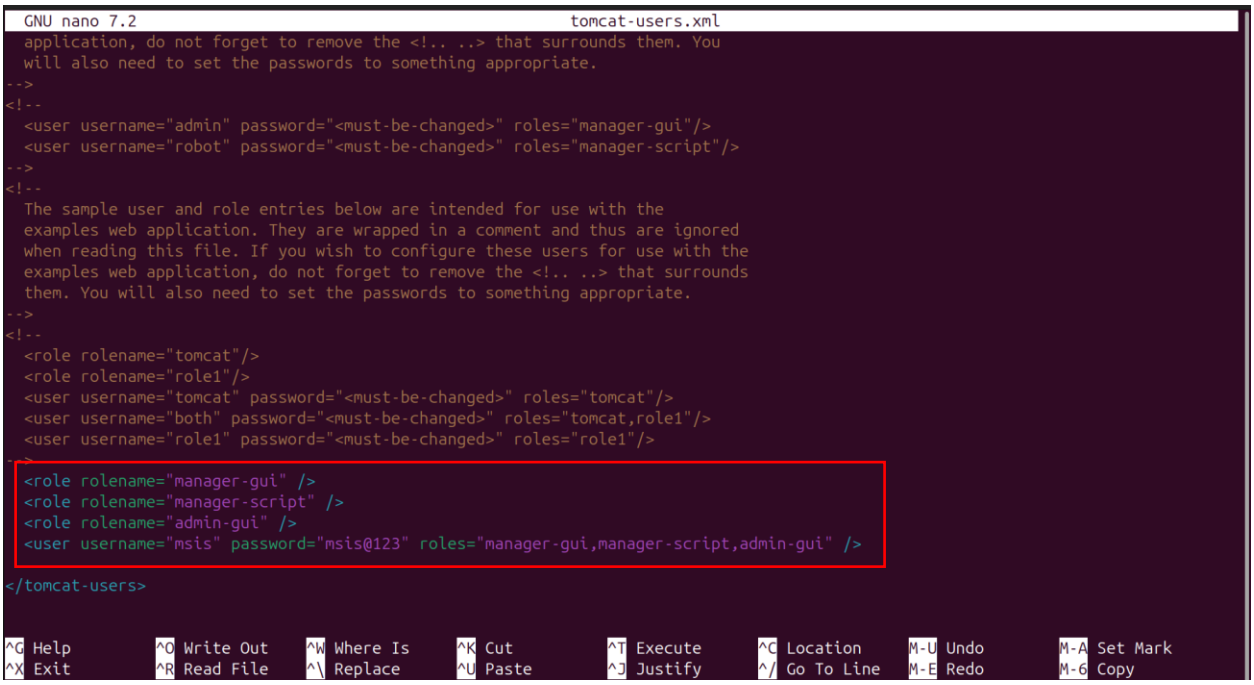
For Java

1.Implement CI/CD Build Pipeline for Java Application on Jenkins Server

Tomcat Installation reference link: https://docs.vultr.com/how-to-install-apache-tomcat-on-ubuntu-24-04

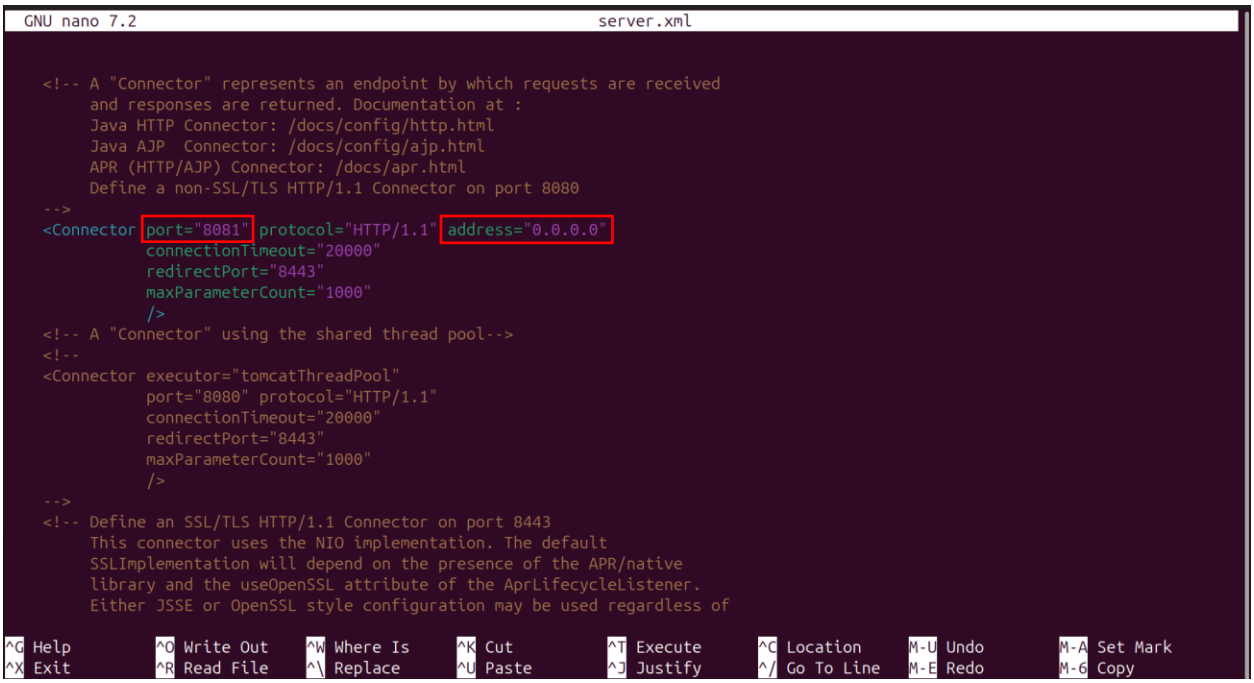
cd /opt/tomcat/conf

Sudo nano tomcat-users.xml



cd /opt/tomcat/conf

sudo nano server.xml



Plugins required:

- Maven Integration plugin

- Warnings: For Code review

- JUnit: For Testing (if required)

- Deploy to container

The above plugins can be installed by navigating to Manage Jenkins->Plugins.

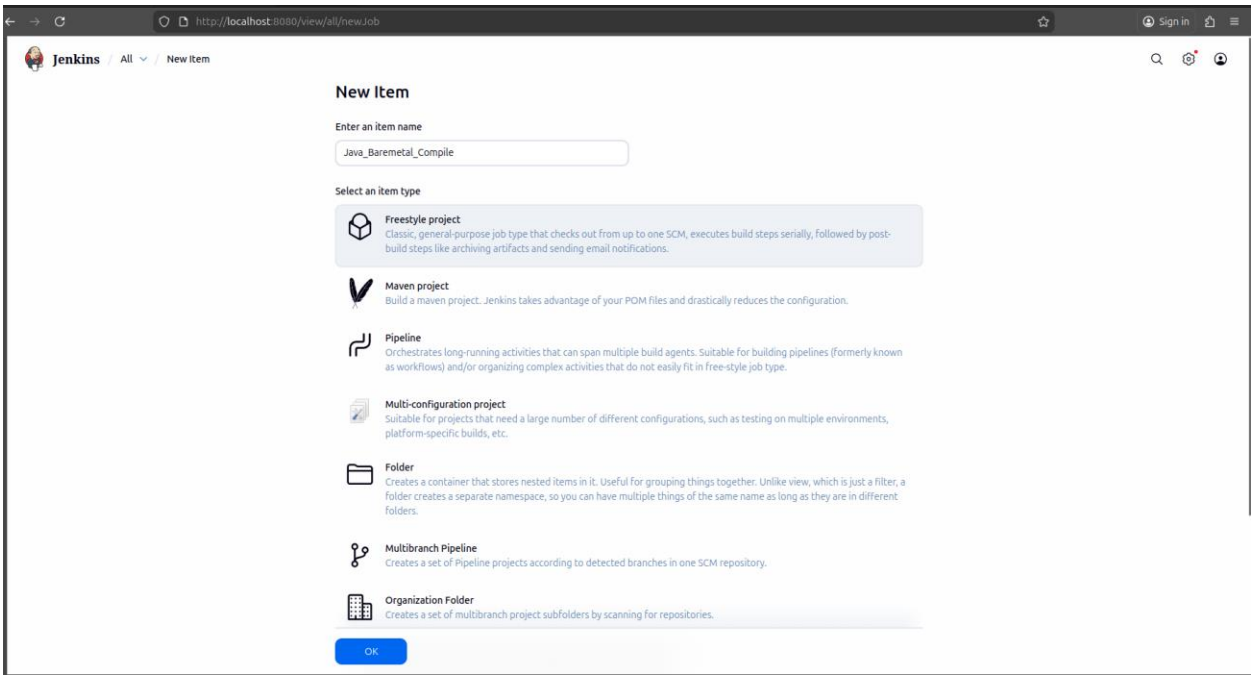
GitHub Repo: https://github.com/sreepathysois/java-tomcat-maven-example.git

Log in to Jenkins Server.

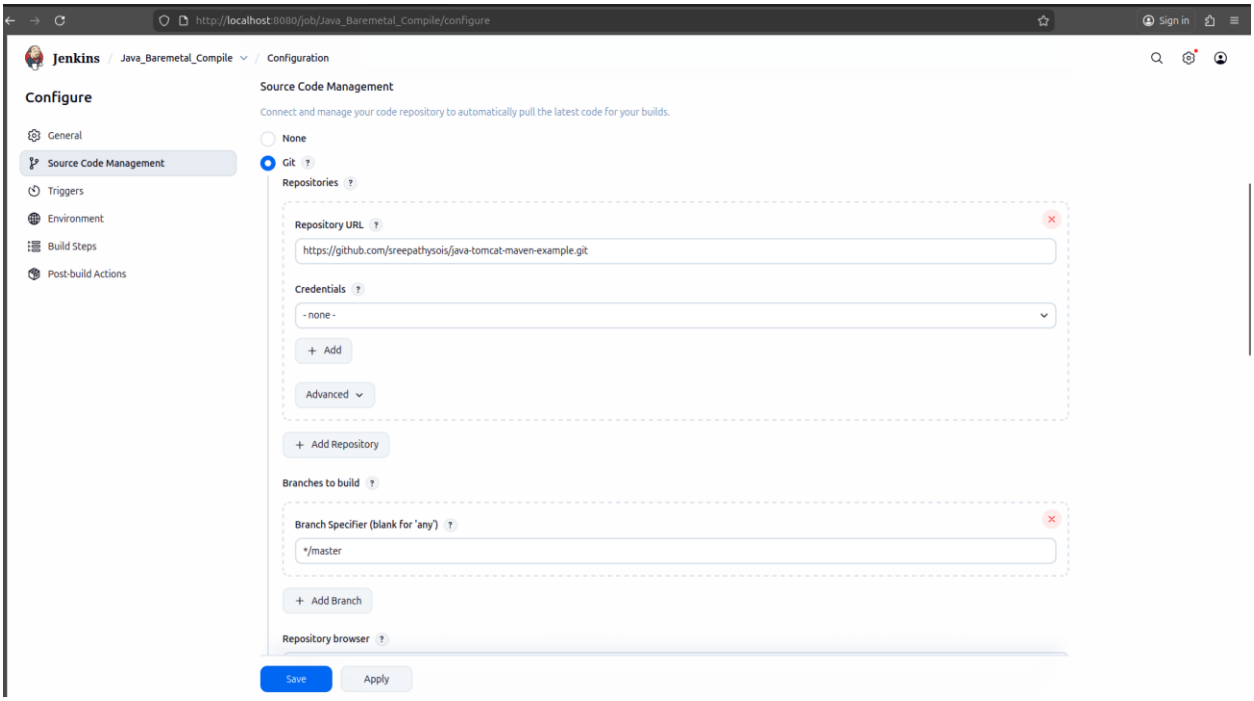
Step1: Compile

Create a Free Style Jenkins Project.

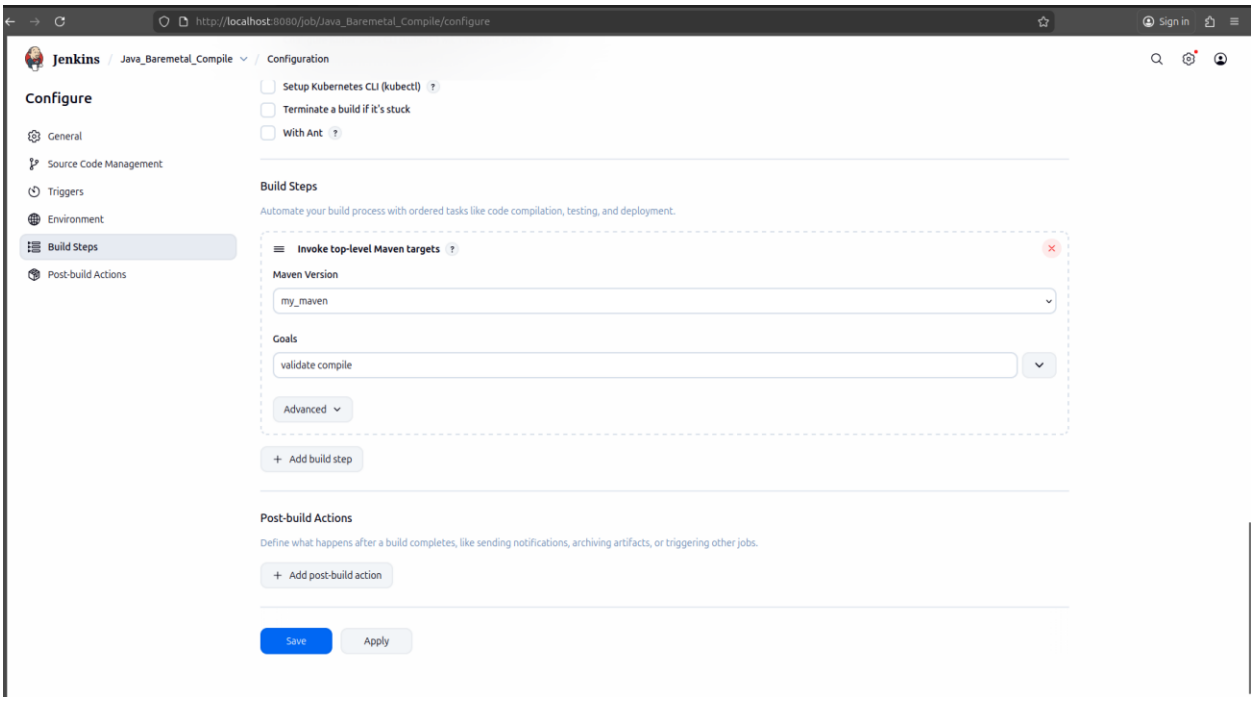
Provide a name to Project for Example: Java_Baremetal_compile then click Ok.



On the next page, select Source Code Management as Git, then provide the GitHub repository URL under Repository URL and specify the branch.

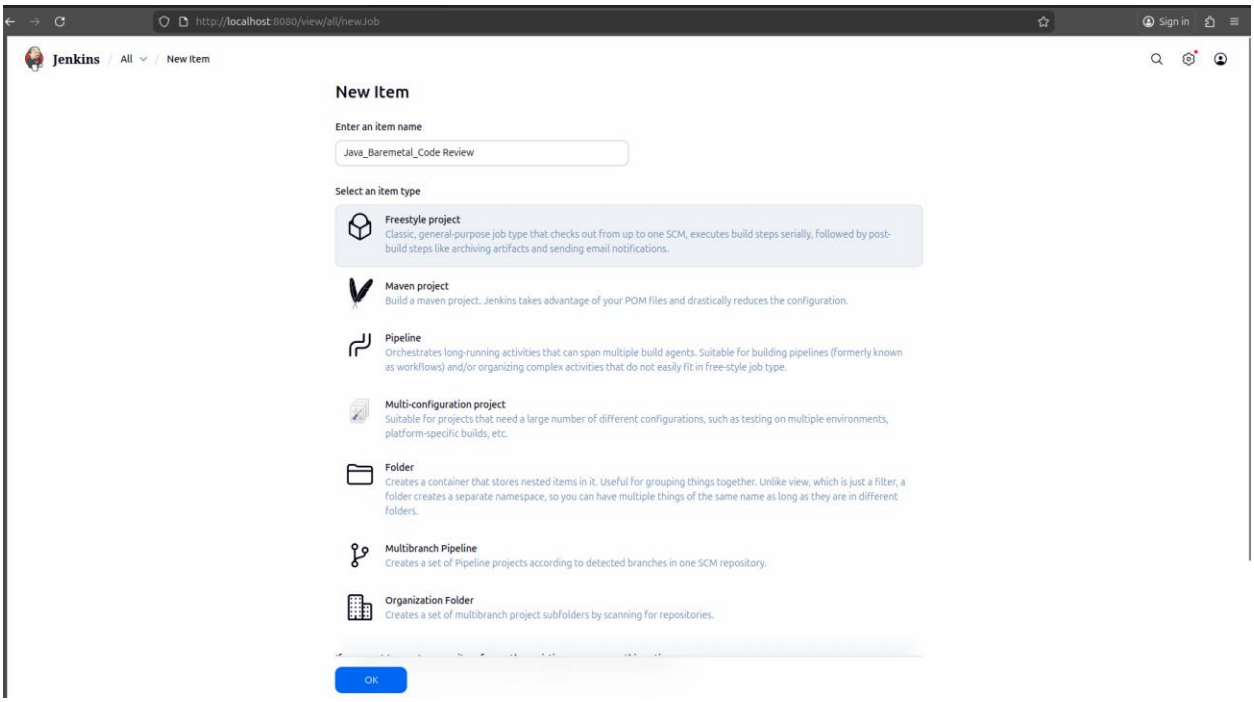


Under the Build step, select “Invoke Top-Level Maven Targets,” and under Goals, enter validate compile, then click Save.

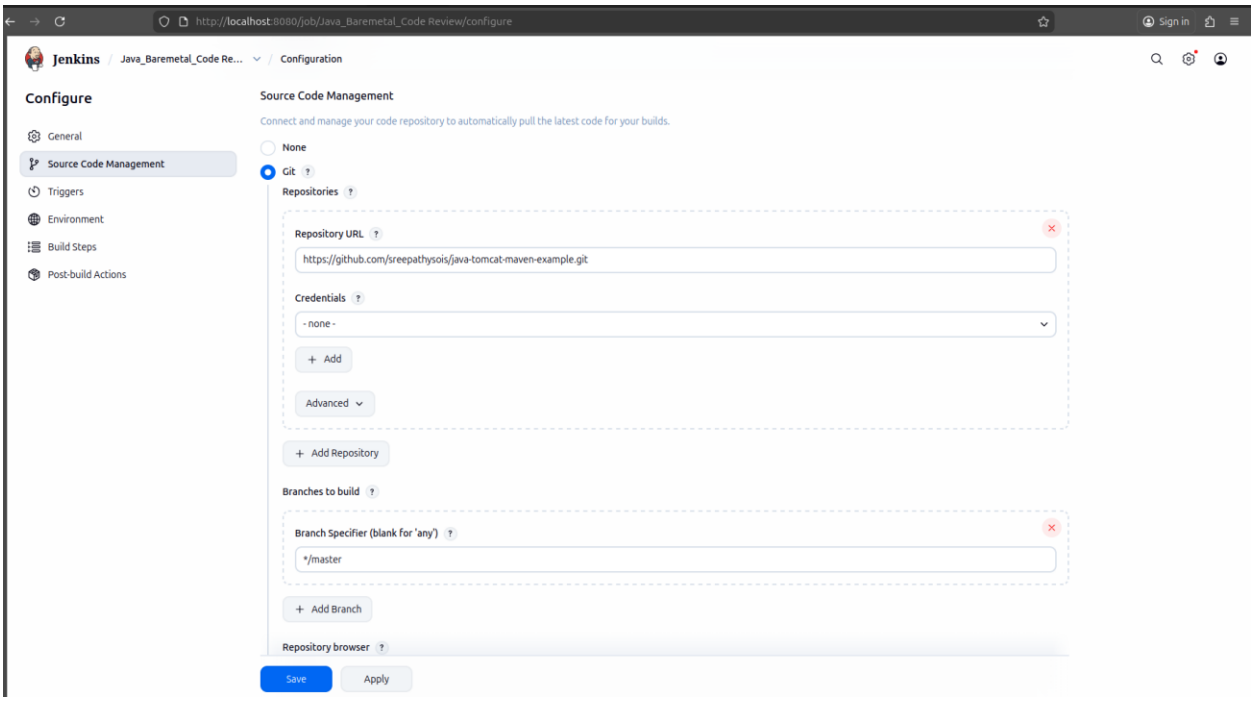


Step2: Code review

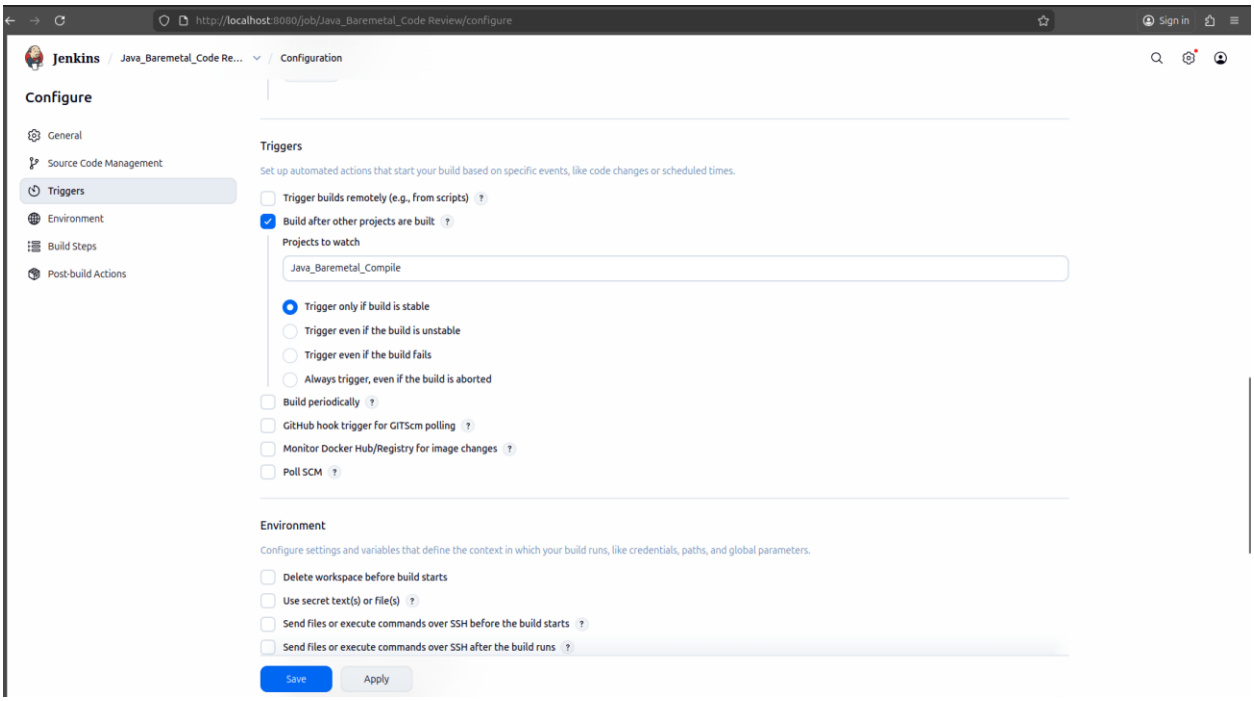
Create a Freestyle Jenkins project. Provide a name for the project, for example: Java_Baremetal_CodeReview, then click OK.



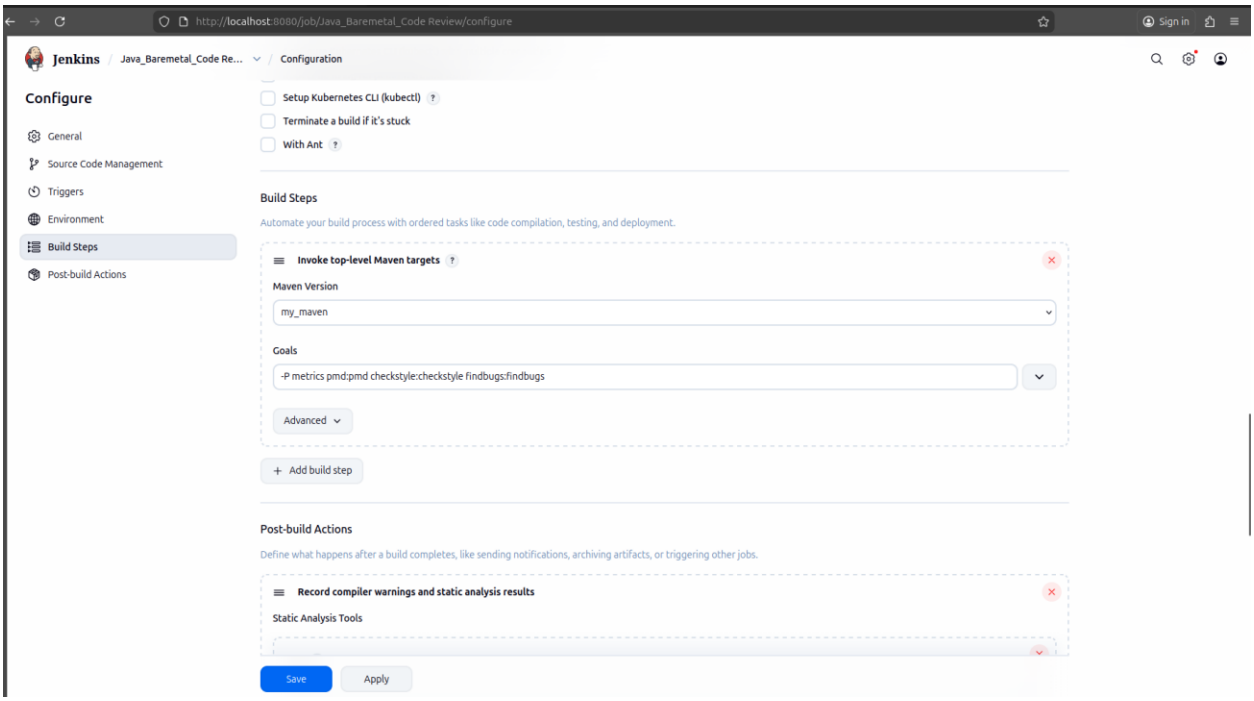
On the next page, select Source Code Management as Git, then provide the GitHub repository URL under Repository URL and specify the branch.



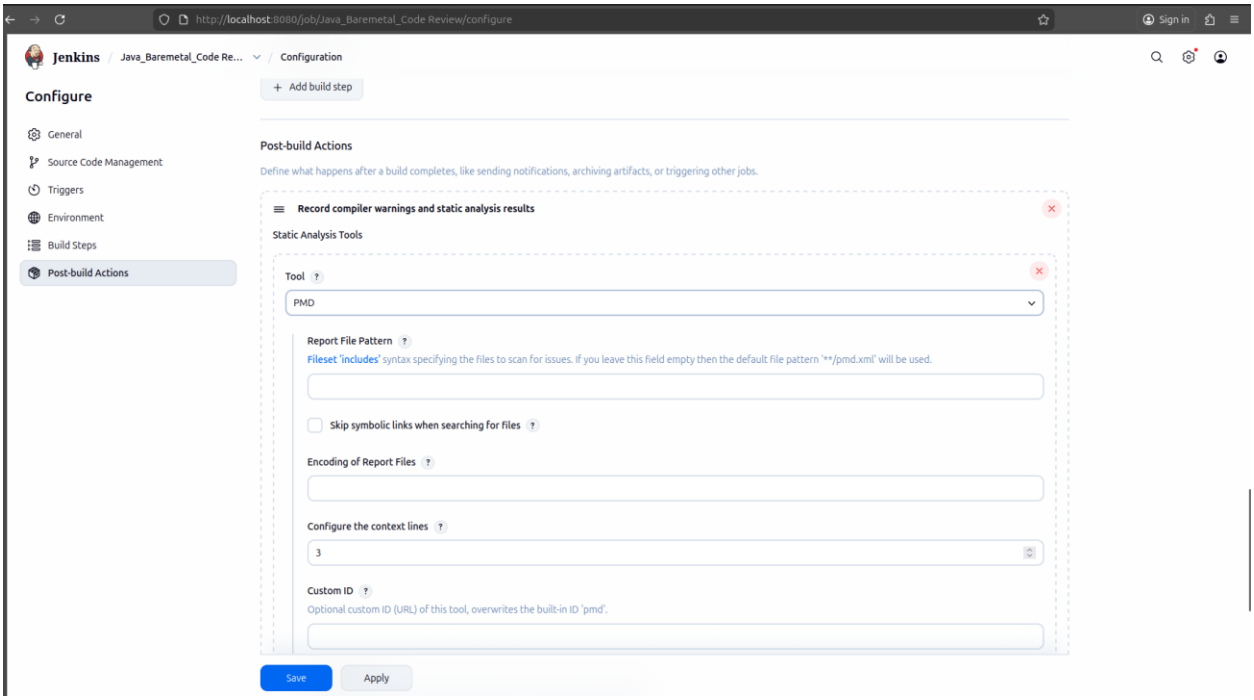
Under Triggers, select “Build after other projects are built,” and under Projects to watch, enter the previous item name (Java_Baremetal_Compile). Then select “Trigger only if build is stable.”

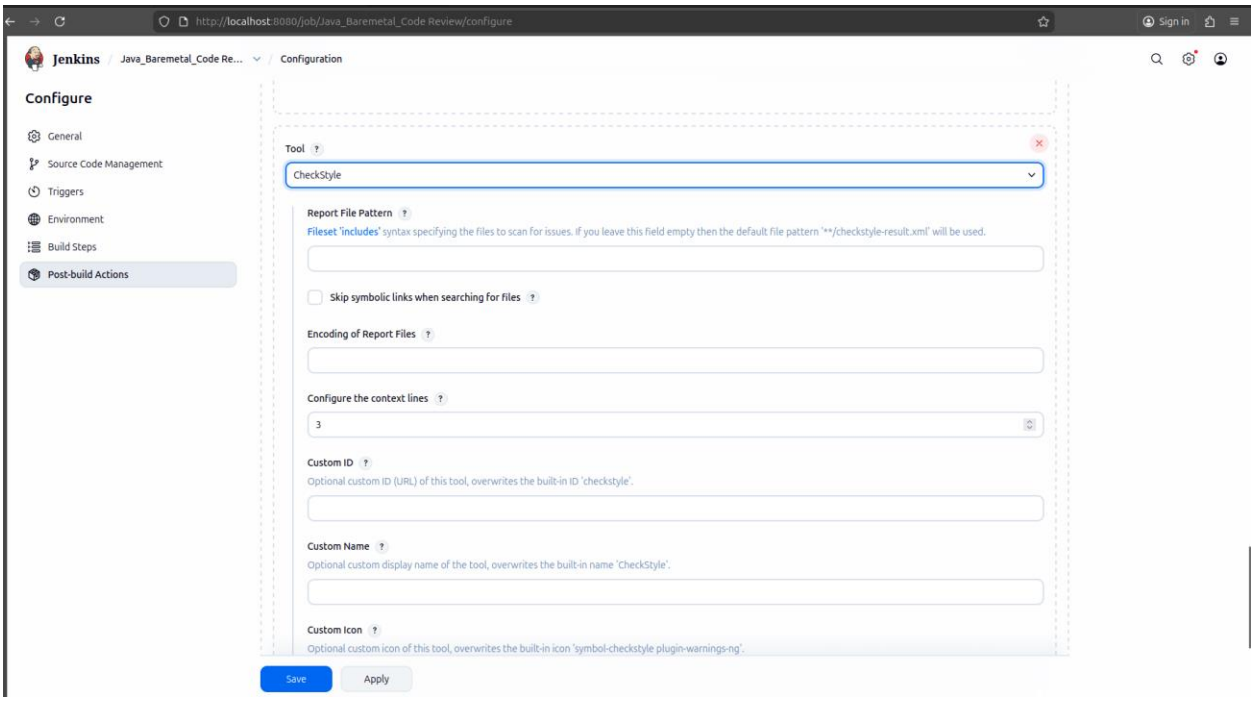


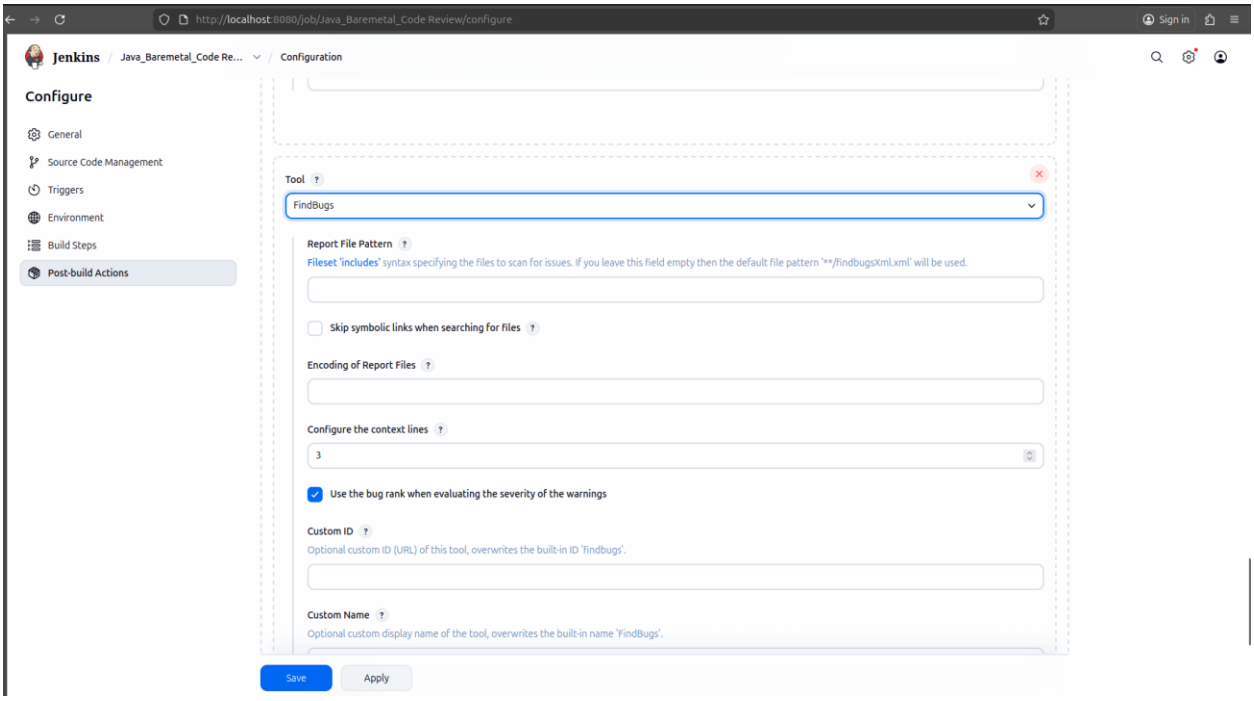
Under the Build step, select “Invoke Top-Level Maven Targets,” and under Goals, enter -P metrics pmd:pmd checkstyle:checkstyle findbugs:findbugs, then click Save.

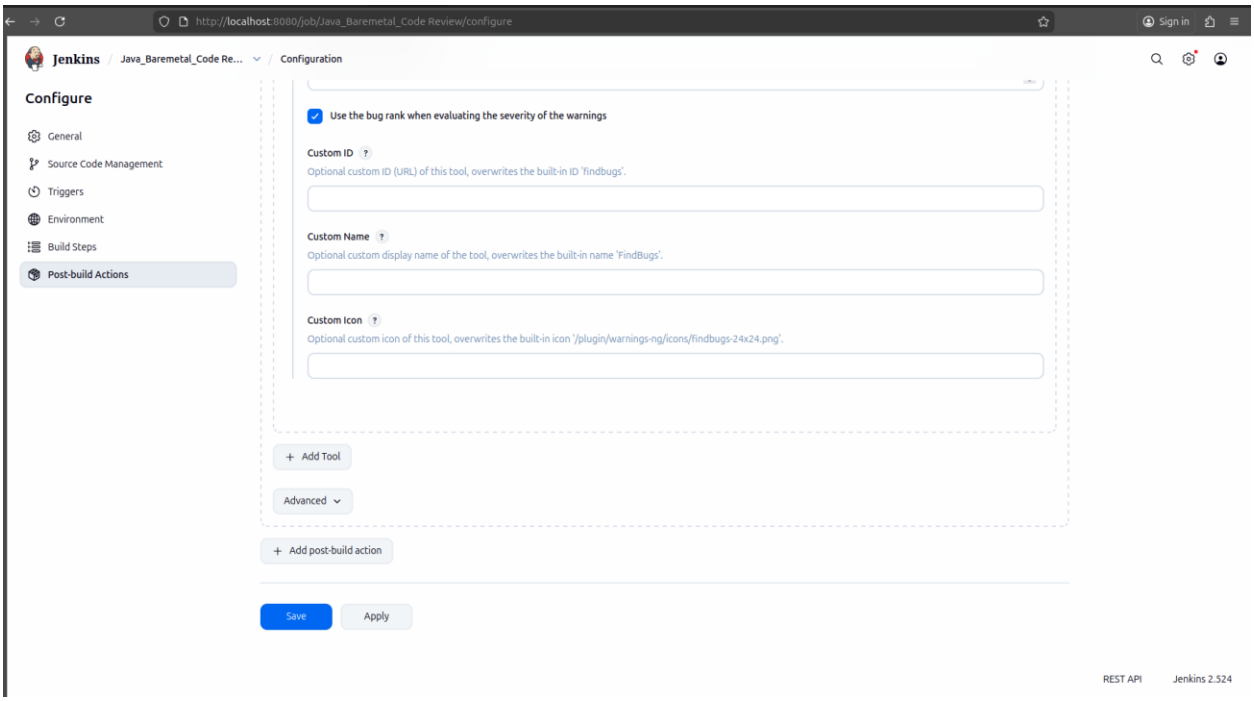


Under post-build, select “Record compiler warnings and static analysis results.” Then select PMD, CheckStyle, and FindBugs, and click Save.



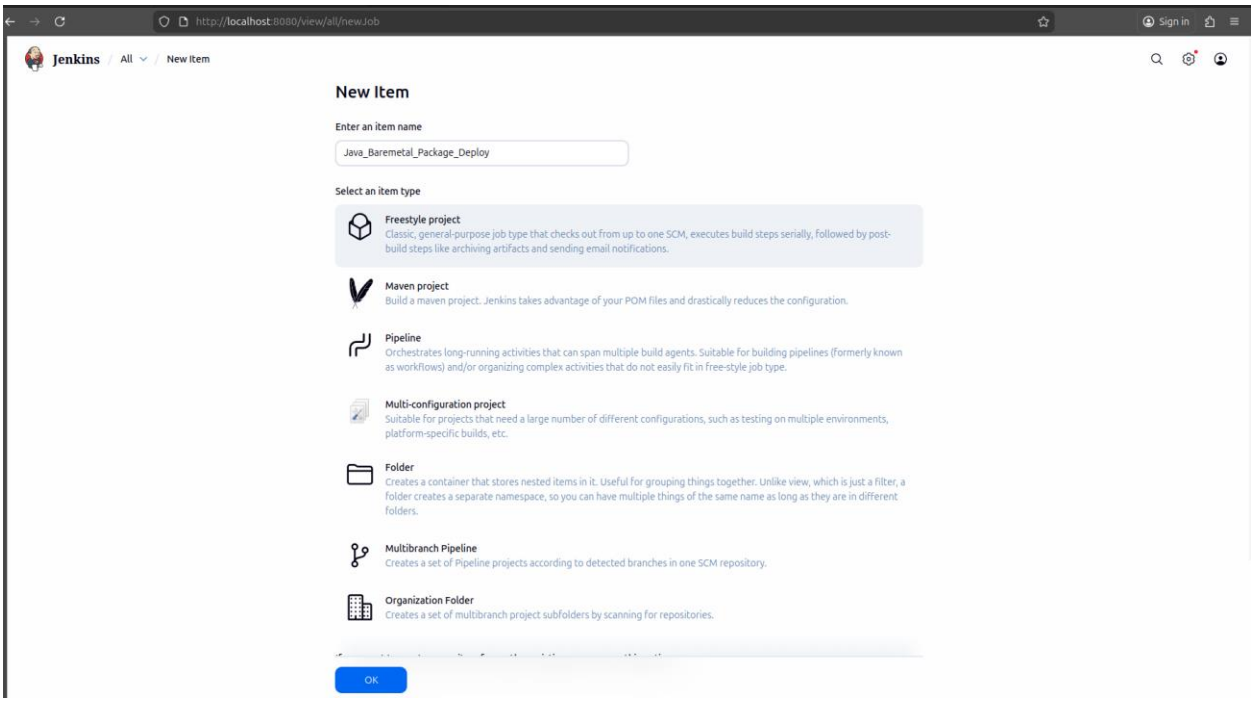

http://localhost:8080/job/Java_Baremetal_Code Review/configure
Sign in
Jenkins
Java_Baremetal_Code Re...
Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Tool
CheckStyle
Report File Pattern
Fileset 'includes' syntax specifying the files to scan for issues. If you leave this field empty then the default file pattern '**/checkstyle-result.xml' will be used.
Skip symbolic links when searching for files
Encoding of Report Files
Configure the context lines
3
Custom ID
Optional custom ID (URL) of this tool, overwrites the built-in ID 'checkstyle'.
Custom Name
Optional custom display name of the tool, overwrites the built-in name 'CheckStyle'.
Custom Icon
Optional custom icon of this tool, overwrites the built-in icon 'symbol-checkstyle plugin-warnings-ng'.
Save
Apply


http://localhost:8080/job/Java_Baremetal_Code Review/configure
Sign in
Jenkins
Java_Baremetal_Code Re...
Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Tool
FindBugs
Report File Pattern
Fileset 'includes' syntax specifying the files to scan for issues. If you leave this field empty then the default file pattern '**/findbugsXml.xml' will be used.
Skip symbolic links when searching for files
Encoding of Report Files
Configure the context lines
3
Use the bug rank when evaluating the severity of the warnings
Custom ID
Optional custom ID (URL) of this tool, overwrites the built-in ID 'findbugs'.
Custom Name
Optional custom display name of the tool, overwrites the built-in name 'FindBugs'.
Save
Apply

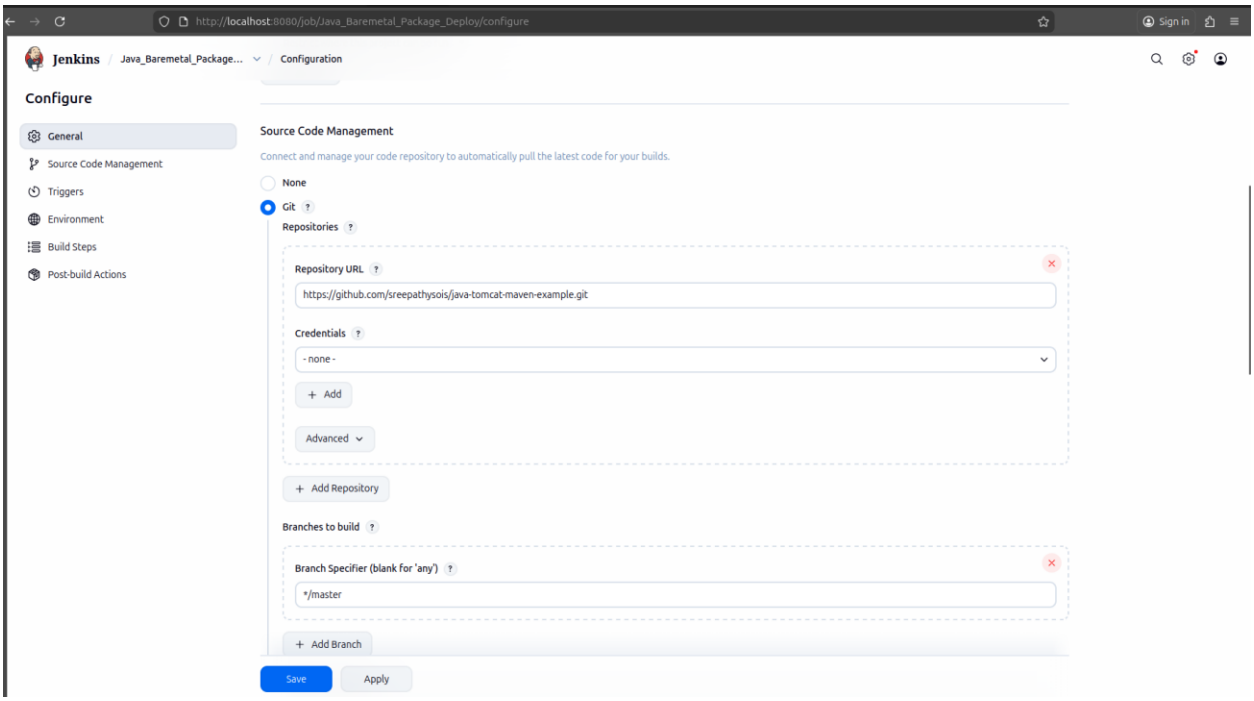


Step3: Package and Deploy

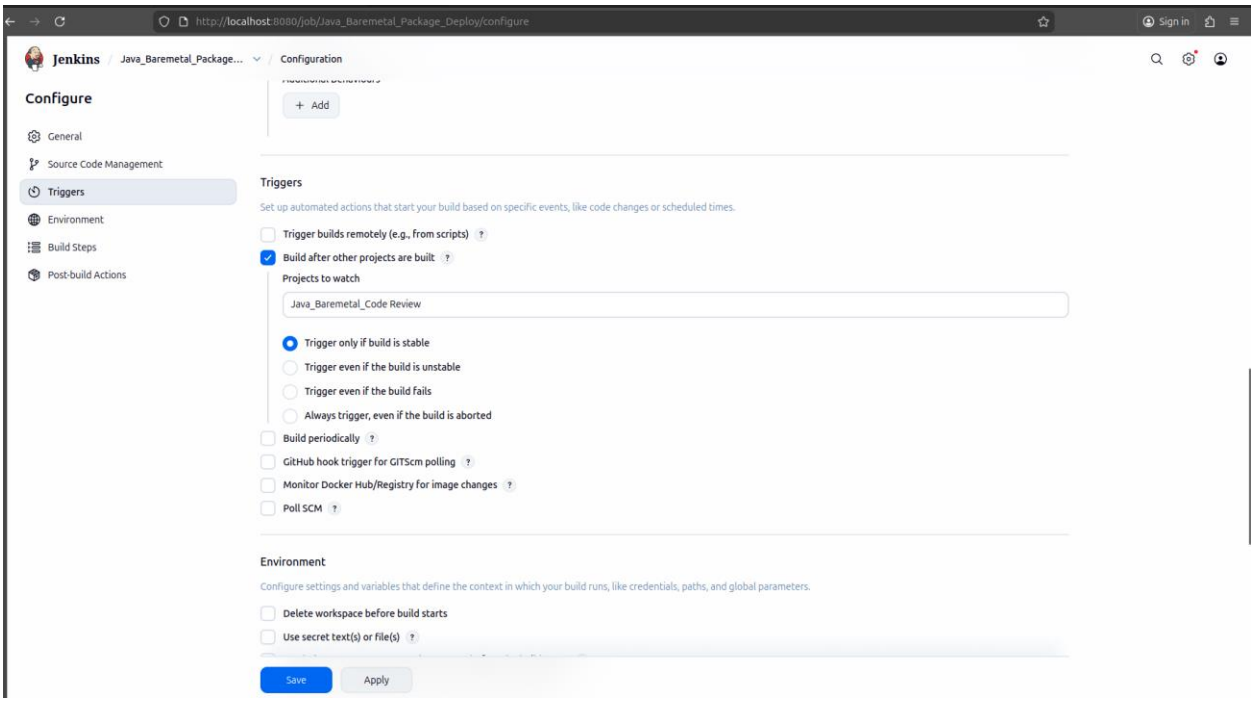
Create a **Freestyle Jenkins Project** and provide a name for the project, for example **Java_Baremetal_Package_Deploy**, then click **OK**.



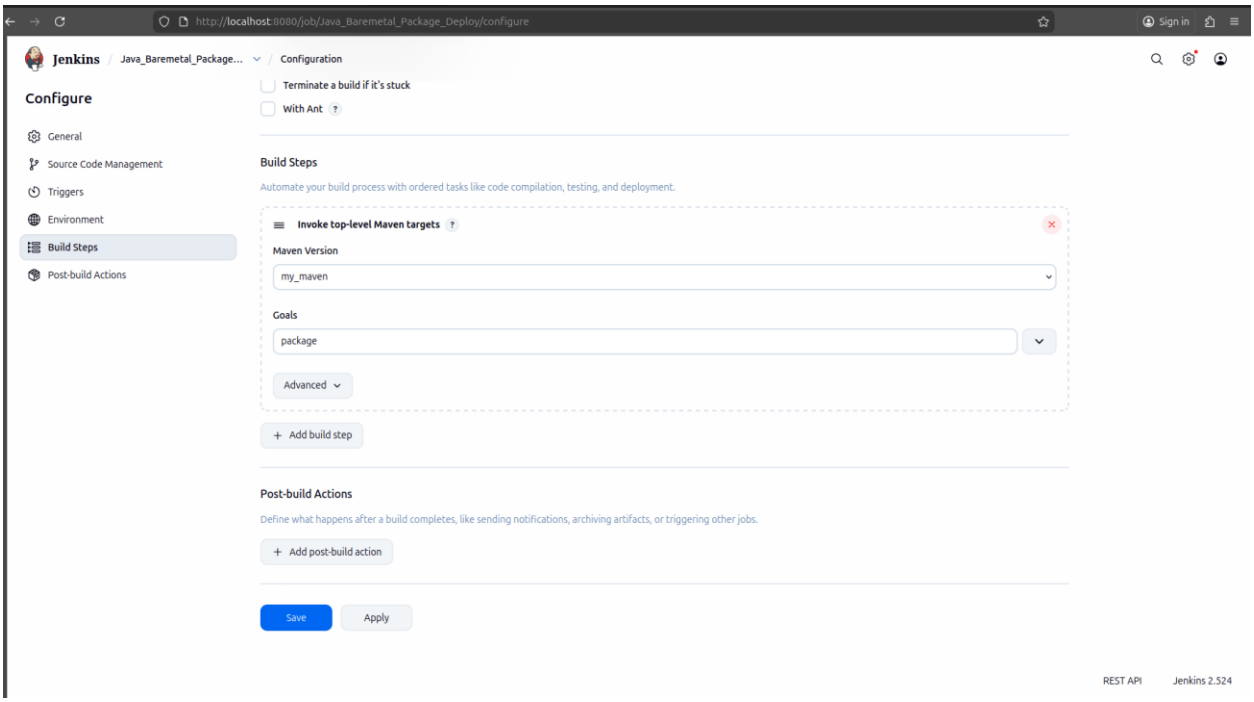
On the next page, select **Git** under **Source Code Management**, then enter the GitHub repository URL in the **Repository URL** field and specify the branch.



Under **Triggers**, select **“Build after other projects are built.”** In the **Projects to watch** field, enter the name of the previous job (**Java_Baremetal_codeReview**). Then select **“Trigger only if build is stable.”**

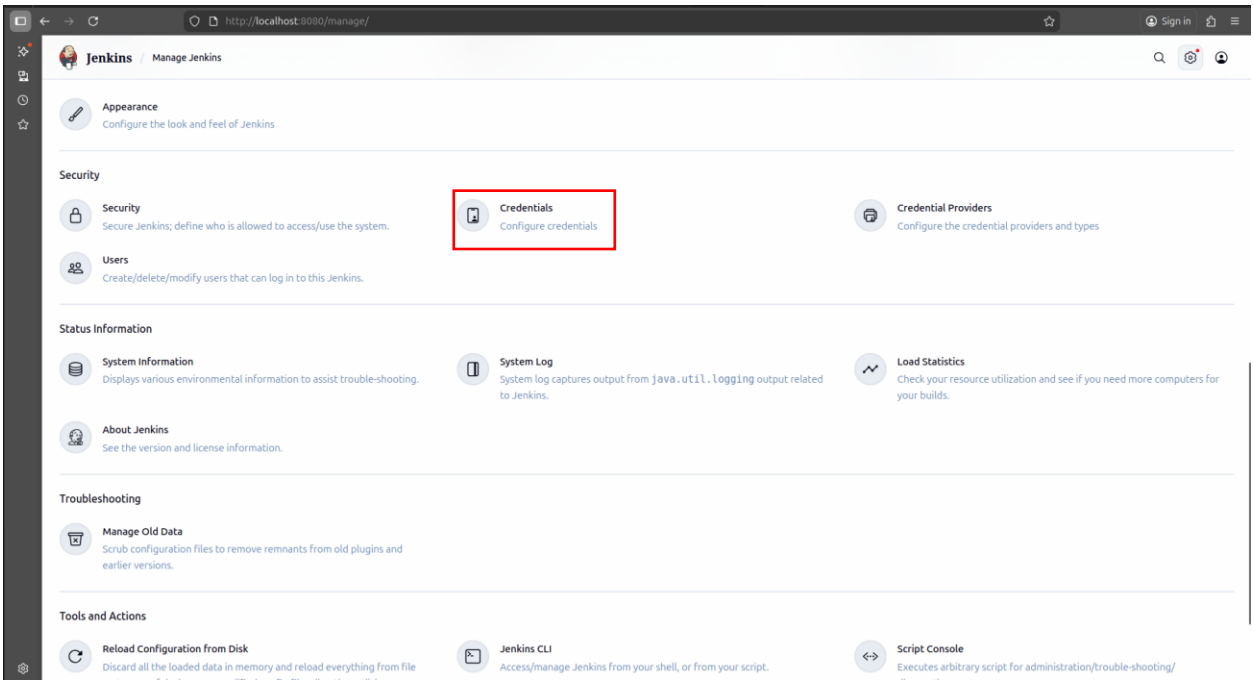


Under the **Build** step, select **Invoke Top-Level Maven Targets**, and in the **Goals** field, enter package. Then click **Save**.

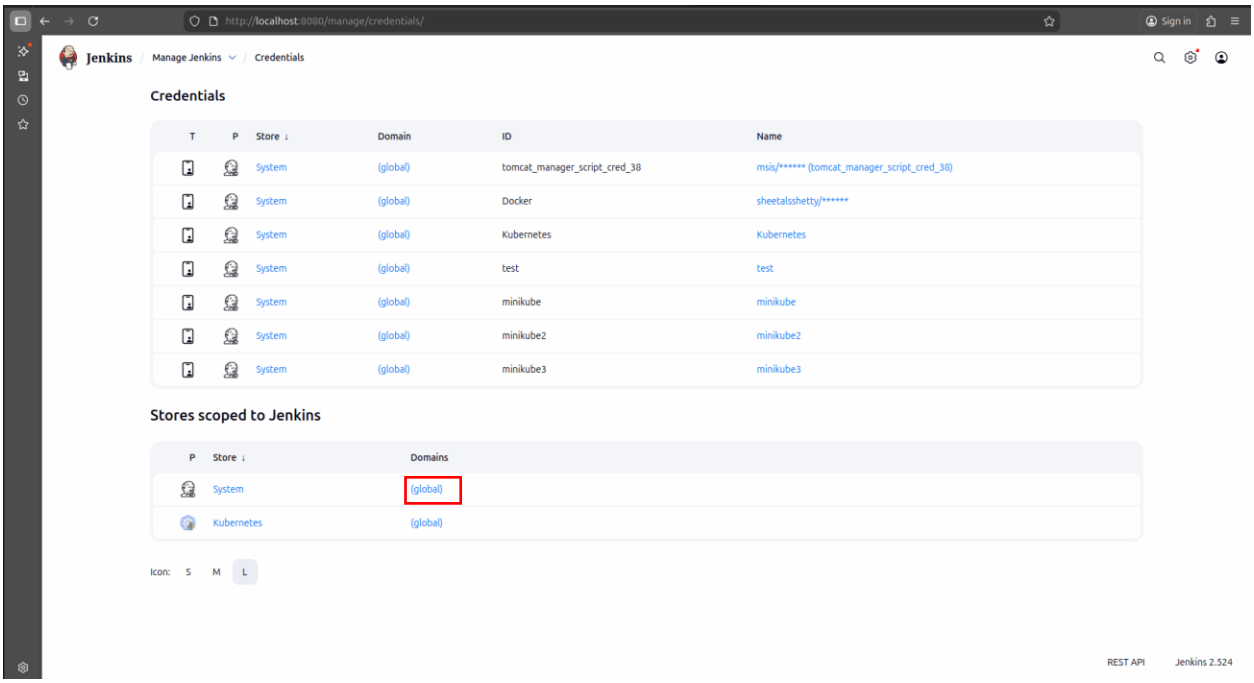


Under **Post-build Actions**, select **“Deploy to Container.”**

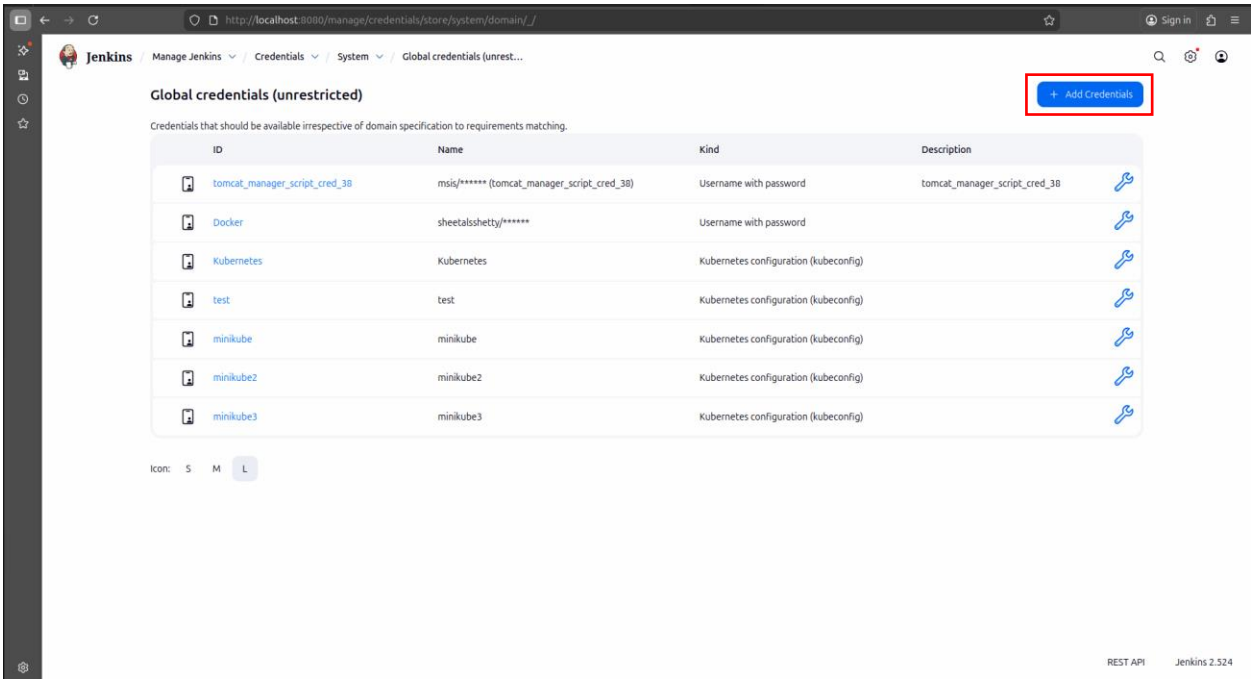
To configure Tomcat credentials in Jenkins, go to **Manage Jenkins → Credentials**.



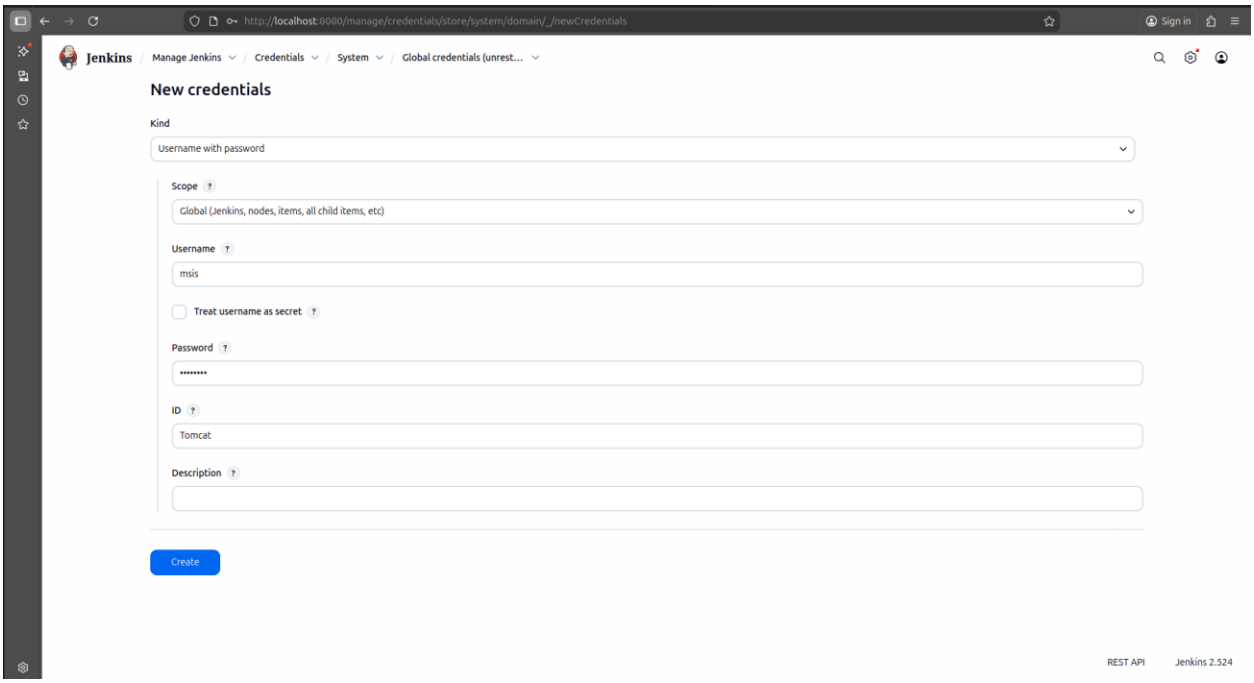
Click on Global.



Click on Add Credential



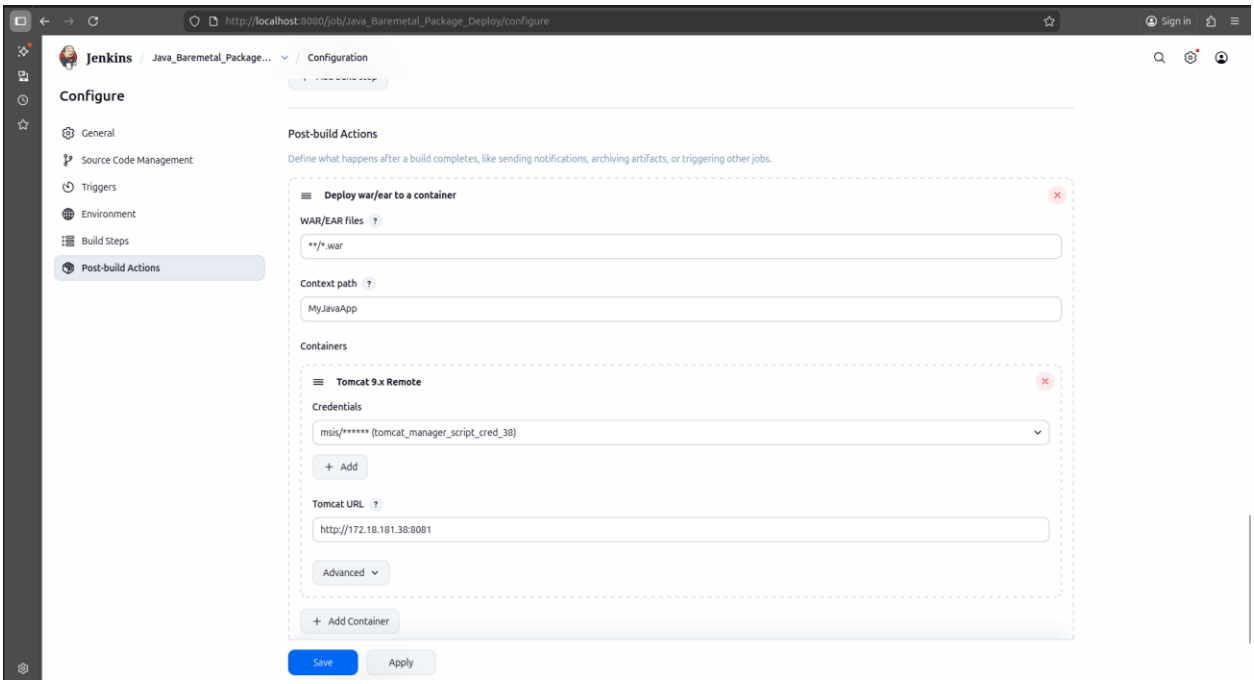
Under **Kind**, select **“Username with Password.”** Then enter the username and password that were configured in the tomcat-users.xml file.



Go back to the Jenkins item **“Java_Baremetal_Package_Deploy.”** Under **Deploy to container**, enter **/*.war in the **WAR/EAR Files** field, provide a name in the **Context path** (e.g., MyJavaApp), and select **Tomcat 9.x Remote** as the container.

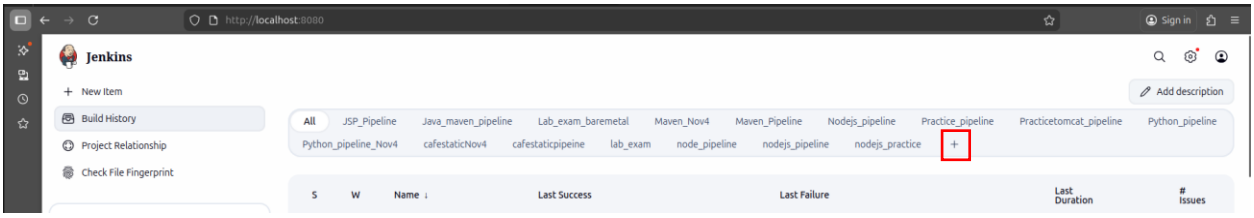
Choose the appropriate **Tomcat credential ID**, and in the **Tomcat URL** field, enter http://<IP>:<Tomcat_Port>.

Then click **Save**.

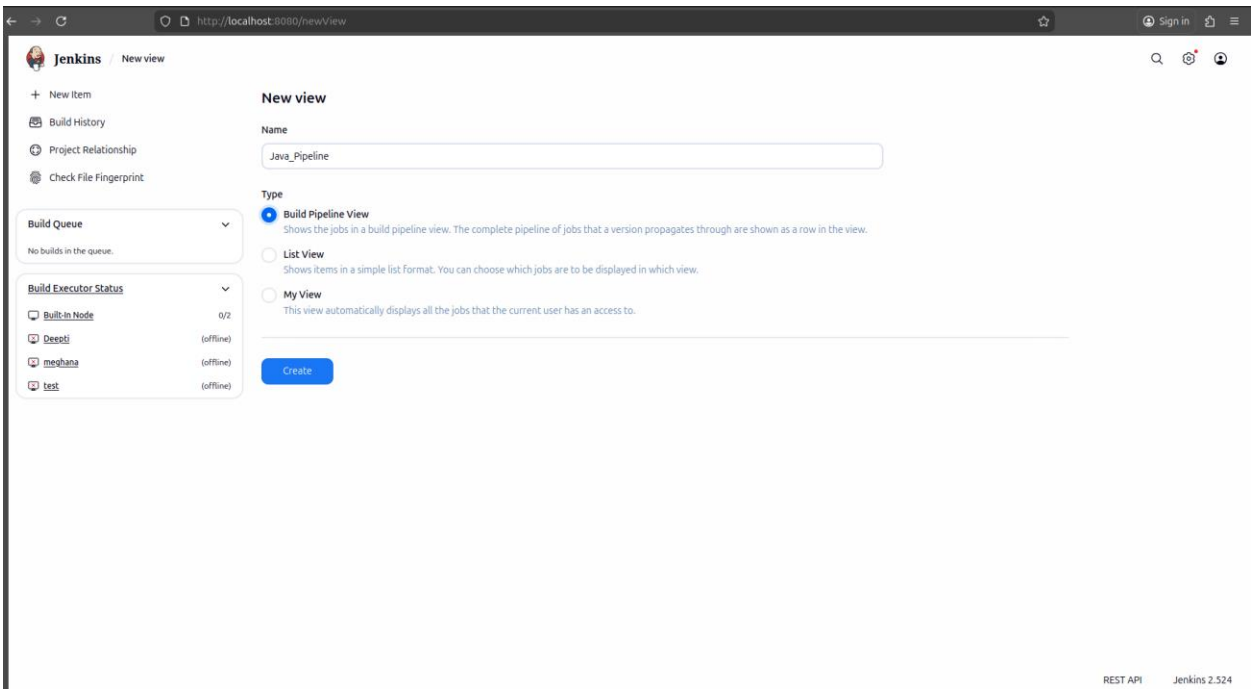


Now we will create the **Build Pipeline View**.

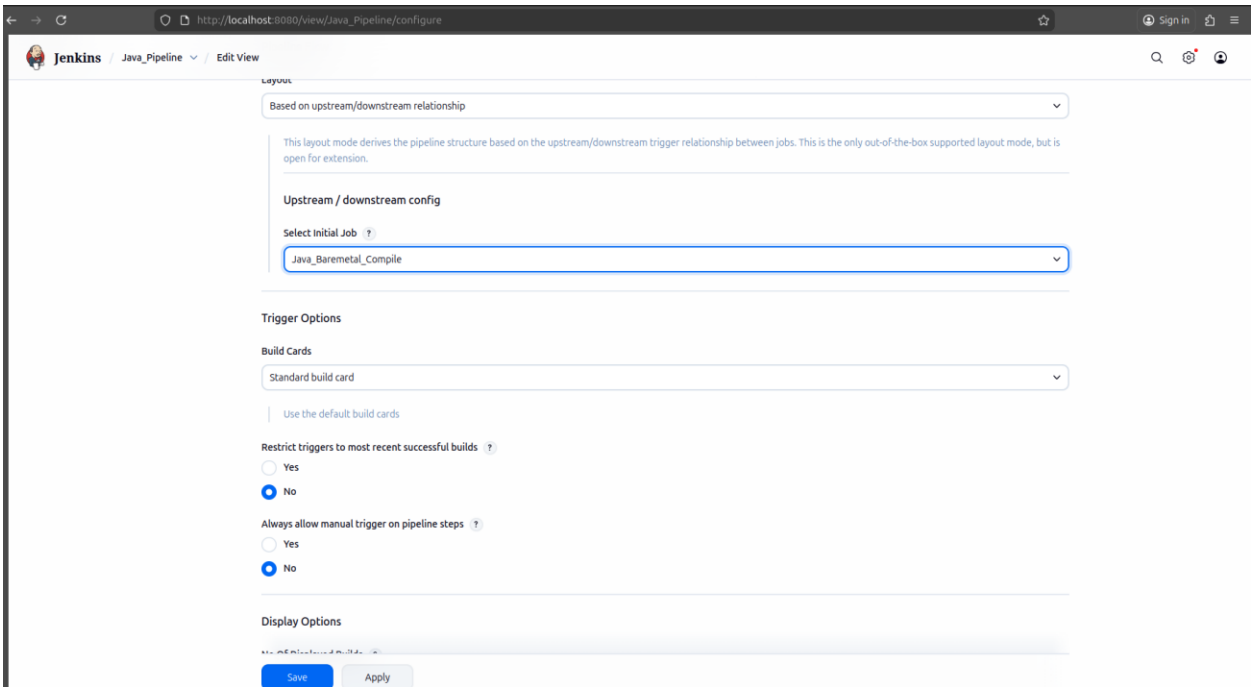
Navigate to the Jenkins home page and click on the **“+”** icon.



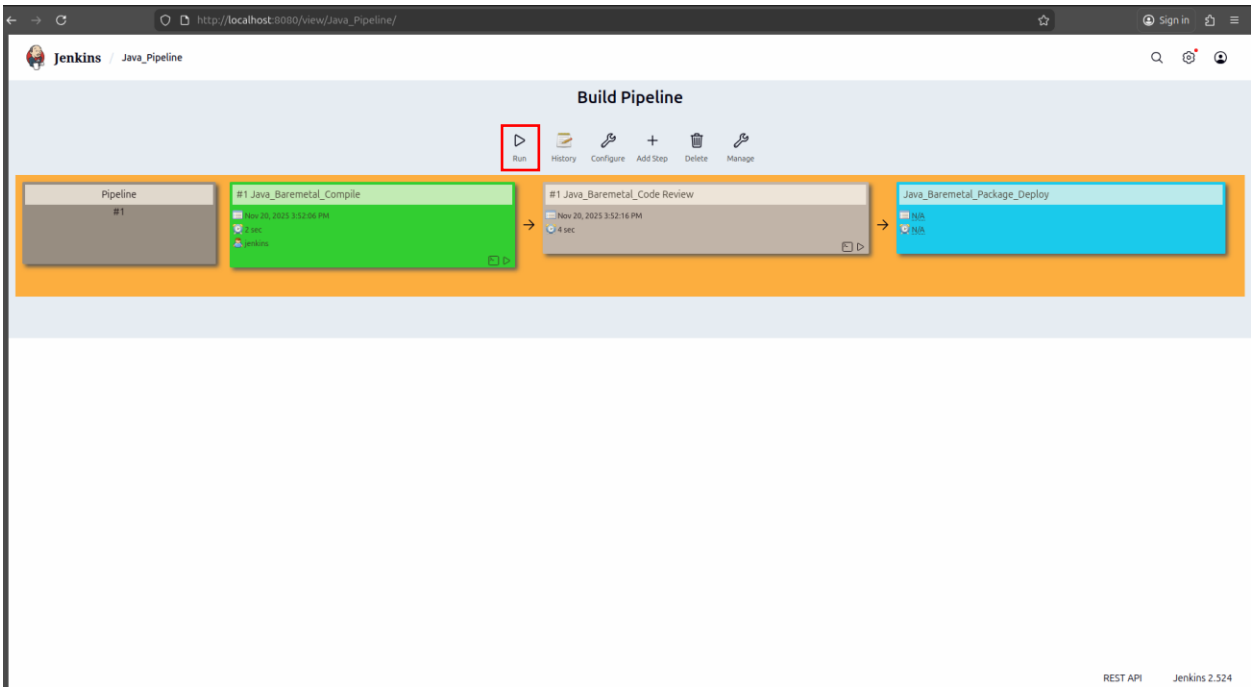
Provide a pipeline name (e.g., **Java_Pipeline**), select **Build Pipeline View** under **Type**, and then click **Create**.

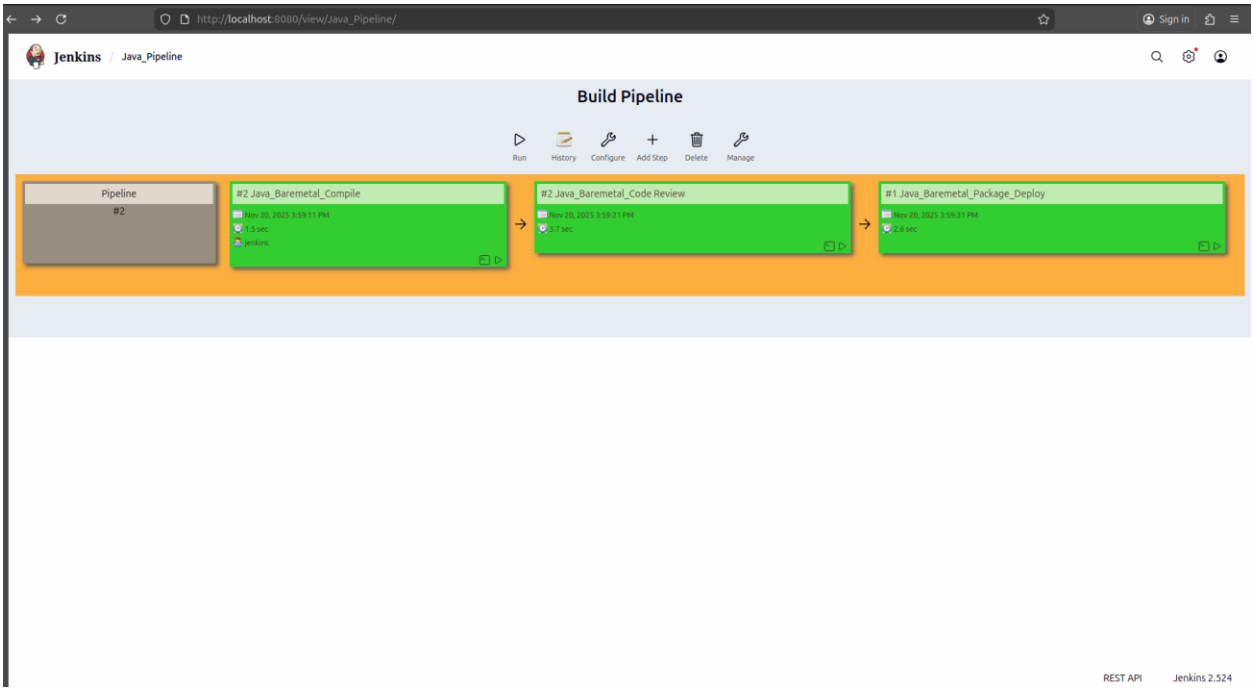


On the next page, under the **Upstream/Downstream config** section, enter the first job name (**Java_Baremetal_compile**) in the **Select Initial Job** field, then click **Save**.



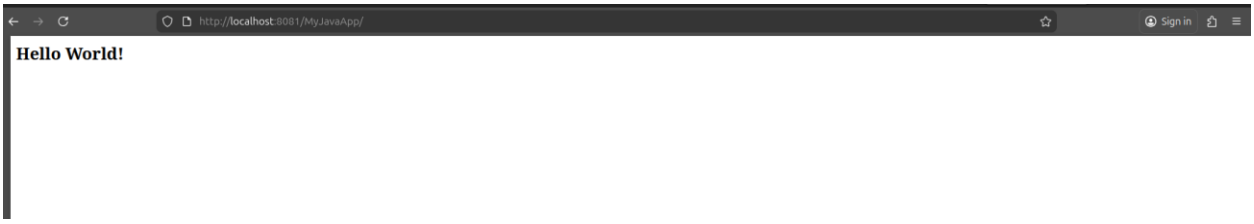
Now, you can see Pipeline with all the items. Click on Run button





Once, the Pipeline runs successfully we can access the page at

http://localhost:<Tomcat Port>/<Context Path> (for e.g http://localhost:8081/ MyJavaApp)



****

2: Containerizing the Java Application using Docker

Plugins: Install Docker relevant plugins

Below is the Dockerfile for Java tomcat application

#Stage 1: Build WAR using Maven

FROM maven:3.9-eclipse-temurin-17 AS build WORKDIR /app COPY . . RUN mvn clean package -DskipTests

#Stage 2: Deploy to Tomcat

FROM tomcat:9.0-jdk17-temurin

RUN rm -rf /usr/local/tomcat/webapps/*

COPY --from=build /app/target/*.war /usr/local/tomcat/webapps/ROOT.war

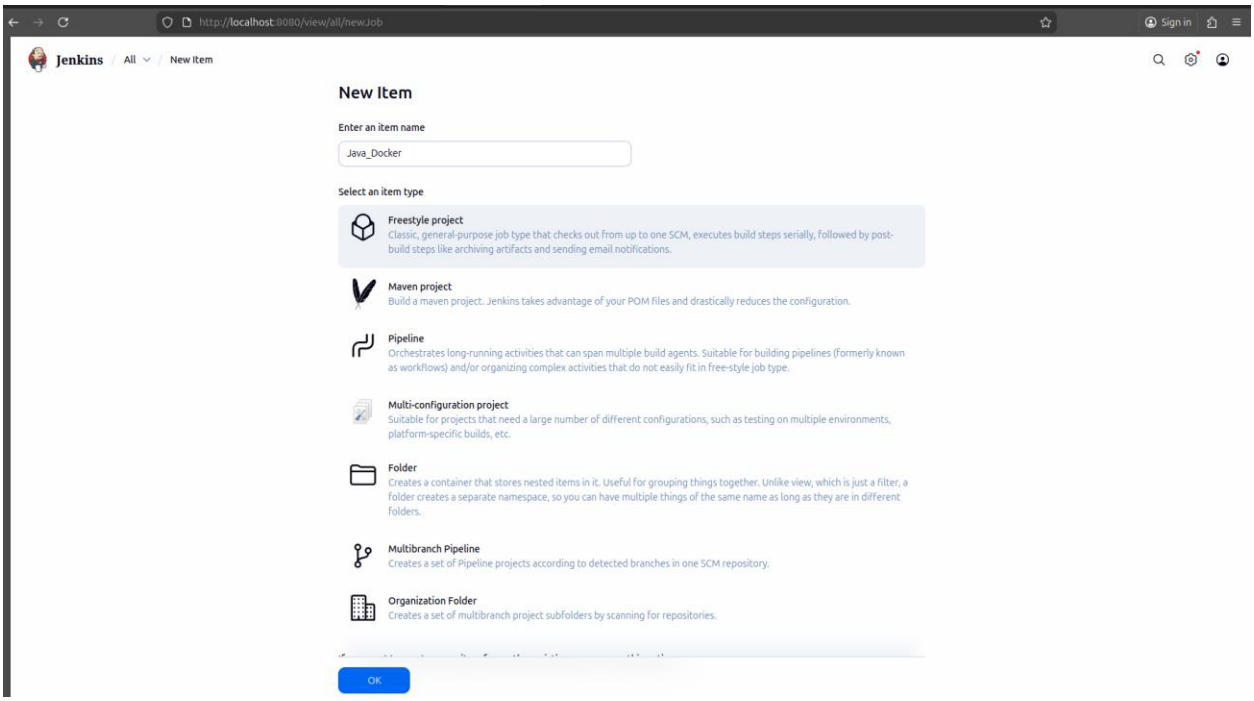
EXPOSE 8080

CMD ["catalina.sh", "run"]

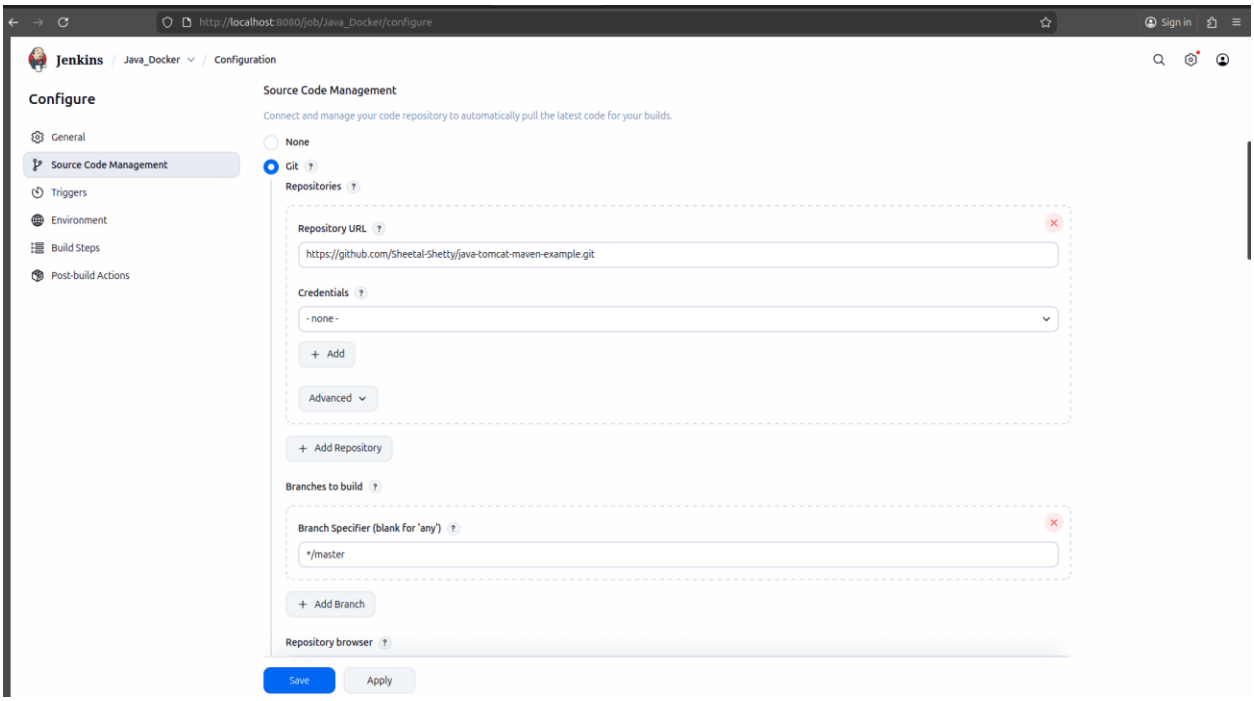
Pushed sir's code along with the Dockerfile to the new repository.

GitHub Repo: https://github.com/Sheetal-Shetty/java-tomcat-maven-example.git

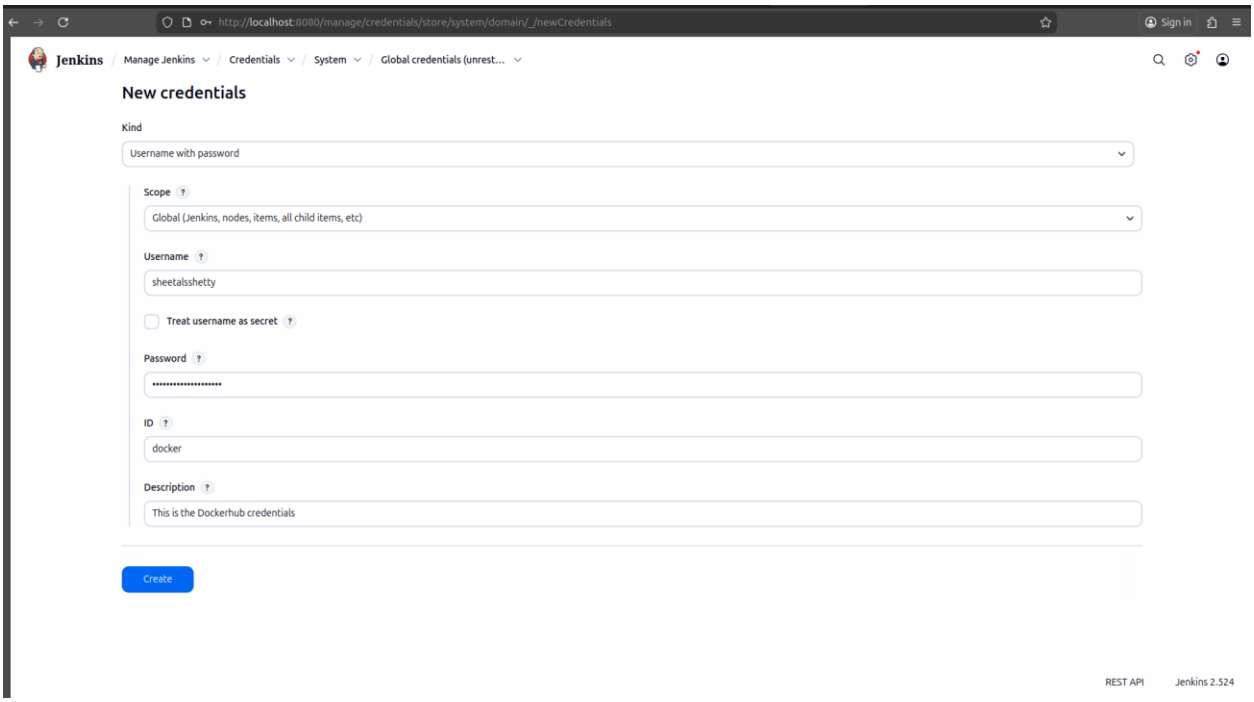
Create a **Freestyle Jenkins Project** and provide a name for the project, for example **Java_Docker**, then click **OK**.



On the next page, select **Git** under **Source Code Management**, then enter the GitHub repository URL in the **Repository URL** field and specify the branch.



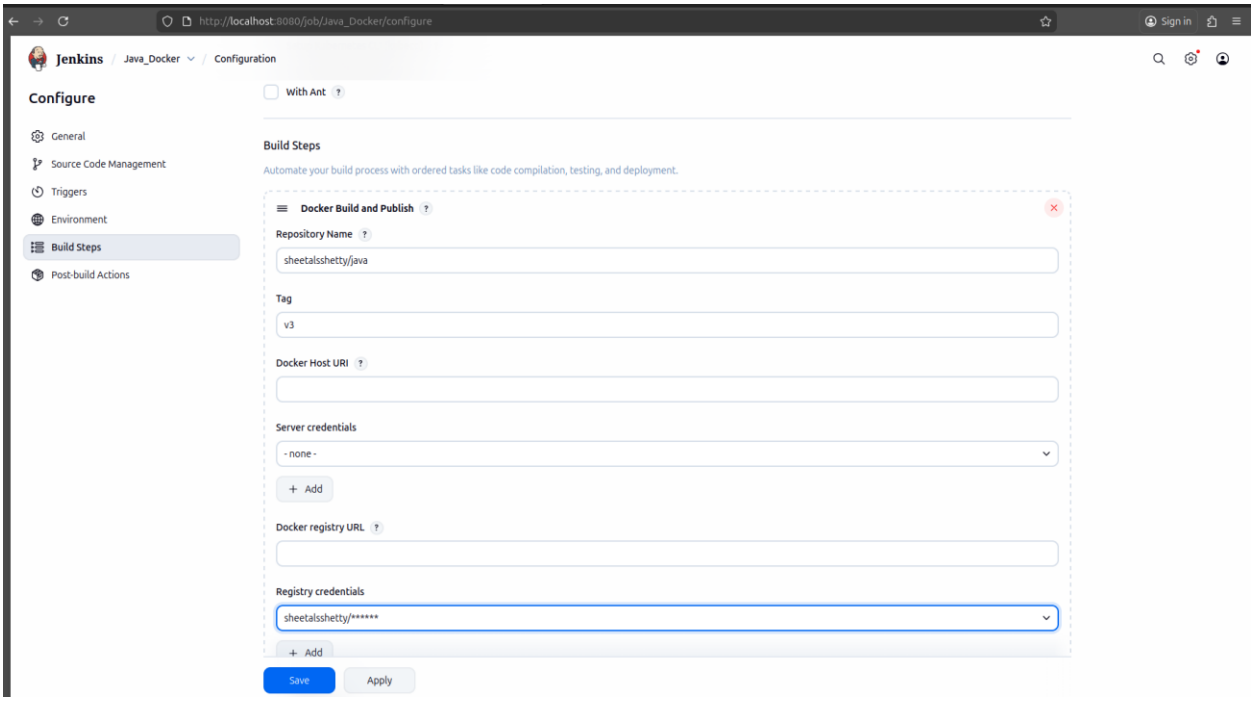
Add Docker credentials to Jenkins by navigating to **Manage Jenkins → Credentials**. Use your Docker Hub username as the **Username** and your Docker Hub access token as the **Password**.



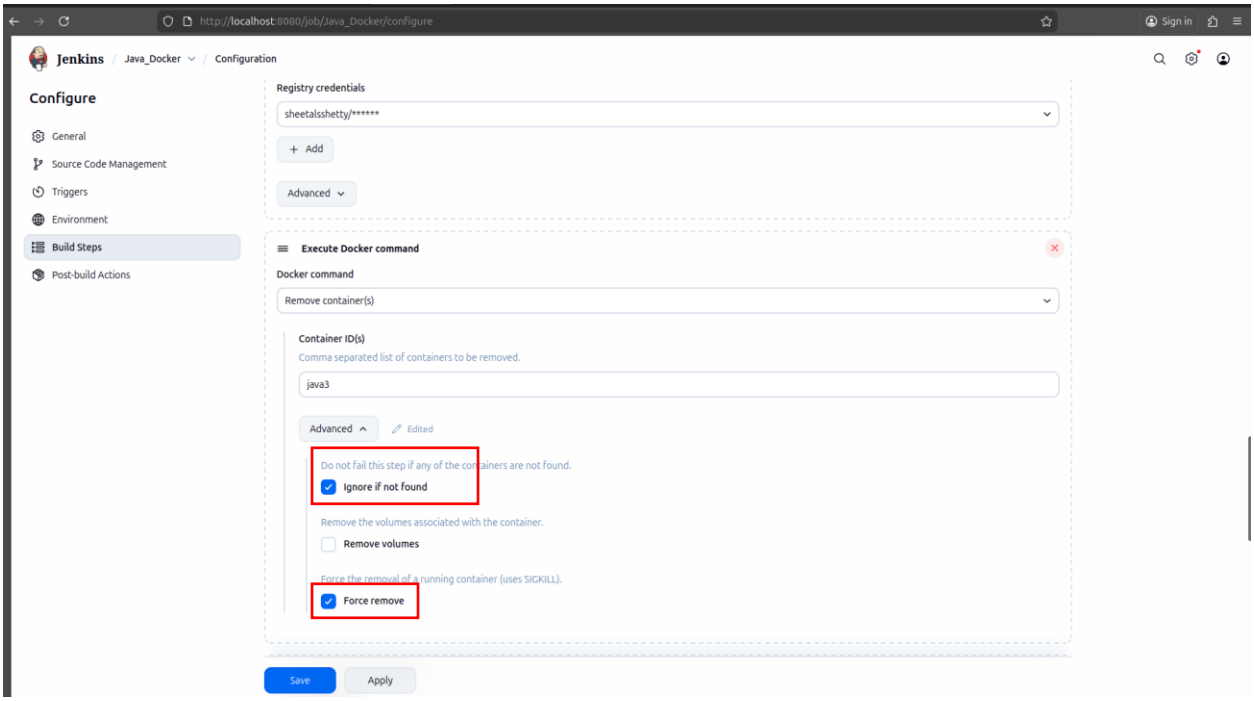
Under **Build Steps**, select **Docker Build and Publish**.

For **Repository Name**, enter <DockerHub_Username>/<image_name>.

Set the **Tag** (e.g., v3 or any version you prefer), and under **Registry credentials**, select the Docker credential ID.



Click on **“+ Add Build Step”** to add another build step and select **“Execute Docker command.”** Under **Docker command**, choose **“Remove container(s).”** In the **Container ID(s)** field, enter the name of the container you plan to create (e.g., java3). Also select the checkboxes for **“Ignore if not found”** and **“Force remove.”**



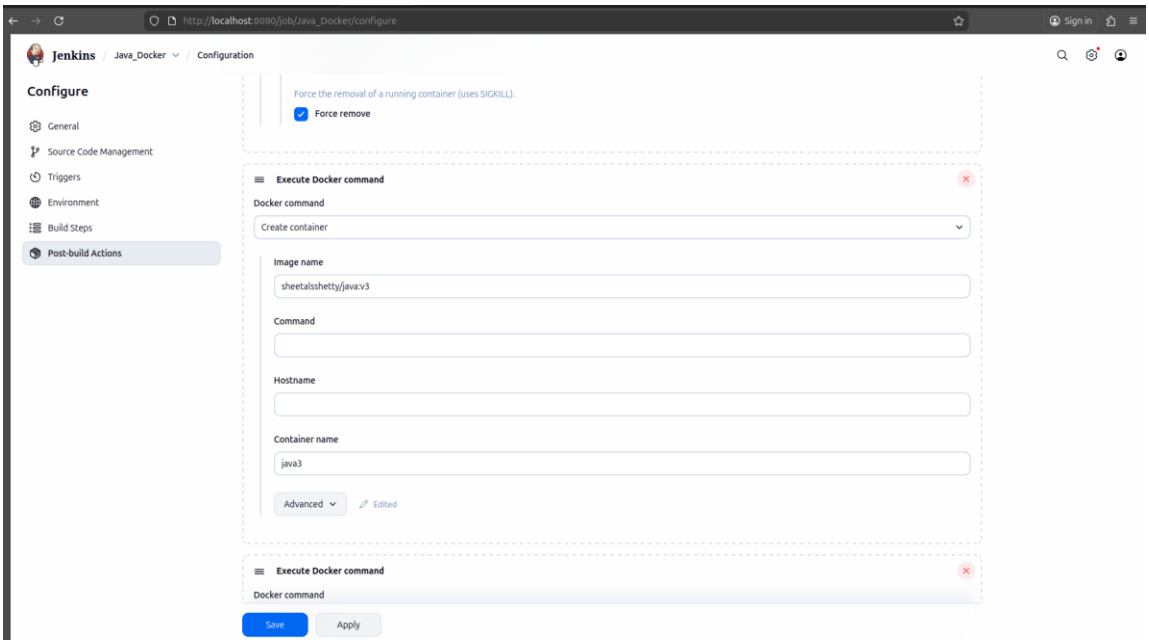
Click on **“+ Add Build Step”** to add another build step and select **“Execute Docker command.”**

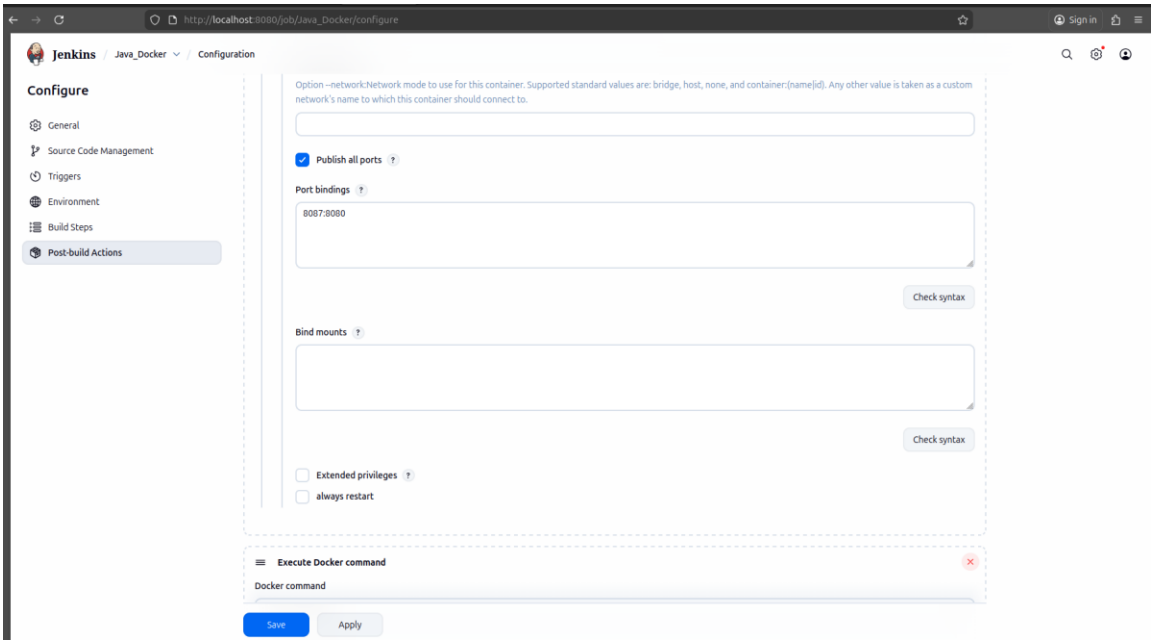
Under **Docker command**, choose **“Create container.”**

In the **Image name** field, enter the name of the image you created in the previous build step along with its version (e.g., sheetalsshetty/java:v3).

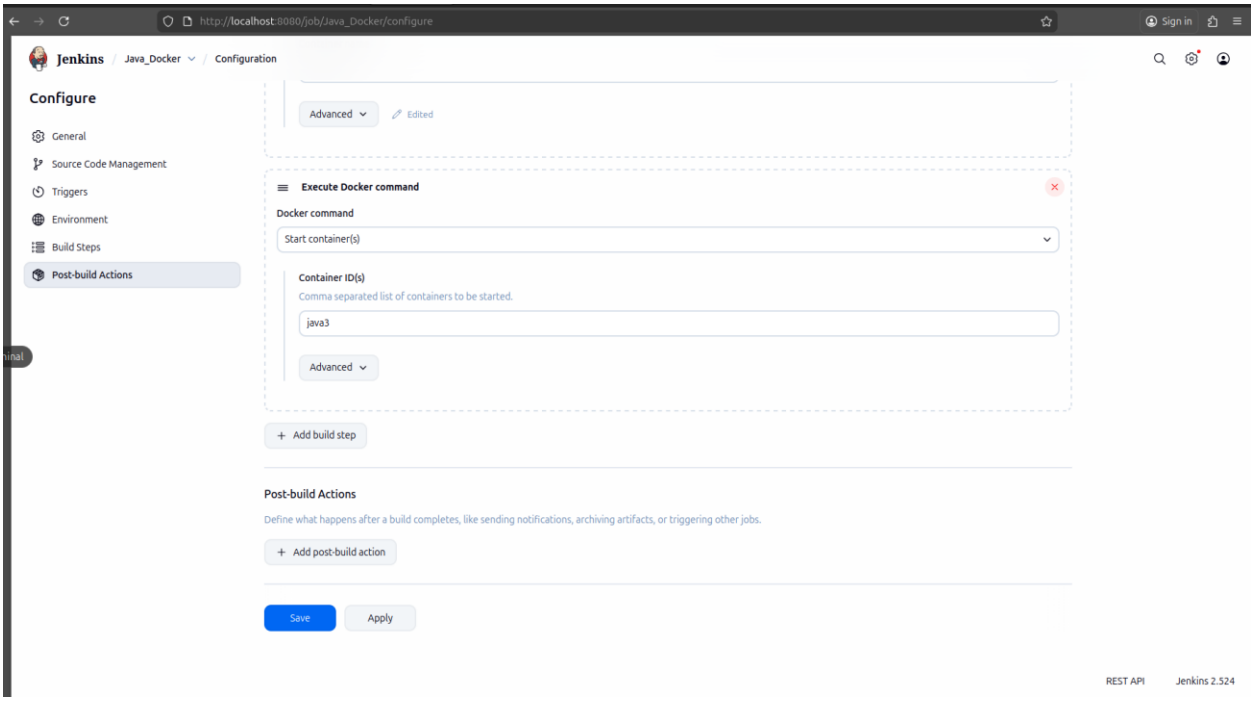
In the **Container name** field, provide the name of the container you want to create (e.g., java3).

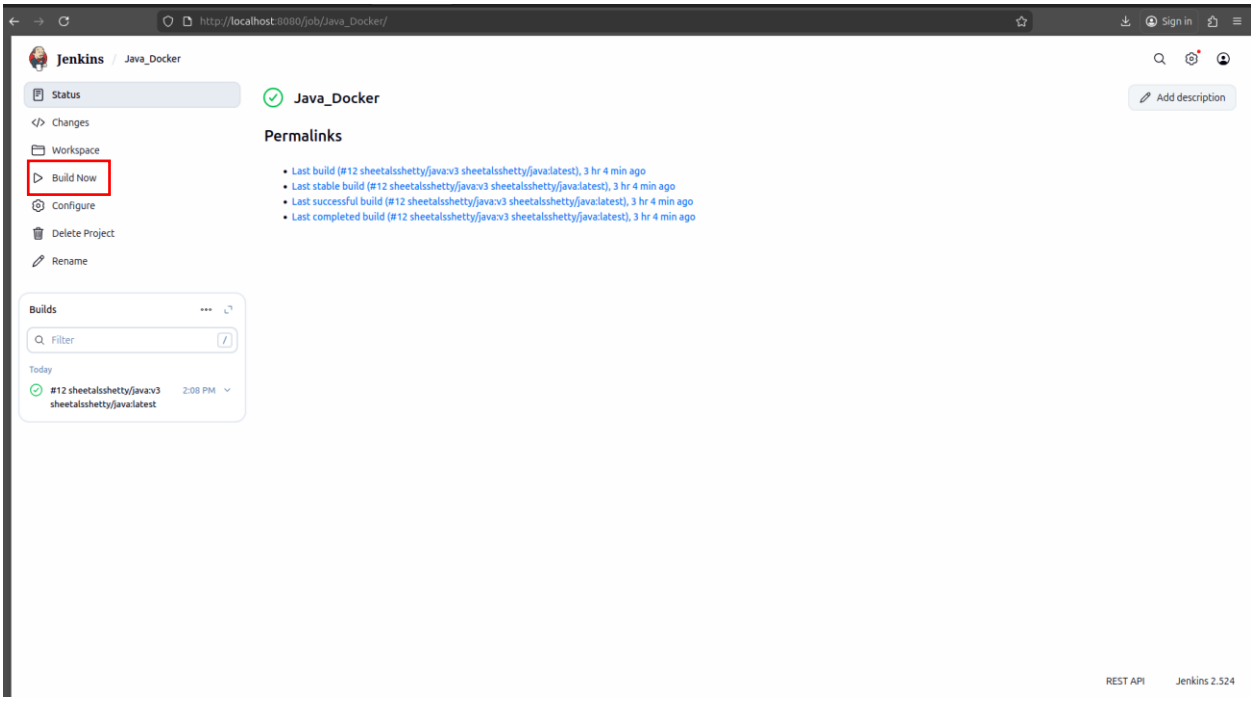
Click **Advanced**, select **“Publish all ports,”** and under **Port bindings**, map the ports in the format **Host_port:Container_port** (e.g., 8089:8080).



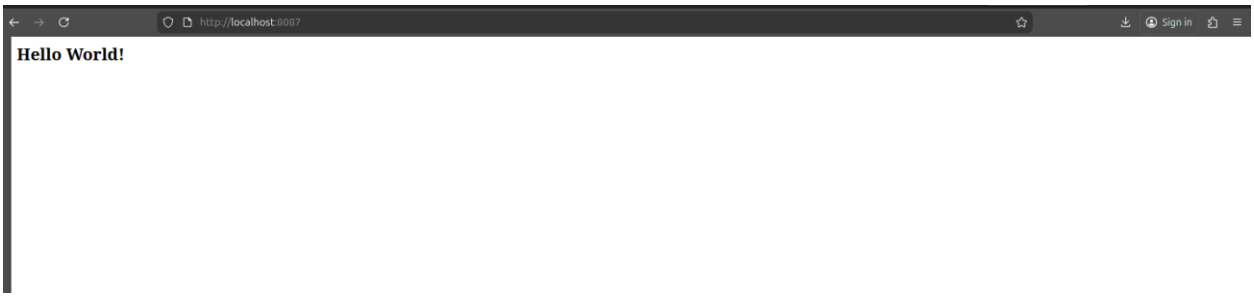


Click on **“+ Add Build Step”** to add another build step and select **“Execute Docker command.”** Under **Docker command**, choose **“Start container(s)”** and provide the name of the container you created in the **Container ID(s)** field (e.g., java3). Then click **Save**.





we can access the web page at http://localhost:<Host Port>/

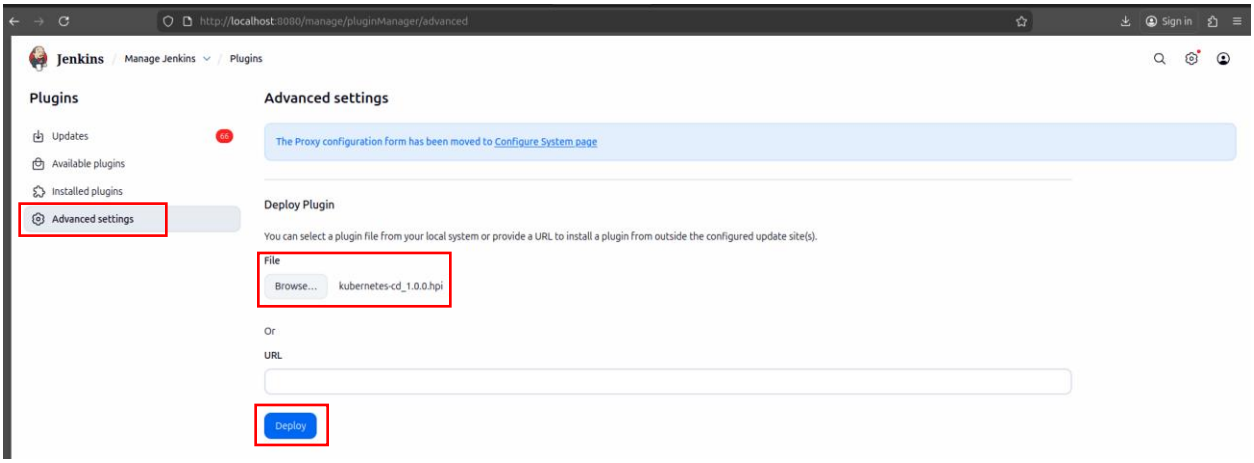


****

3. Deploy containerized application on Kubernetes cluster environment – Java

Plugins: Install the Kubernetes plugin using sir's kubernetes-cd_1.0.0.hpi file (available in the GitHub repository linked below).

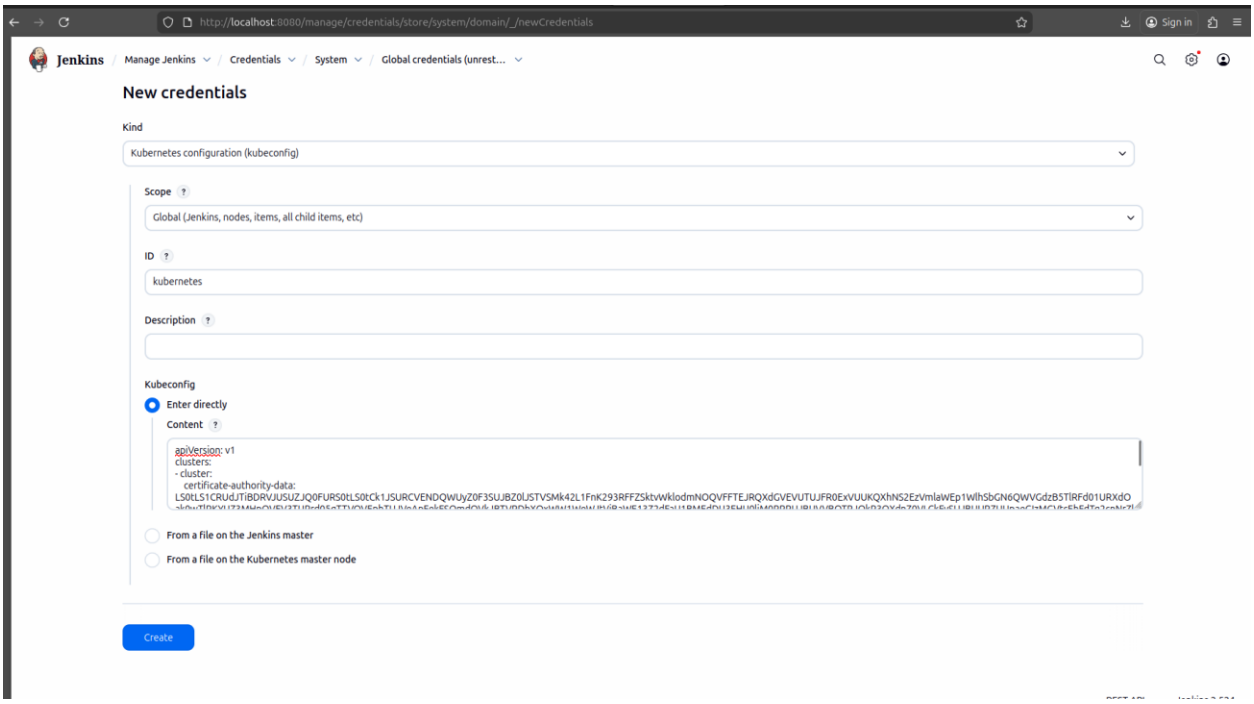
Go to **Manage Jenkins → Plugins → Advanced**, browse for the .hpi file, and click **Deploy**.



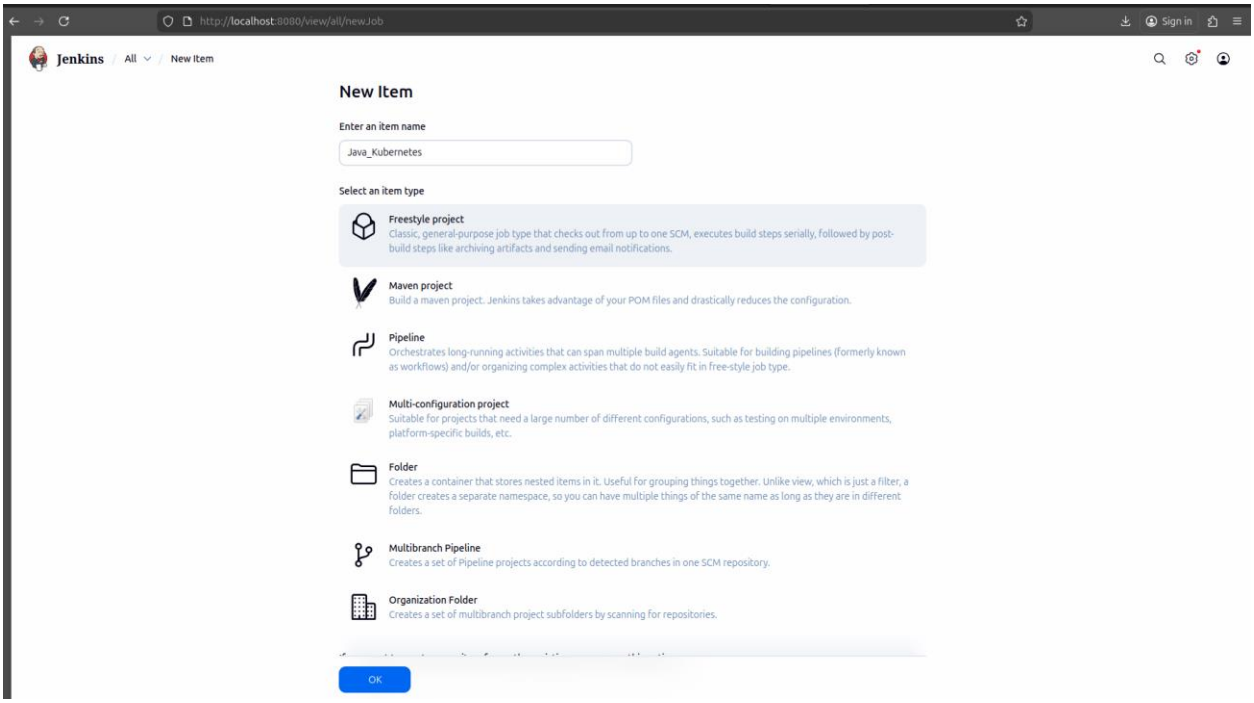
Created the Kubernetes manifest files (deployment.yaml and service.yaml) and added them to the GitHub repository.

GitHub Repo: https://github.com/Sheetal-Shetty/java-tomcat-maven-example.git

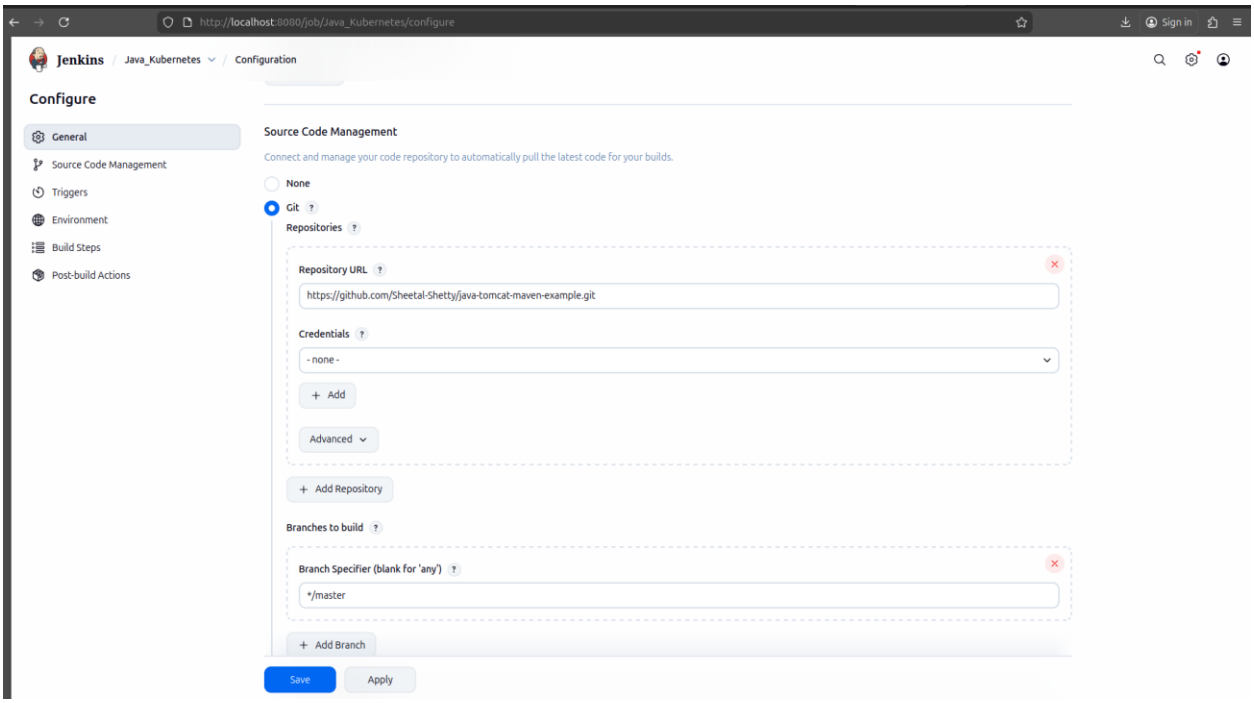
Add Kubernetes credentials to Jenkins by navigating to **Manage Jenkins → Plugins**. Then, under **Kind**, select **"Kubernetes configuration (kubeconfig)"** and provide a name for the credential under **ID**. For **Kubeconfig**, choose **"Enter directly"** and paste the output of cat ~/.kube/config from the master node. Finally, click **Create**.



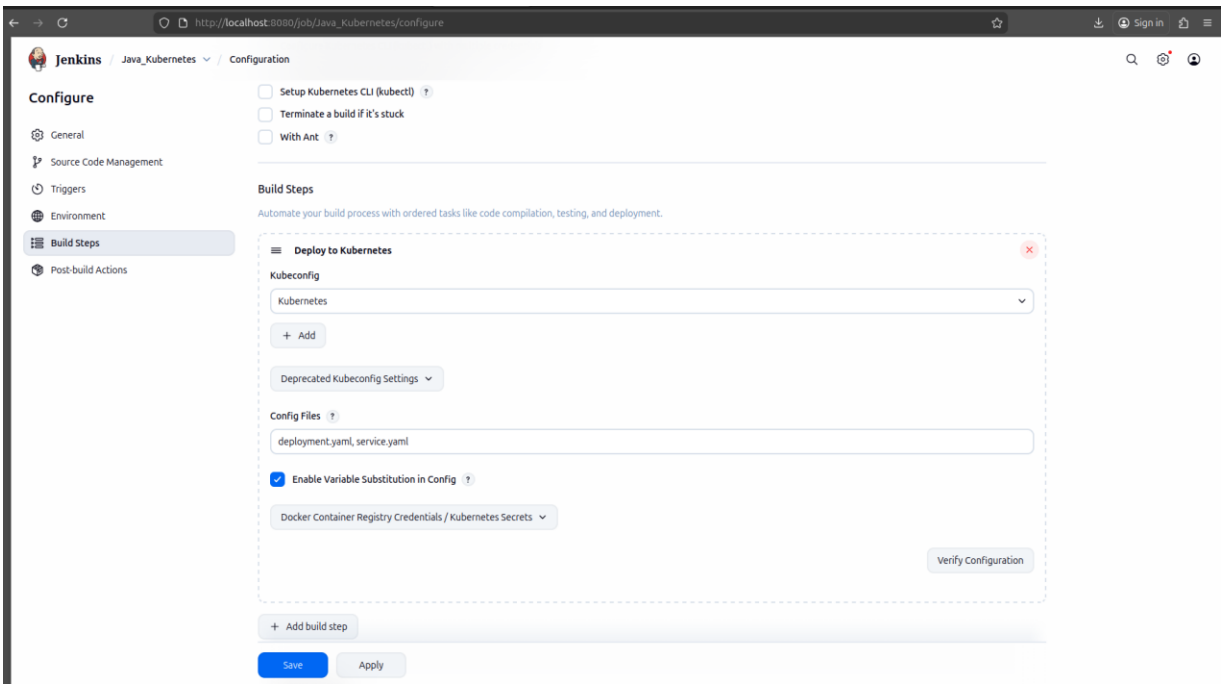
Create a **Freestyle Jenkins Project** and provide a name for the project, for example **Java_Kubernetes**, then click **OK**.

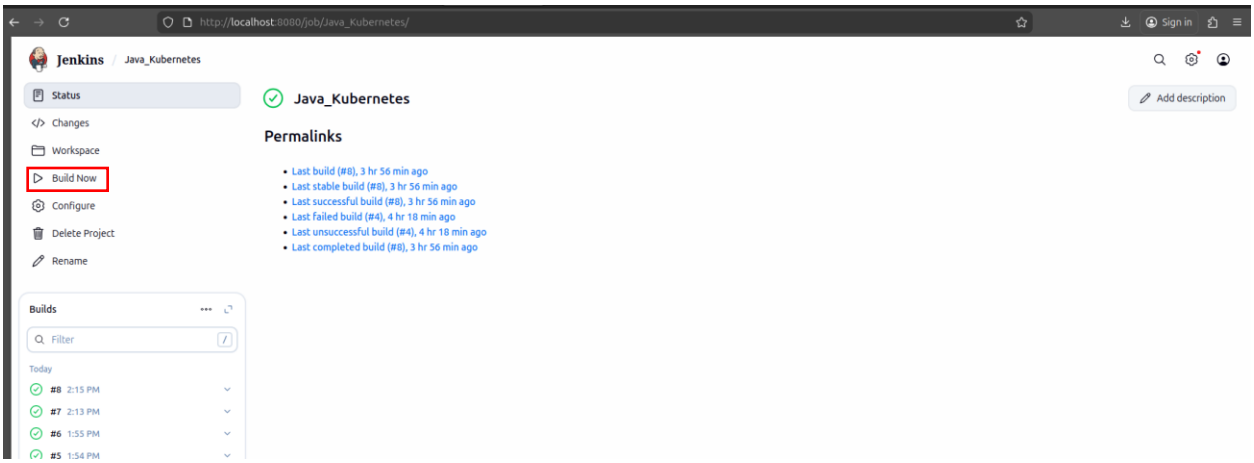


On the next page, select **Git** under **Source Code Management**, then enter the GitHub repository URL in the **Repository URL** field and specify the branch.



Under the **Build** step, select **“Deploy to Kubernetes.”** Then choose the Kubernetes credential ID you added under **Kubeconfig**. In the **Config Files** section, specify the names of your deployment and service files (e.g., deployment.yaml, service.yaml). Finally, click **Save** and then **Build**.





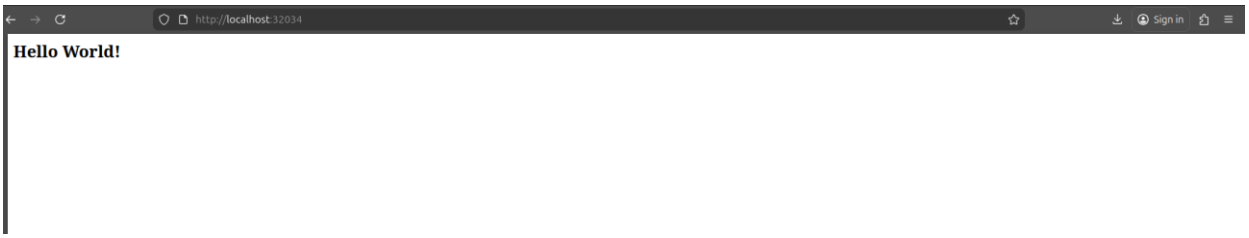
In terminal type kubectl get svc

```
msis@msis:~$ kubectl get svc javamaven
NAME         TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
javamaven    NodePort   10.104.151.252   <none>        8081:32034/TCP   69s
msis@msis:~$
```

We can access the web page at,

http://localhost:<NodePort> or http://<cluster ip>:<Target Port>

http://localhost:32034/

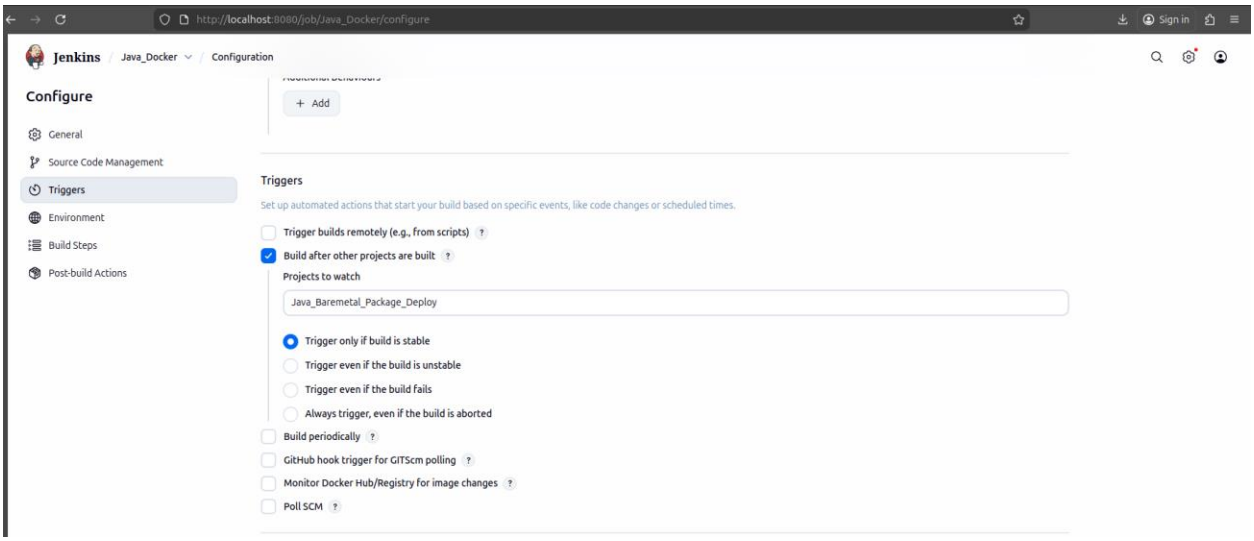


****

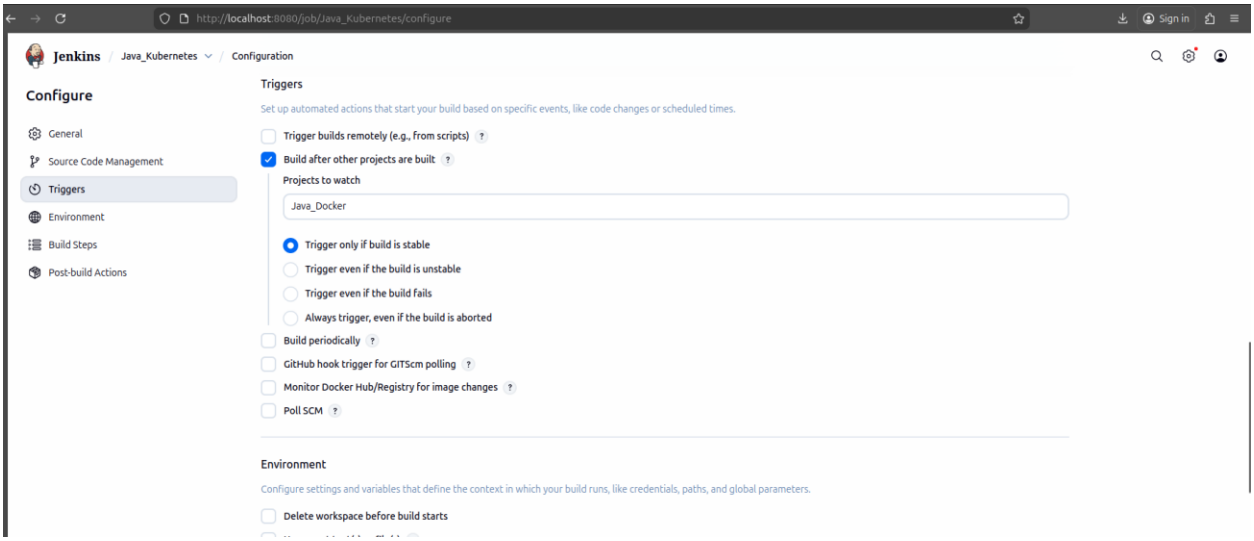
Create Pipeline for all these items (Baremetal->Docker->Kubernetes)

Since we have already created the pipeline for Java_Baremetal_Compile → Java_Baremetal_CodeReview → Java_Baremetal_Package_Deploy, we now need to add the remaining two stages (Java_Docker and Java_Kubernetes).

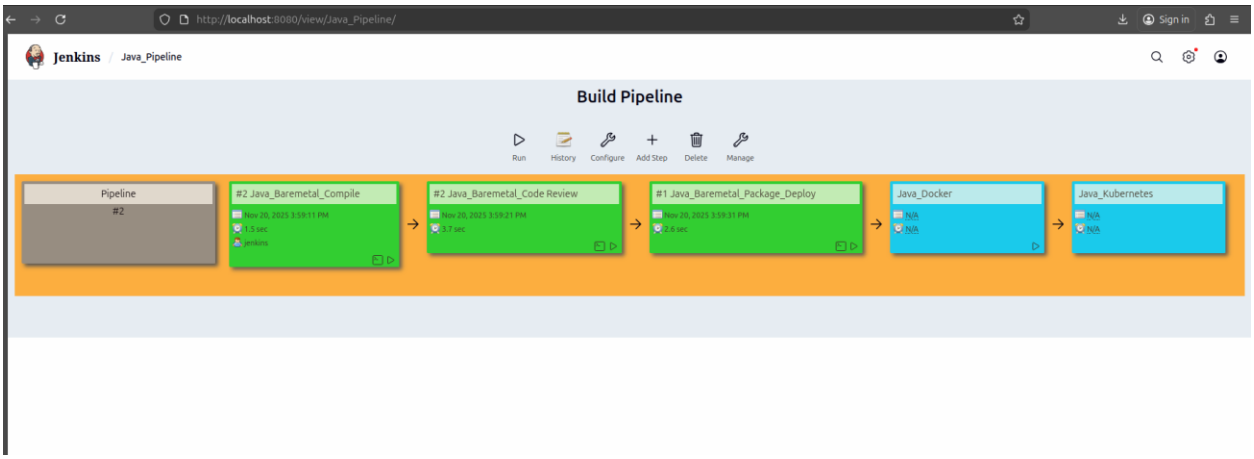
Go to the item named Java_Docker. Under Triggers, select “Build after other projects are built.” Then, under Projects to watch, select Java_Baremetal_Package_Deploy.



Go to the item named Java_Baremetal_Package_Deploy. Under Triggers, select “Build after other projects are built.” Then, under Projects to watch, select Java_Docker.



Now, the pipeline creation is complete.



Run the Pipeline.