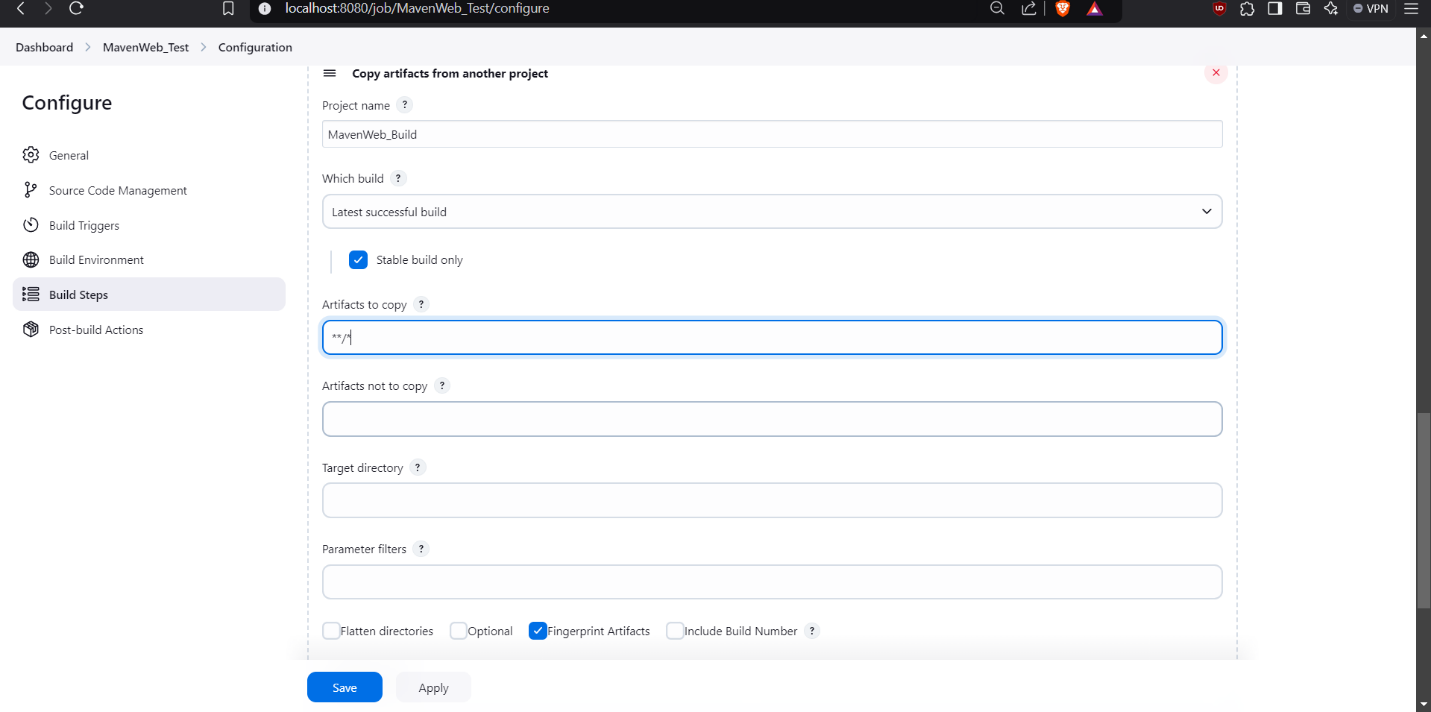
Step 6: Again, click Add Build Step-> Invoke top-level Maven targets-> Type MAVEN\_HOME in Maven version, Goals: install.

Step 7: Click add post build actions-> select Archive the artifacts, Type \*\*/\* in Files to Archive.

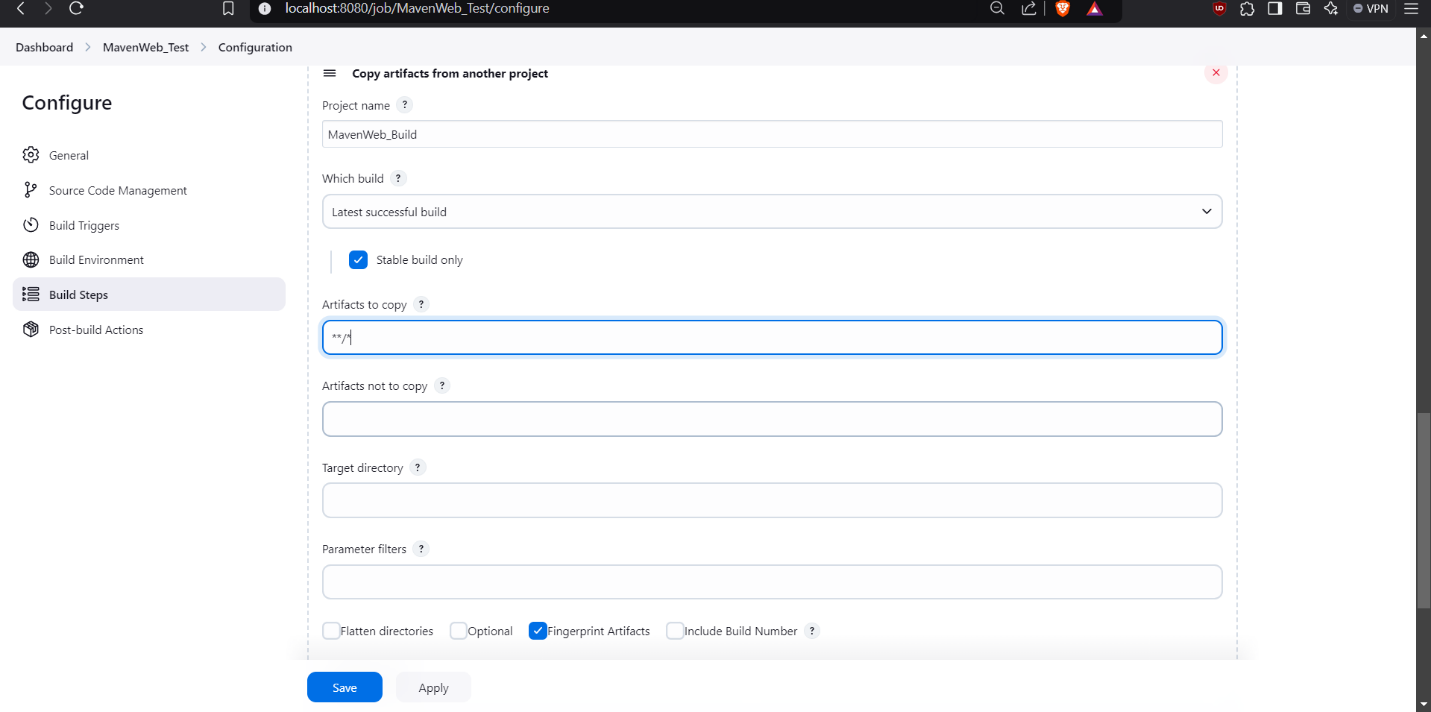
Step 8: Now click Add Post Build Action-> Build other projects**.**  In Projects to build, write MavenWeb\_Test and select Trigger only if build is stable. Click on apply and save at the bottom.

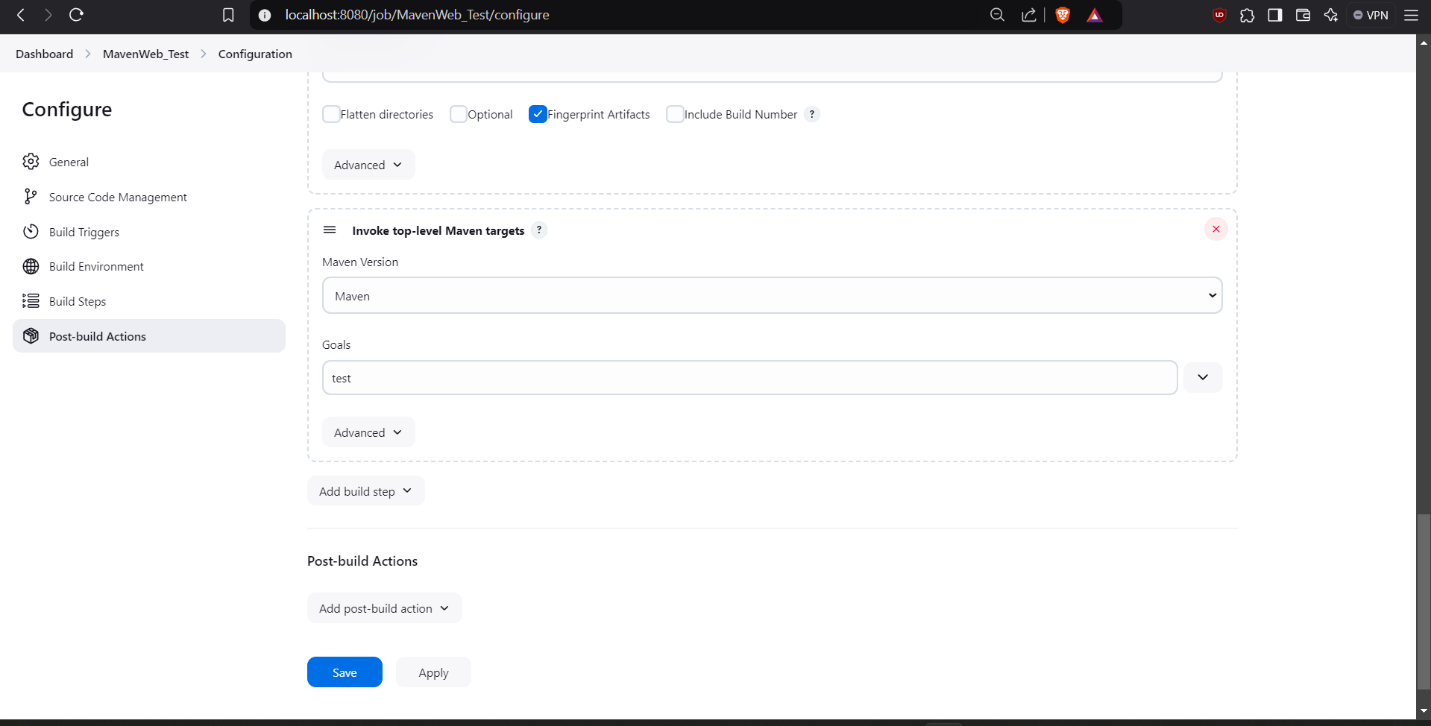
Step 9: Go to dashboard -> New item-> Freestyle Project, give project name as **MavenWeb\_Test**, then press on OK.

Step 10: In description type Test demo and in Build environment, check the box with name “Delete the workspace before build starts”.

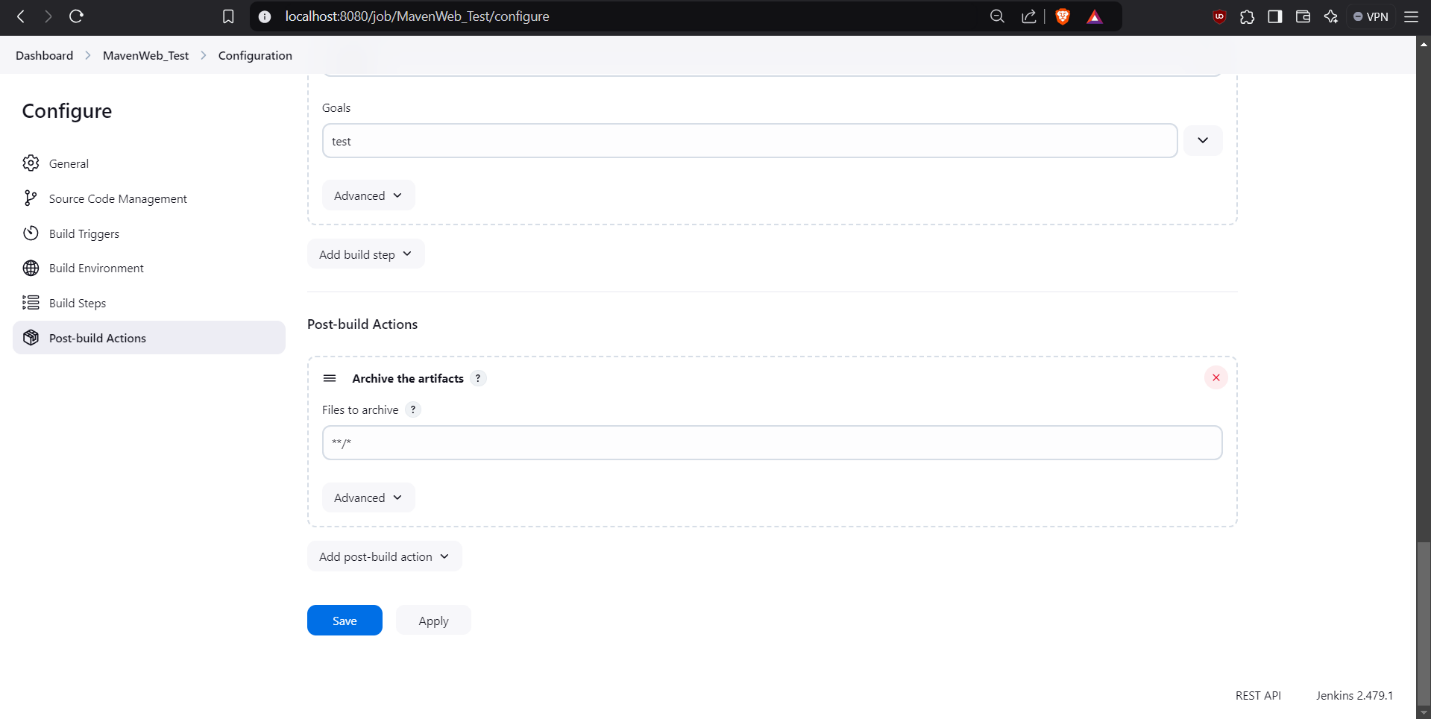
Step 11: Click on Add Build Step-> copy the artifacts from another project.

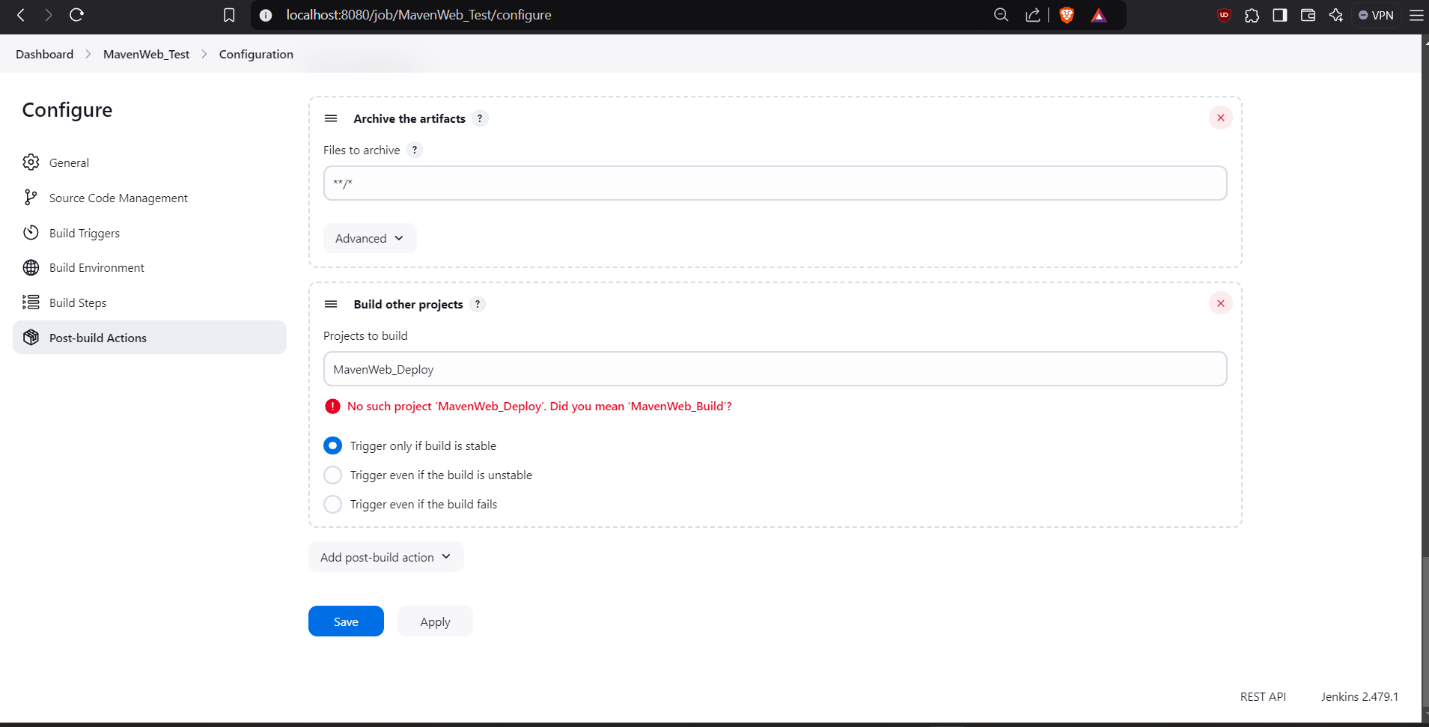
Step 12: Give project name: MavenWeb\_Build, Which build: Check the box stable build only, Artifacts to copy: \*\*/\*.



Step 13: Click Add Build Step-> Invoke top-level Maven targets-> Type MAVEN\_HOME in Maven version, Goals: test.

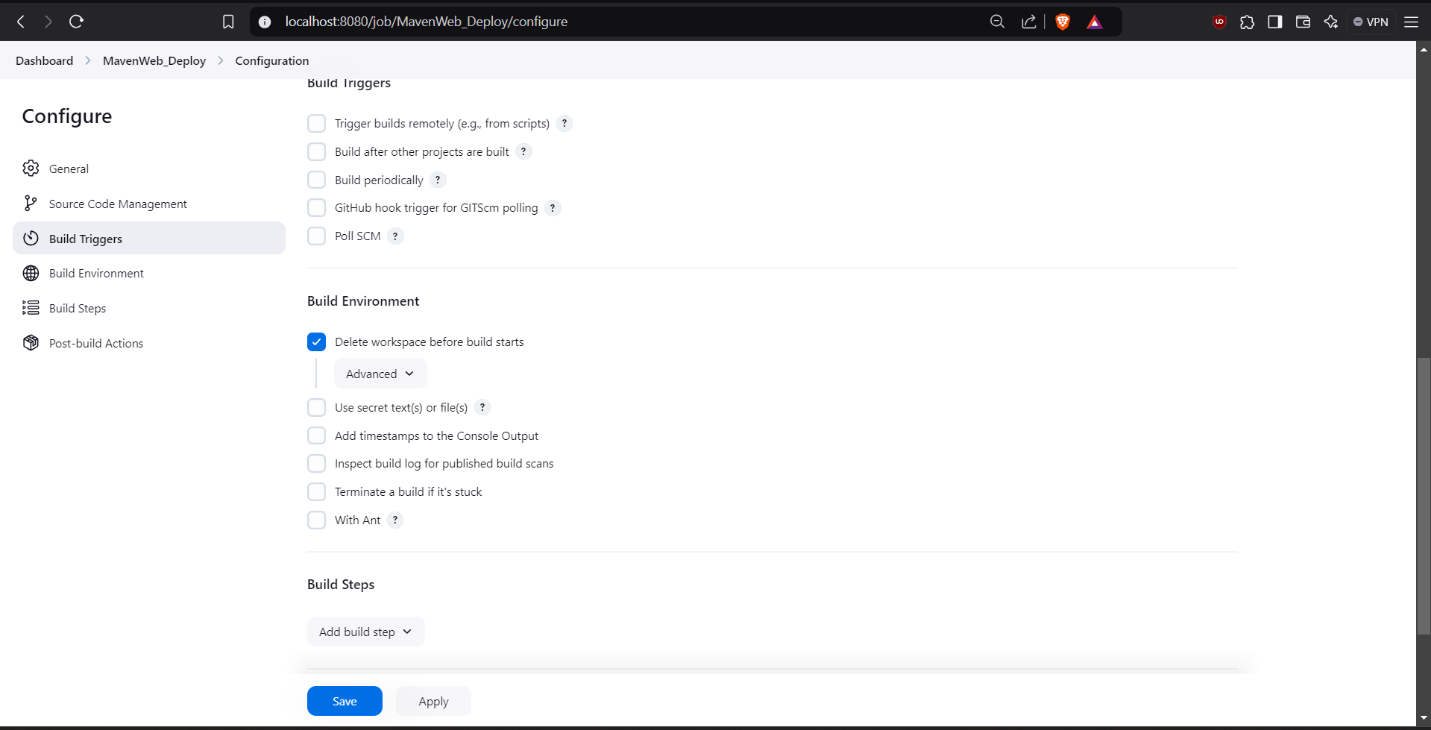
Step 14: Click on Post build actions->Archive the artifacts, in Files to archive type \*\*/\*.

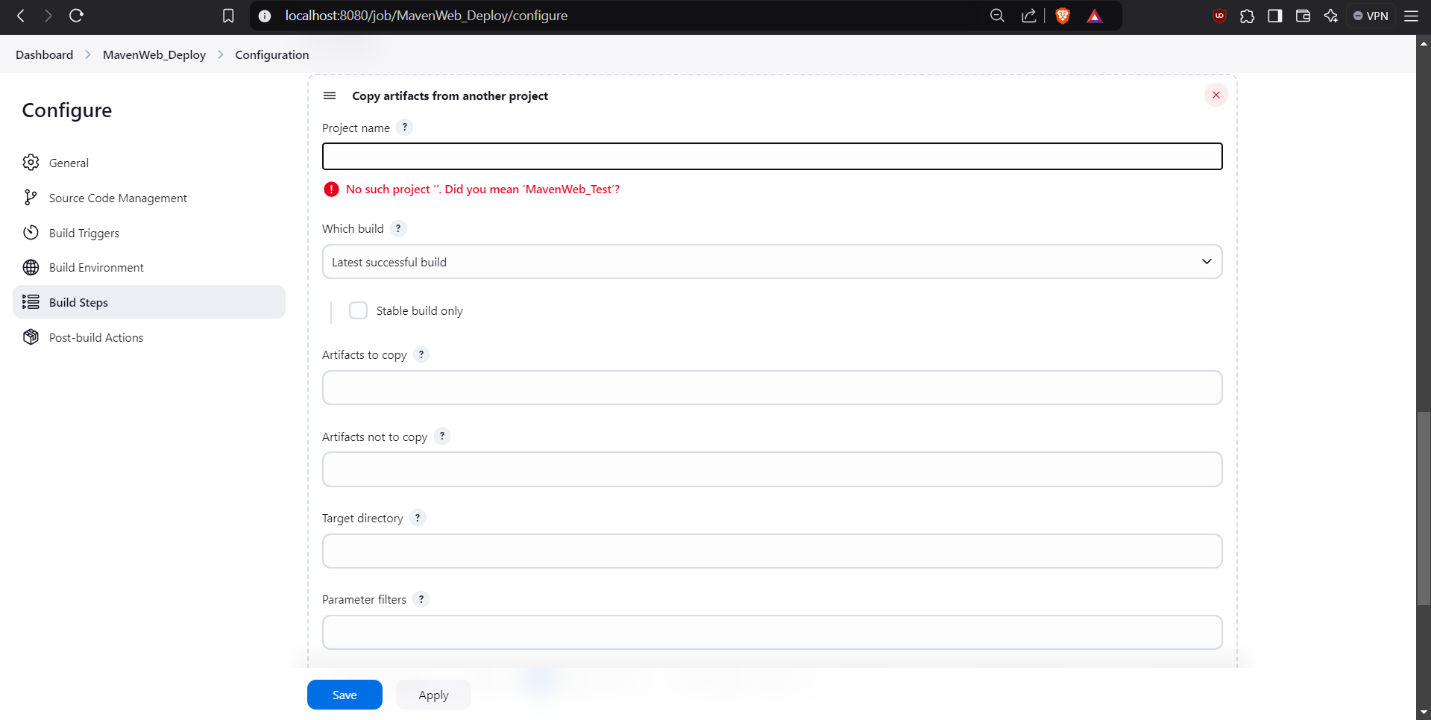


Step 15: Add post build action->Build Other Projects->MavenWeb\_Deploy, Apply and Save.

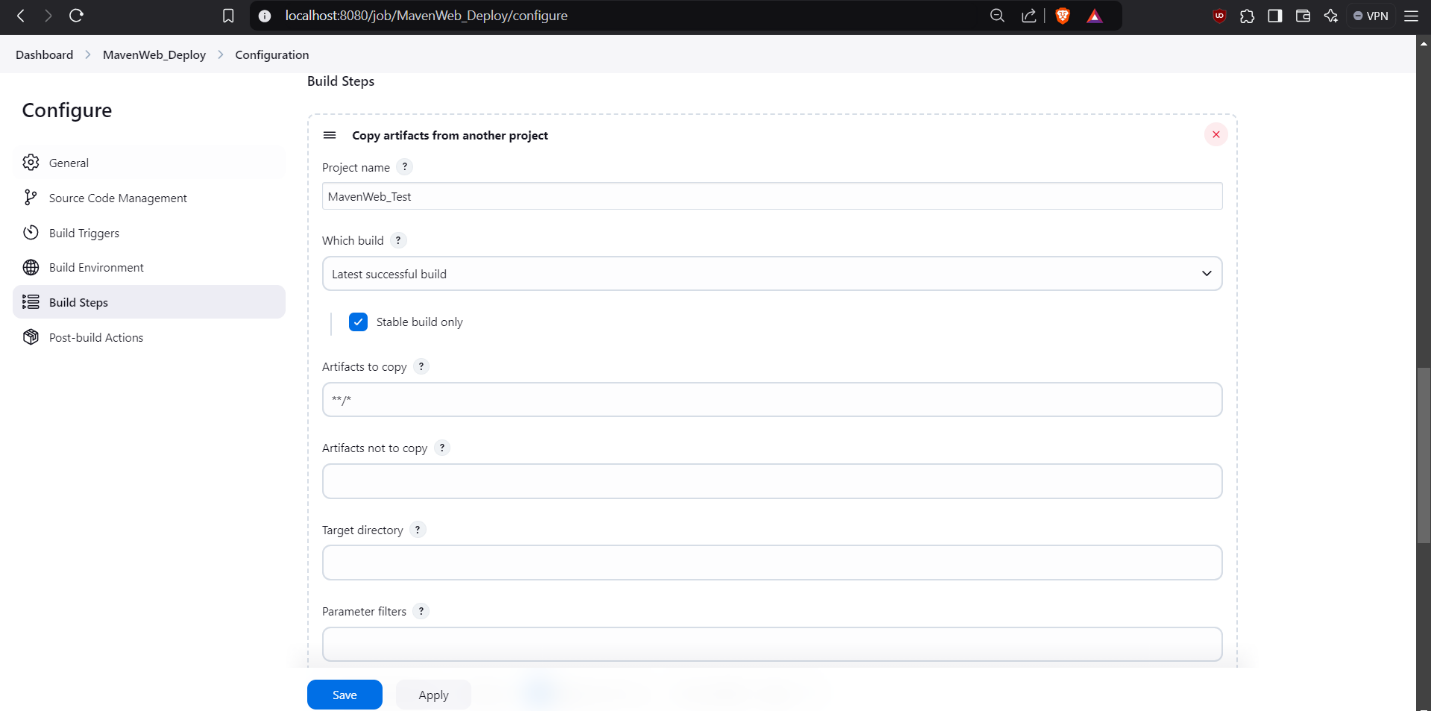
Step 16: Go to dashboard -> New item-> Freestyle Project, give project name as **MavenWeb\_Deploy**, then press on OK.

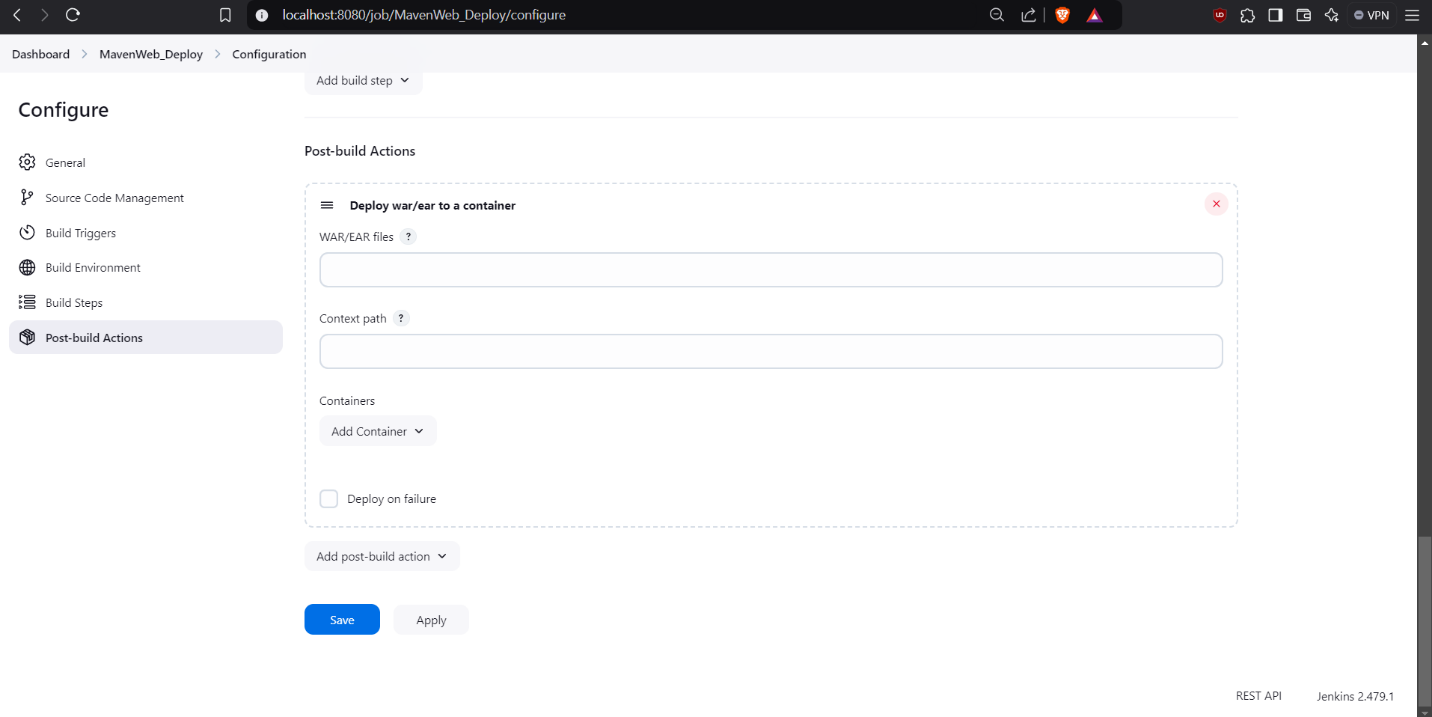
Step 17: In description give Web Code Deploement, scroll down to Build environment, Check “Delete the workspace before build starts”.



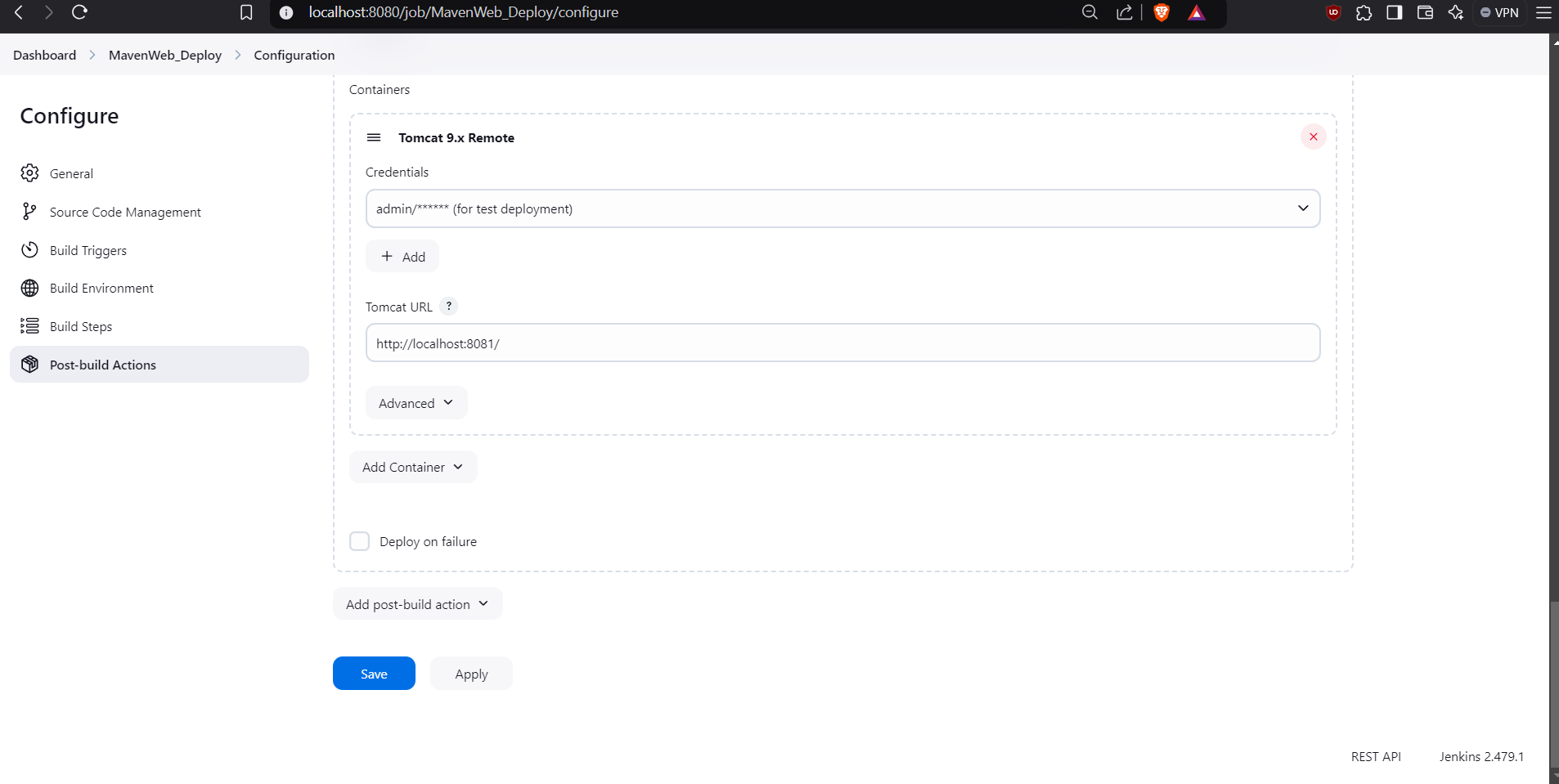
Step18: Goto Build Steps->Add Build Step-> copy the artifacts from another project.

Step 19: Give project name: MavenWeb\_Test, Which build: Check the box stable build only, Artifacts to copy: \*\*/\*.

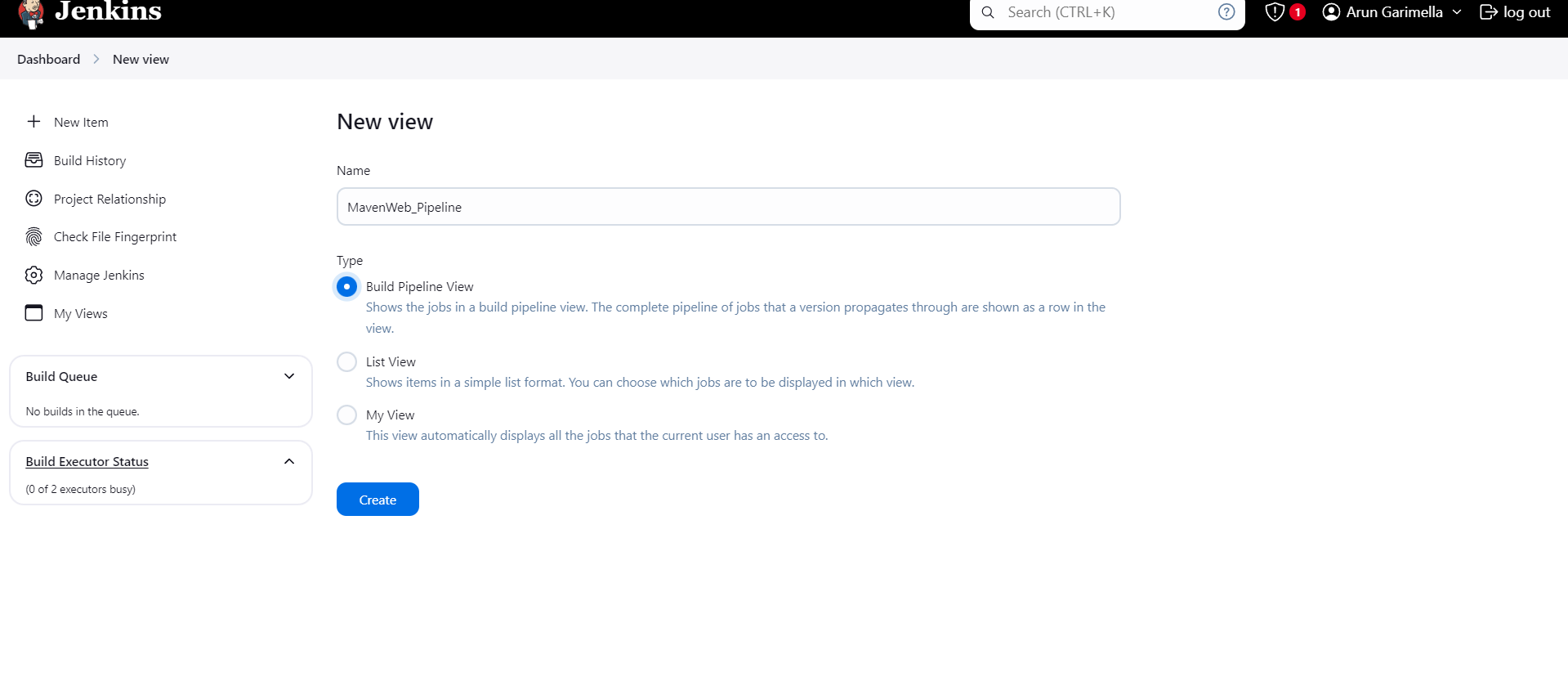


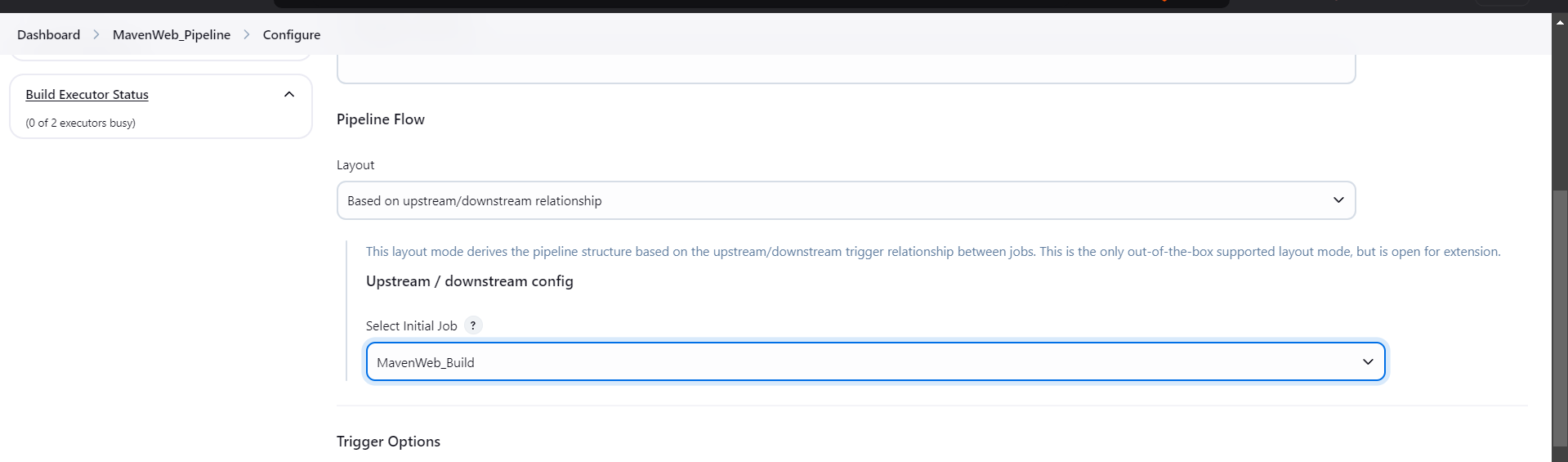
Step 20: Scroll down to post build actions-> Add post build action->Deploy war/ear to a container.

Step 21: Write WAR/EAR File: \*\*/\*.war, Context path: Webpath, click on Add container-> Tomact 9.x remote, Click on admin, add credentials as username: admin and password: 1234, save.

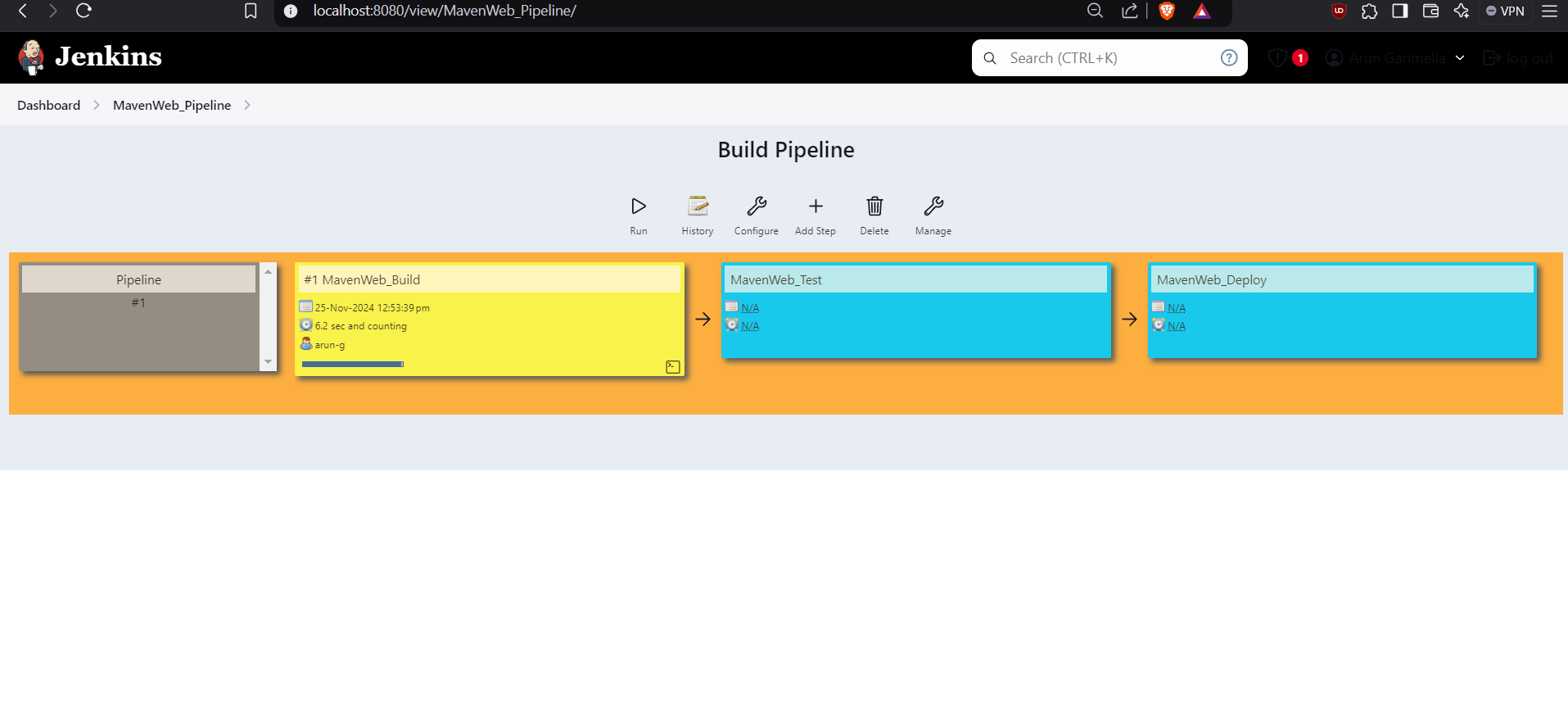


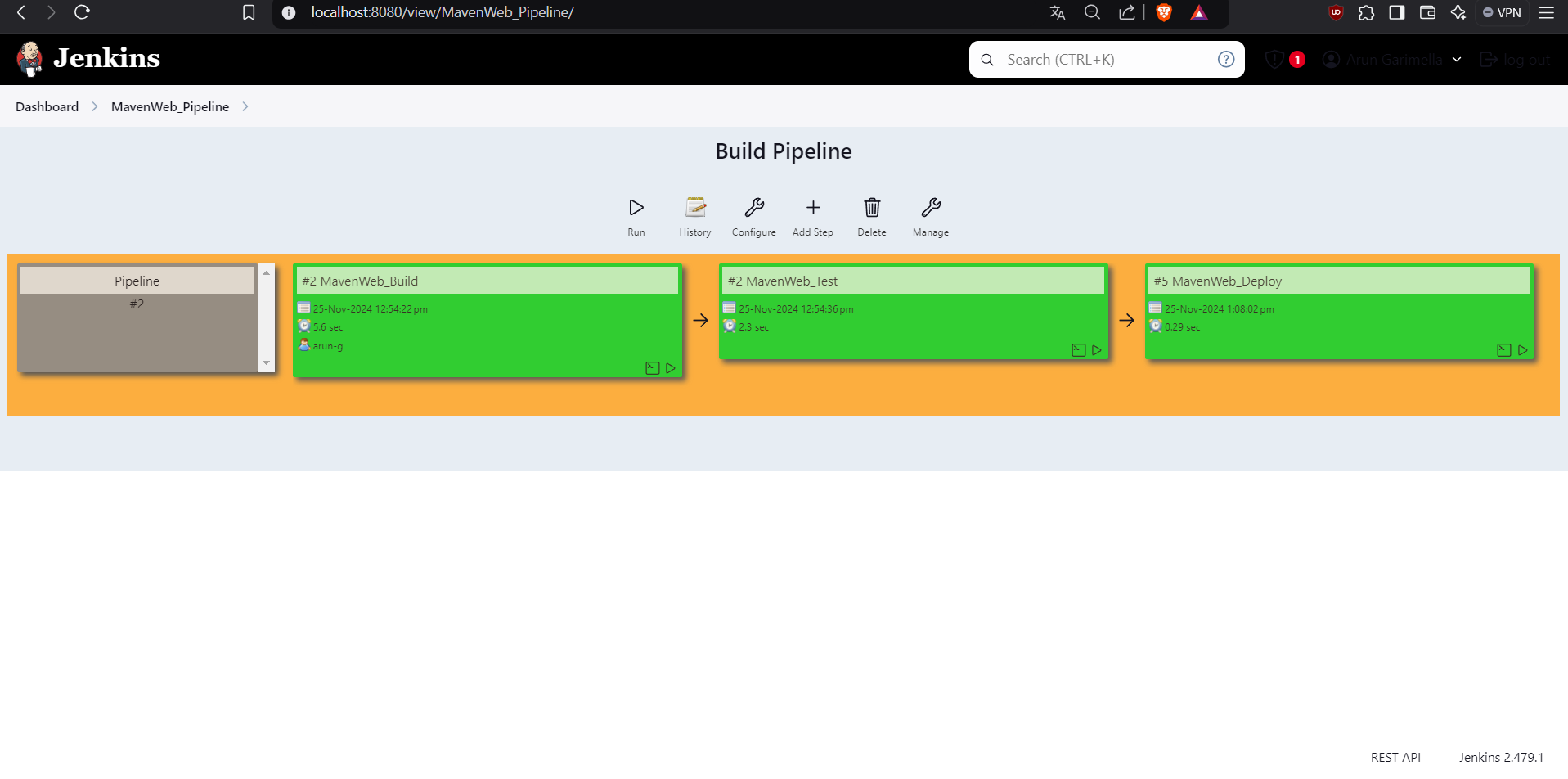
Step 22: Give Tomcat URL: https://localhost:8085/ , Click on apply and save.

Step 23: Now click on “+” button present beside “All” in the dashboard, Give Name: MavenWeb\_Pipeline, select Build pipeline view in the options shown.

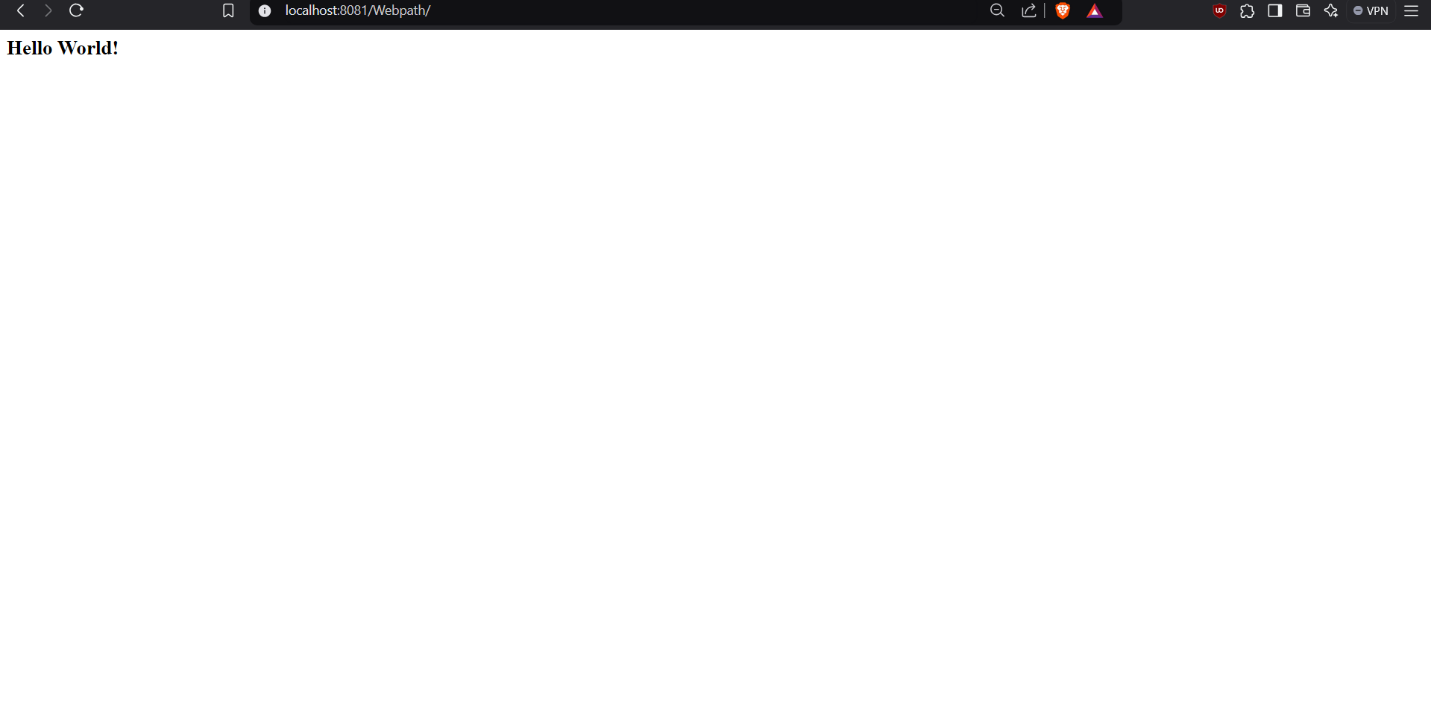
Step 24: Scroll down, goto Pipeline Flow->Layout->Based on upstream/downstream relationship-> Select initial job-> MavenWeb\_Build, Apply and save.

Step 25: Click the trigger to run the pipeline shown on the page for web application automation.





Step 26: Now open tomcat homepage in another tab and check the **manager app** option present at the right side.

Step 27: Click on /webpath option, and now output of MavenWeb project is seen in the tomcat server in a new tab.

**Creating a Scripted pipeline using Jenkins:**

**Pipeline Script:**

pipeline {

agent any

tools{

maven 'Maven'

git 'GIT\_HOME'

}

stages {

stage('git repo & clean') {

steps {

//bat "rmdir /s /q maven\_java"

bat "git clone https://github.com/ArunGarimella04/maven\_java.git"

bat "mvn clean -f maven\_java"

}

}

stage('install') {

steps {

bat "mvn install -f maven\_java" //project name

}

}

stage('test') {

steps {

bat "mvn test -f maven\_java"

}

}

stage('package') {

steps {

bat "mvn package -f maven\_java"

}

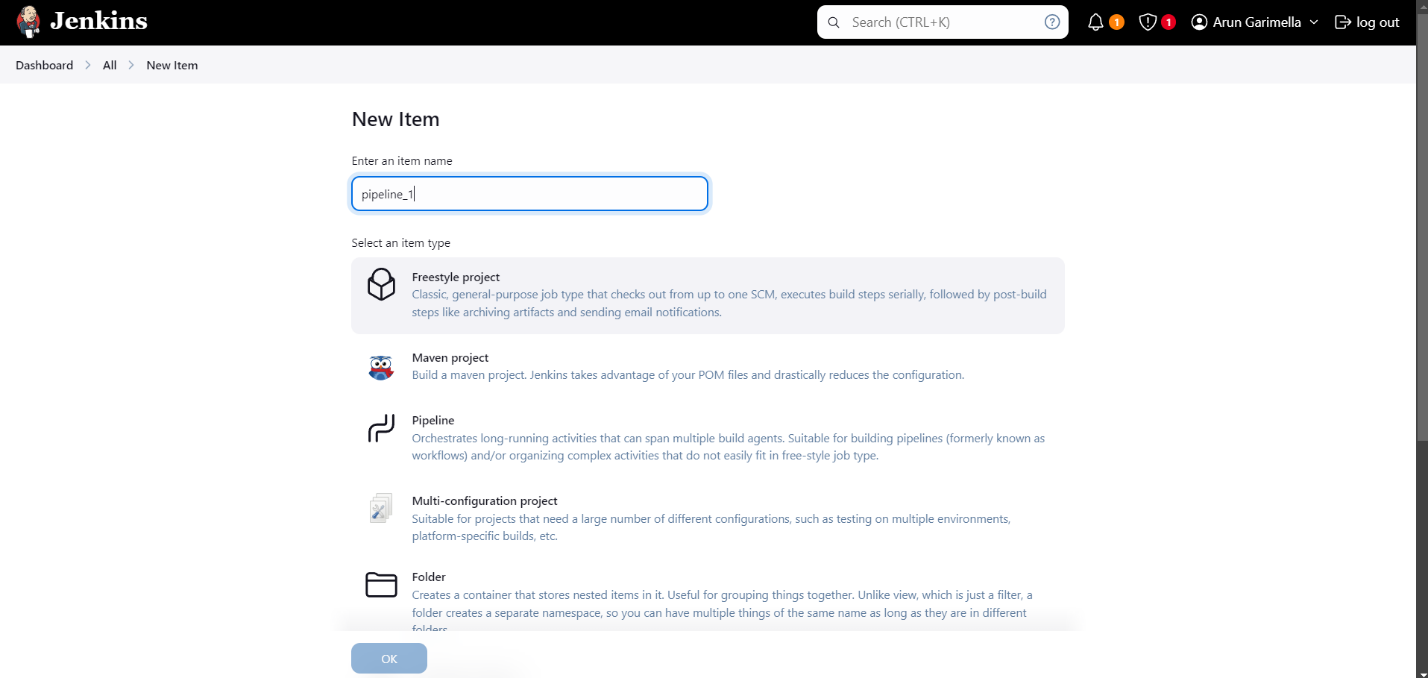
}

}

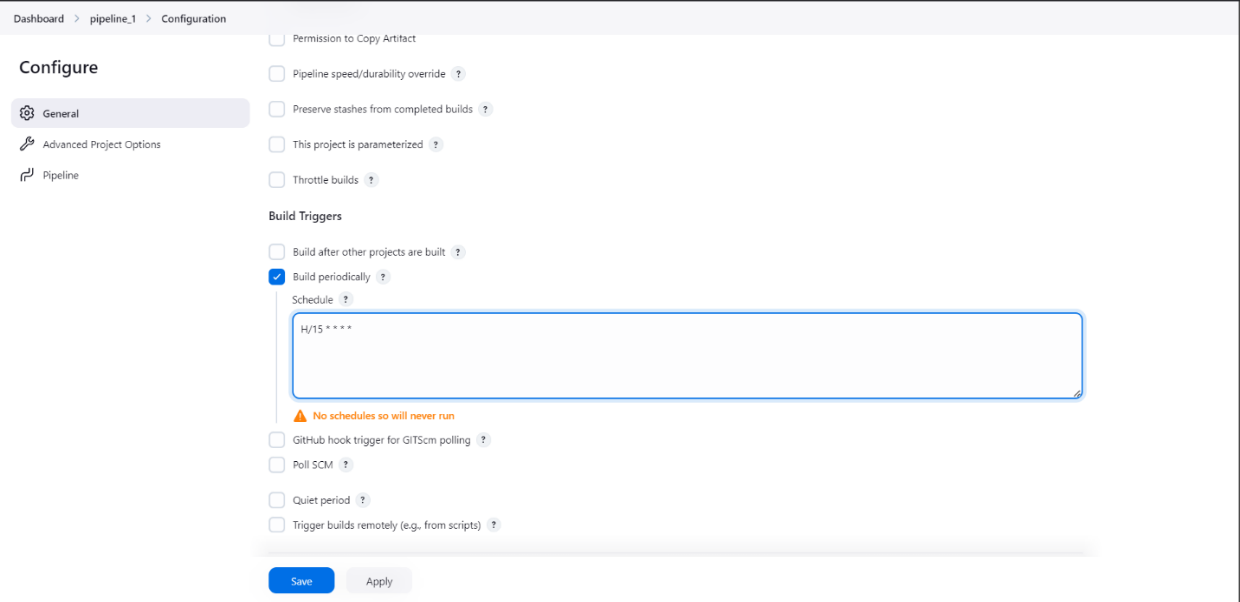
}

**Steps to create a Build Trigger for maven Java project using poll SCM:**

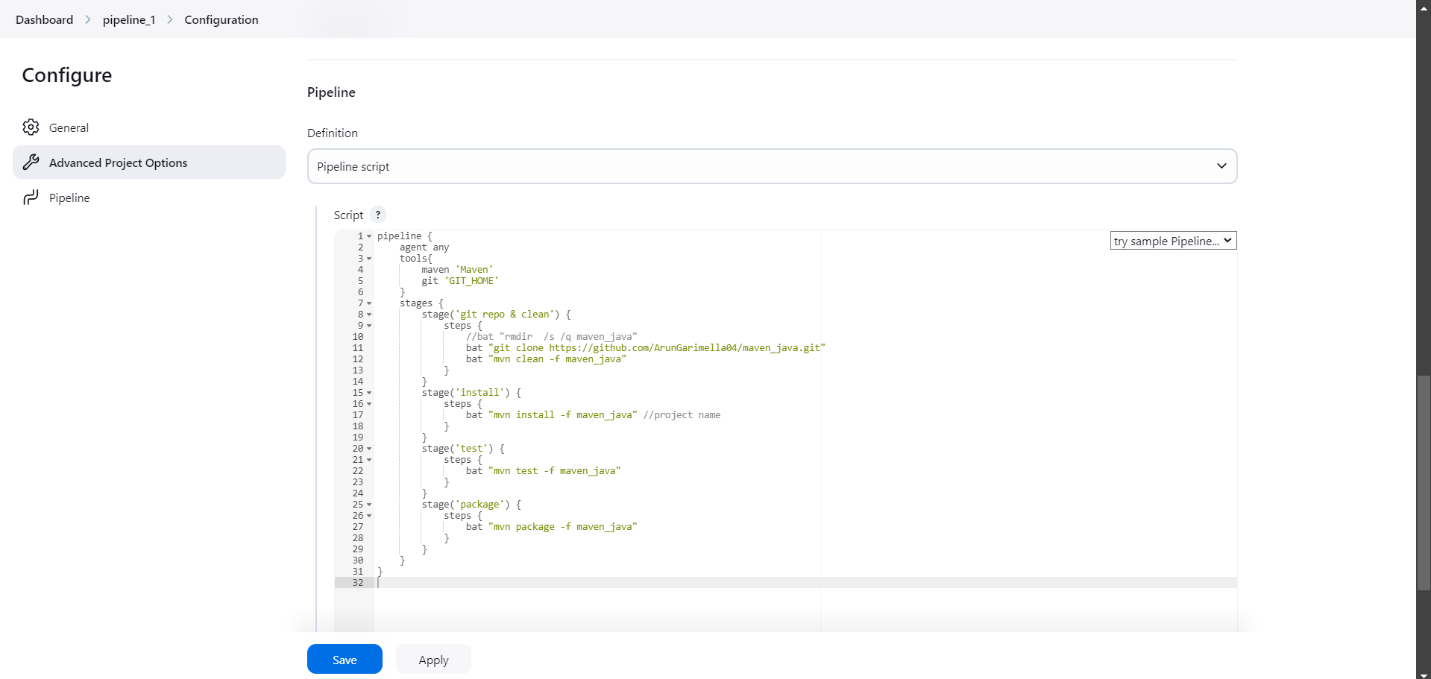
1. Create a Jenkins Pipeline.



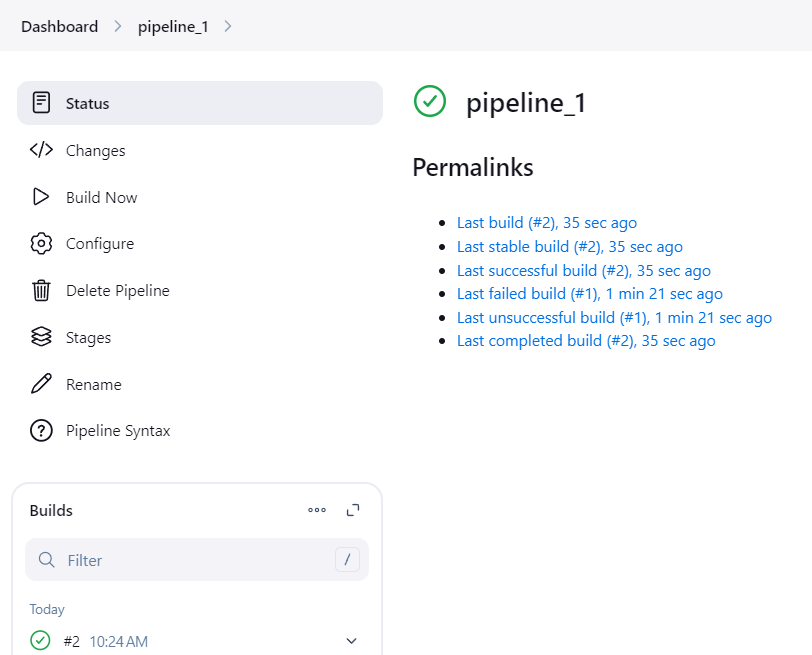
2. Select “Build periodically” in Build Triggers section and type “H/15 \* \* \* \*” in the description.



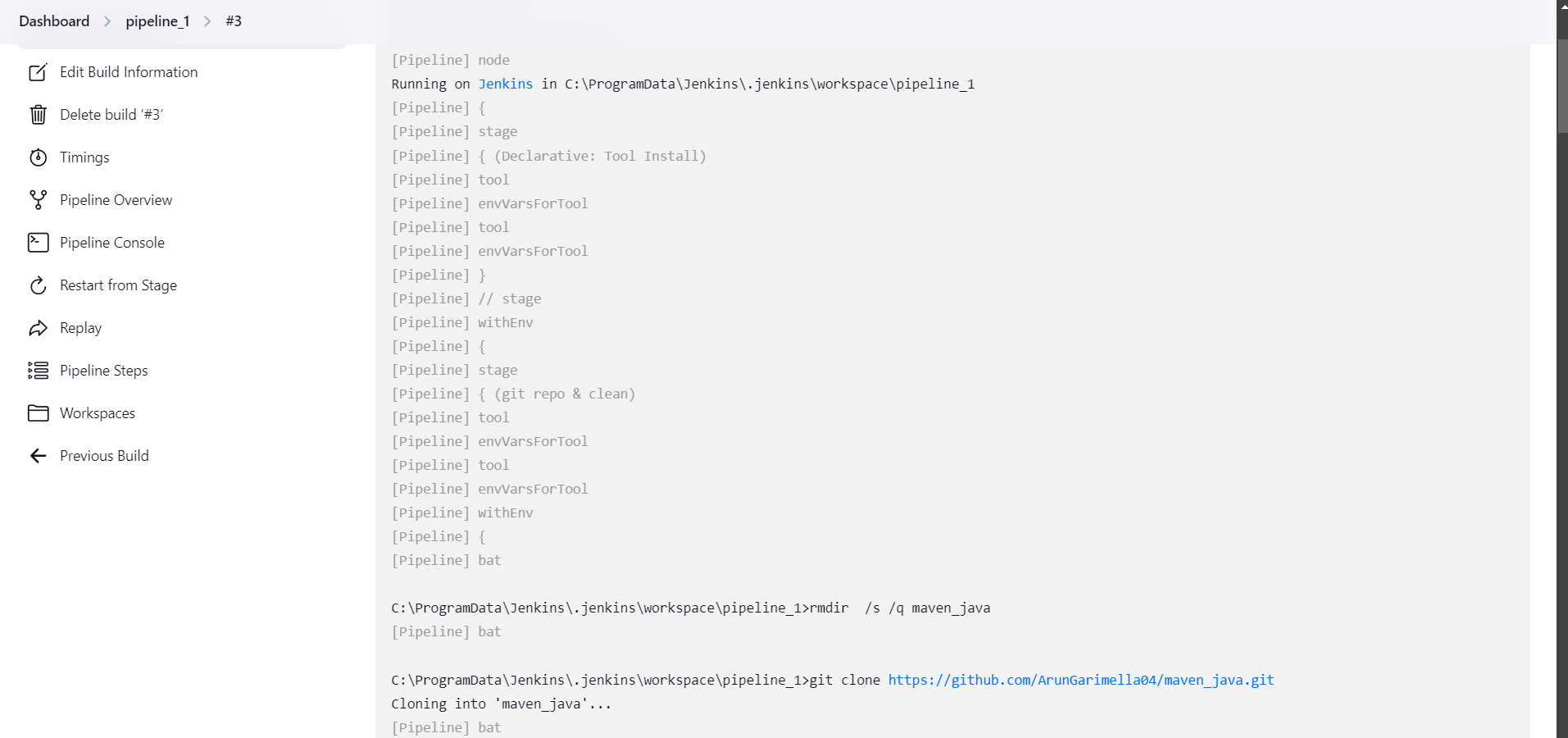
3. Add your pipeline script in the Pipeline definition section.

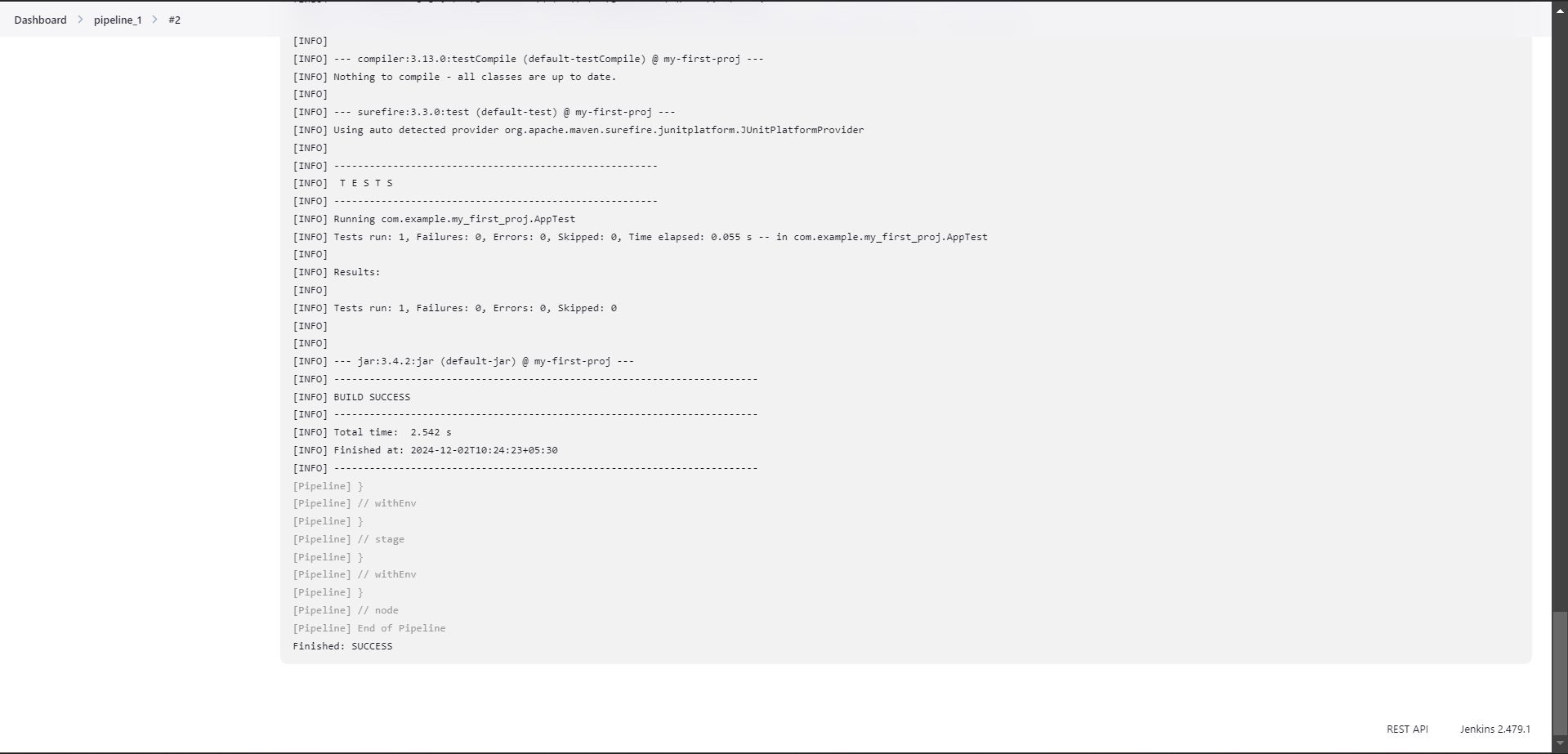


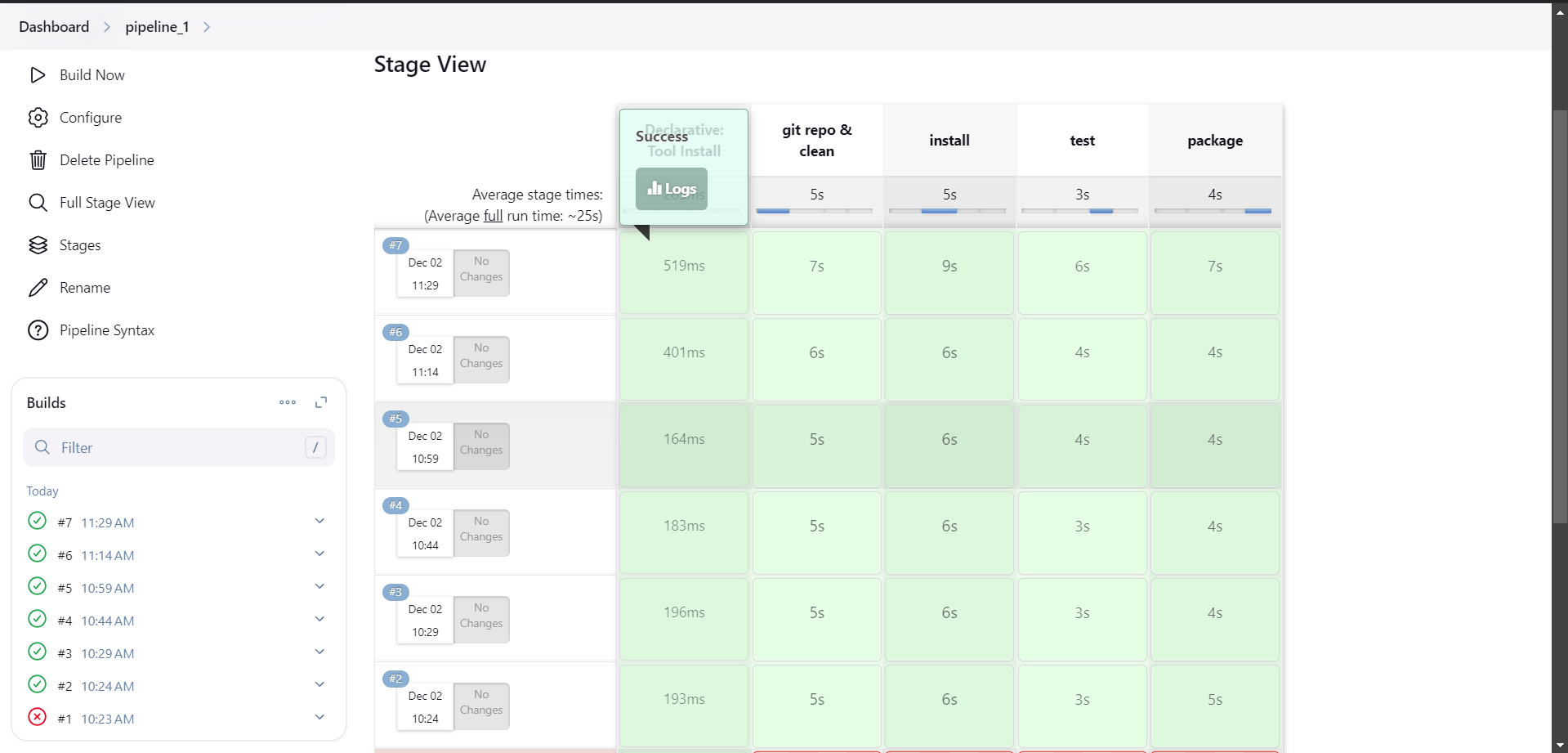
4. Click on Build Now option and wait for the build to complete.



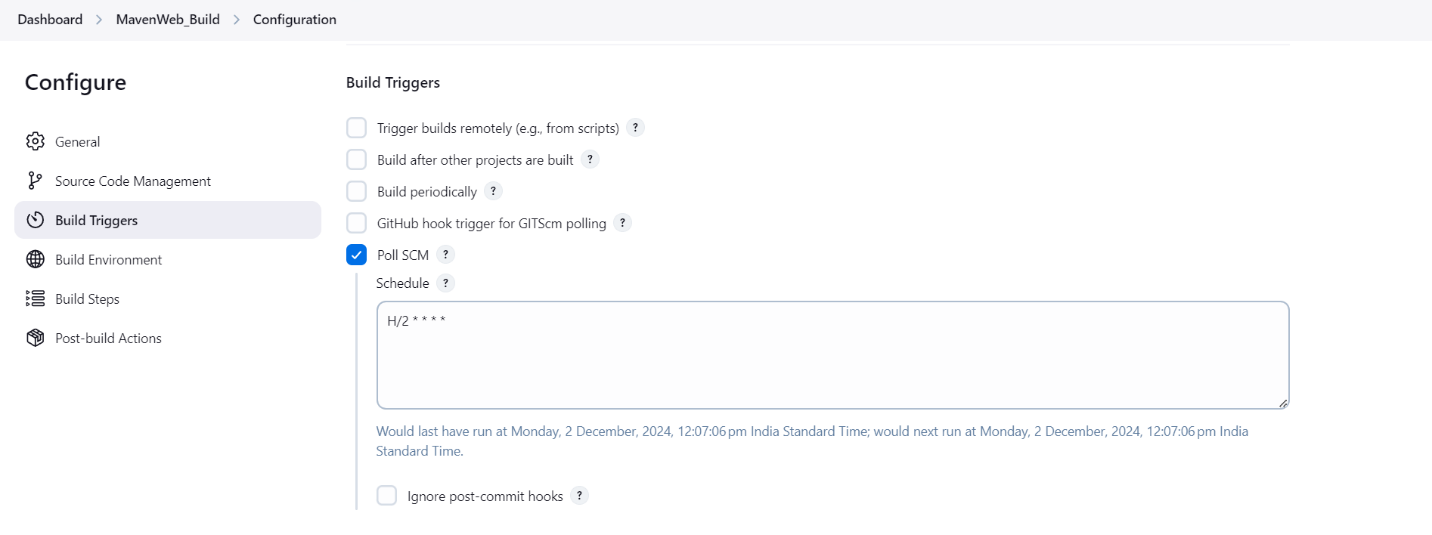
5. You will see a “BUILD SUCCESS” output in the Console Output.

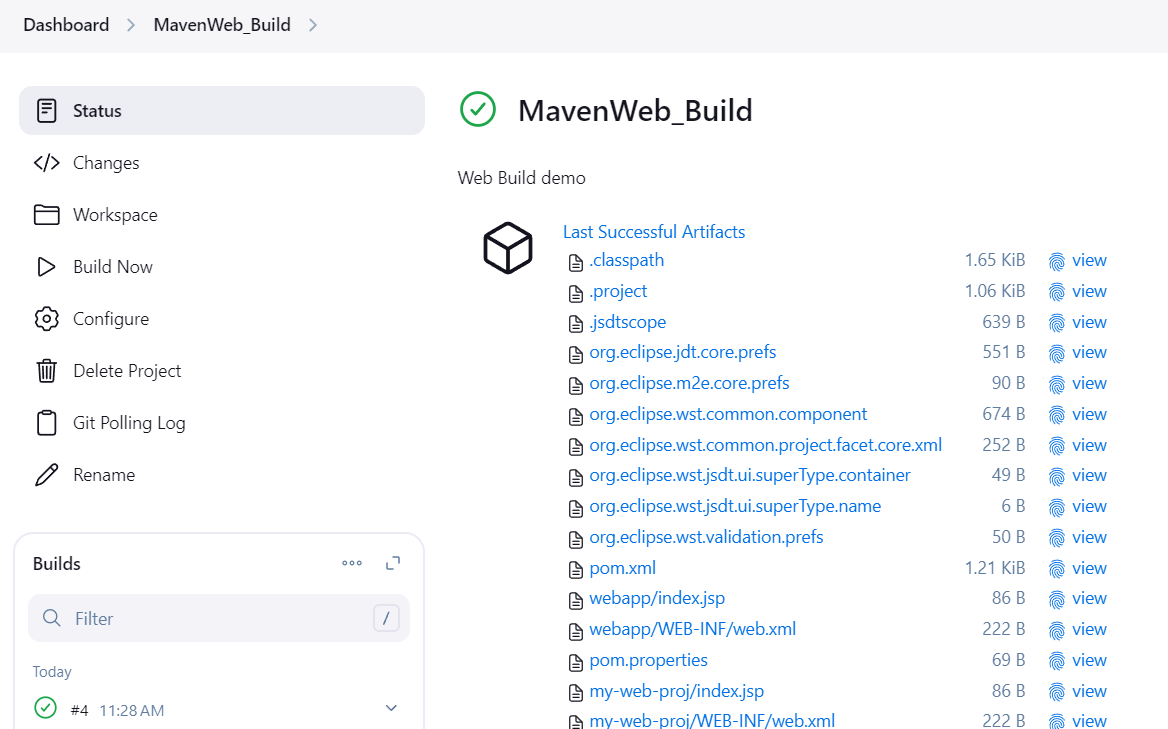




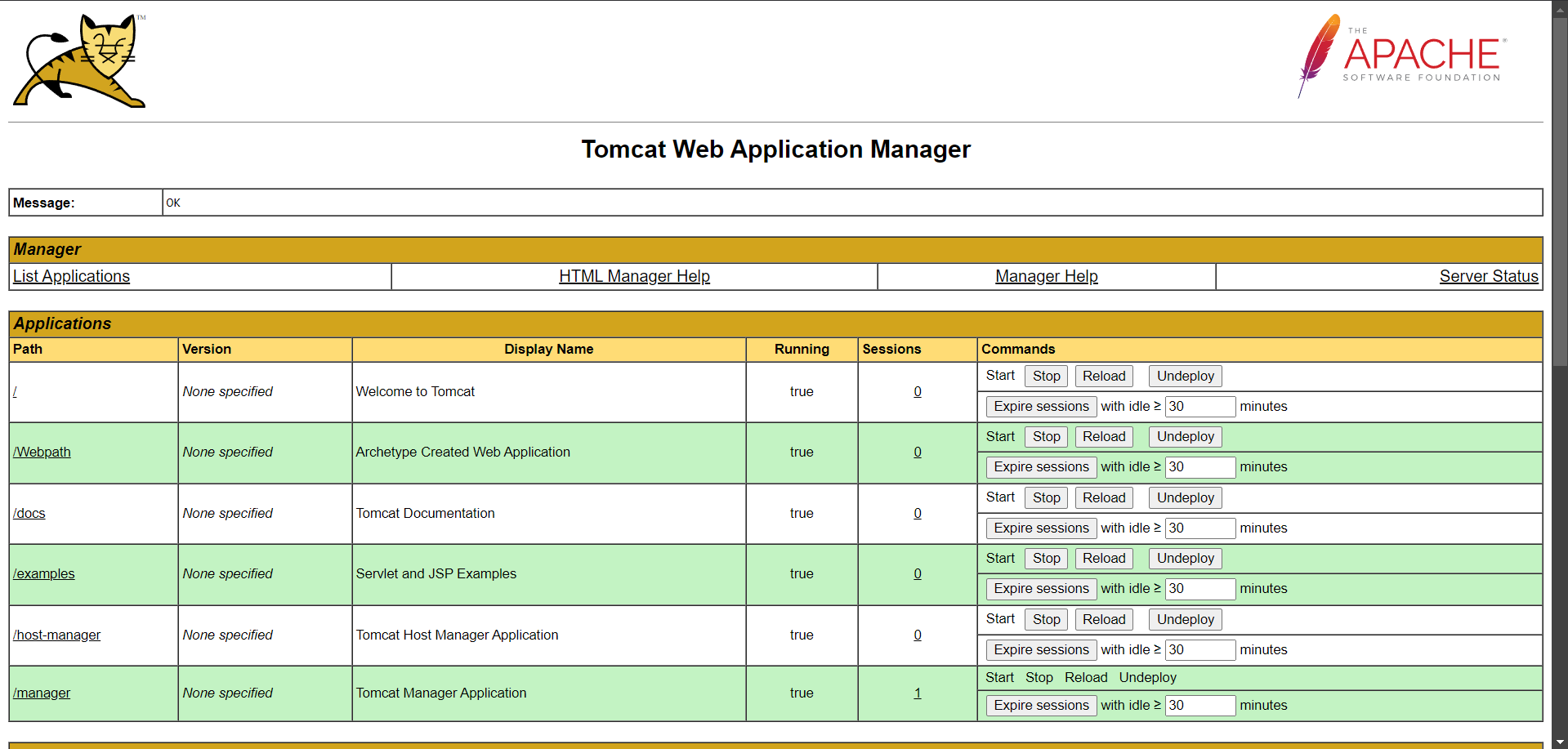


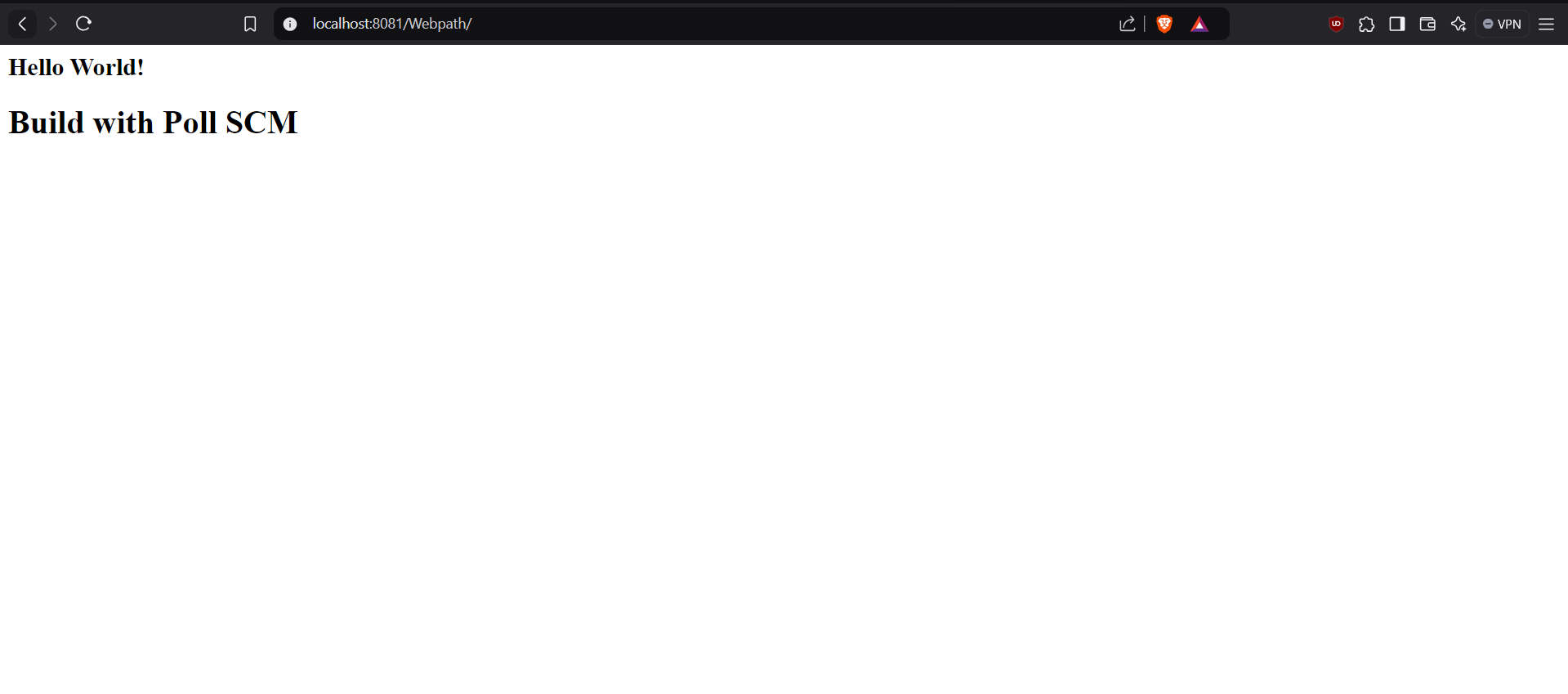
**Steps to create a Build Trigger for maven Web project using poll SCM:**

1. Select an existing maven web build project and go to configure.
2. Select poll SCM option from “BUILD TRIGGERS”.
3. Type “H/2 \* \* \* \*” in the description.
4. Click on Build now.



1. Open tomcat server and click on “/Webpath” in “manager app” and look for output.





1. You have successfully created a maven project with poll SCM.

**Result:**

Built CI/CD pipelines using Jenkins for Maven Java and web projects. You will have experience with both freestyle and scripted pipelines, and you will understand how to configure Poll SCM to automate the build process based on repository changes. This will enhance your skills in CI/CD automation, pipeline configuration, and continuous delivery practices.