

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
```

```
GNU nano 7.2                                tomcat-users.xml
application, do not forget to remove the <!-- ... --> that surrounds them. You
will also need to set the passwords to something appropriate.
-->
<!--
  <user username="admin" password="<must-be-changed>" roles="manager-gui"/>
  <user username="robot" password="<must-be-changed>" roles="manager-script"/>
-->
<!--
  The sample user and role entries below are intended for use with the
  examples web application. They are wrapped in a comment and thus are ignored
  when reading this file. If you wish to configure these users for use with the
  examples web application, do not forget to remove the <!-- ... --> that surrounds
  them. You will also need to set the passwords to something appropriate.
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
  <role rolename="manager-gui" />
  <role rolename="manager-script" />
  <role rolename="admin-gui" />
  <user username="msis" password="msis@123" roles="manager-gui,manager-script,admin-gui" />
-->
</tomcat-users>

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo      M-G Copy
```

```
cd /opt/tomcat/conf
```

```
sudo nano server.xml
```

```
GNU nano 7.2 server.xml

<!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
Java HTTP Connector: /docs/config/http.html
Java AJP Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
-->
<Connector port="8081" protocol="HTTP/1.1" address="0.0.0.0"
connectionTimeout="20000"
redirectPort="8443"
maxParameterCount="1000"
/>

<!-- A "Connector" using the shared thread pool-->
<!--
<Connector executor="tomcatThreadPool"
port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443"
maxParameterCount="1000"
/>

-->
<!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443
This connector uses the NIO implementation. The default
SSLImplementation will depend on the presence of the APR/native
library and the useOpenSSL attribute of the AprLifecycleListener.
Either JSSE or OpenSSL style configuration may be used regardless of
-->
```

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.

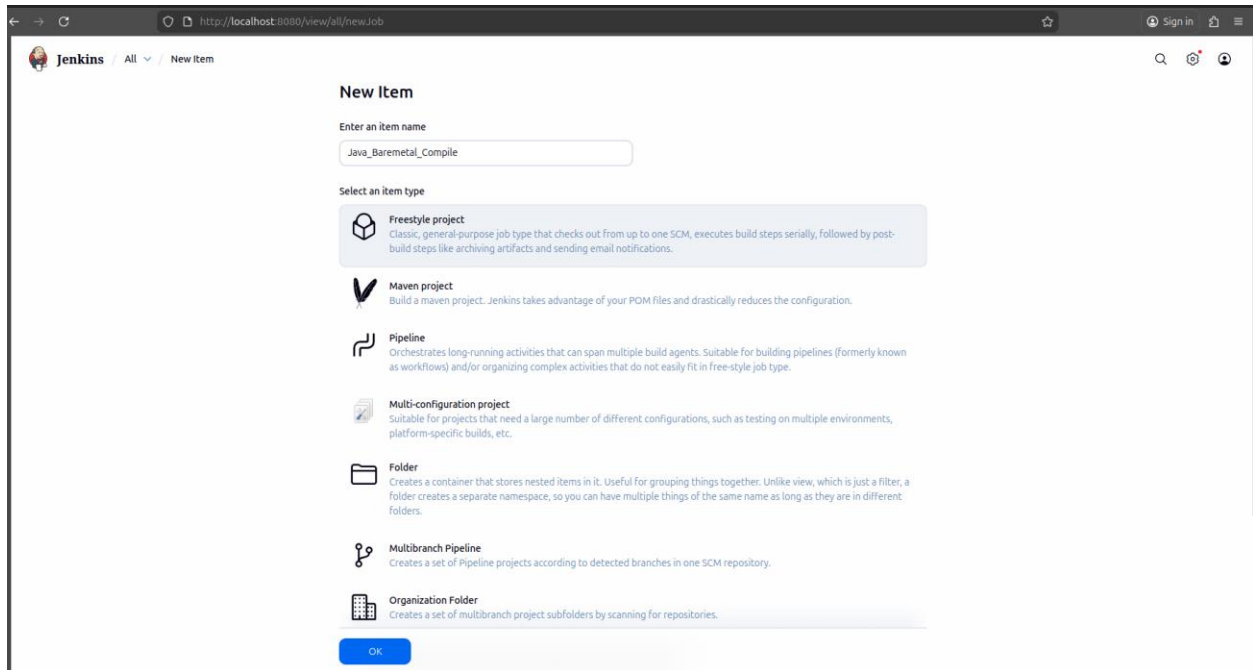
GitHub Repo: <https://github.com/sreepathysois/java-tomcat-maven-example.git>

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.

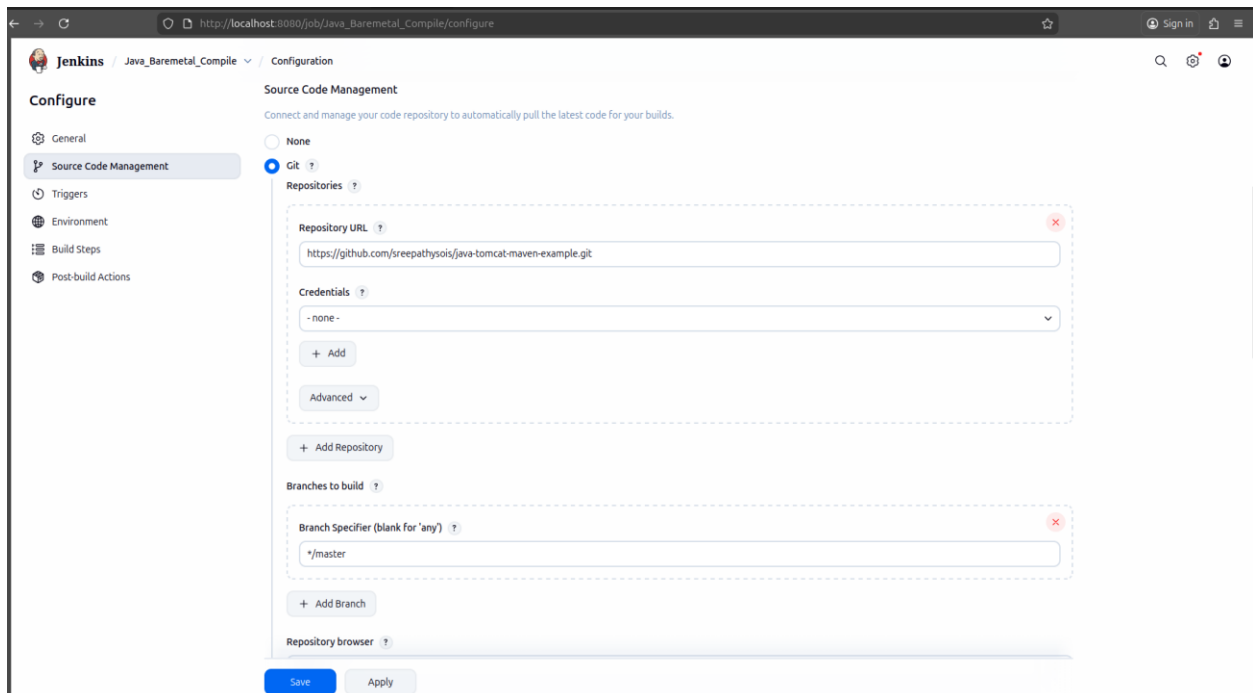


The screenshot shows the Jenkins 'New Item' configuration page. The browser address bar indicates the URL is `http://localhost:8080/view/all/newJob`. The Jenkins logo and 'All / New Item' breadcrumb are at the top left. The page title is 'New Item'. Below the title, there is a text input field for 'Enter an item name' containing the text 'Java_Baremetal_Compile'. Underneath, a section titled 'Select an item type' lists several options with icons and descriptions:

- Freestyle project**: Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository.
- Organization Folder**: Creates a set of multibranch project subfolders by scanning for repositories.

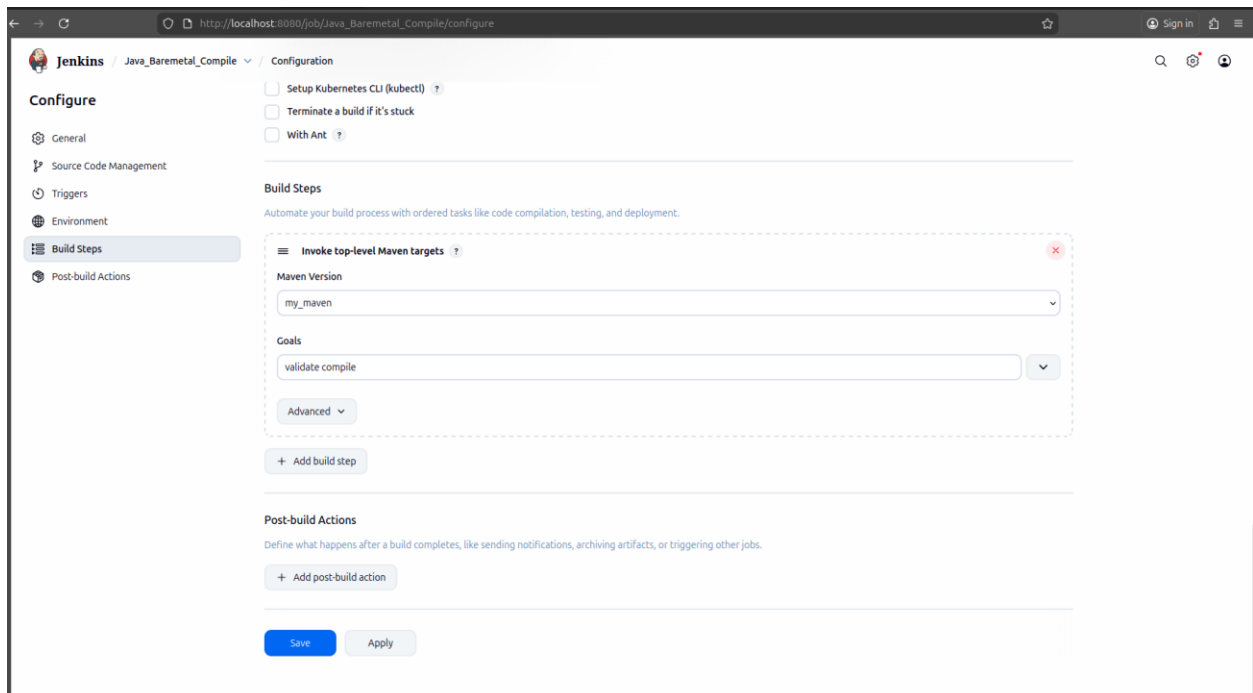
At the bottom of the list is a blue 'OK' button.

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



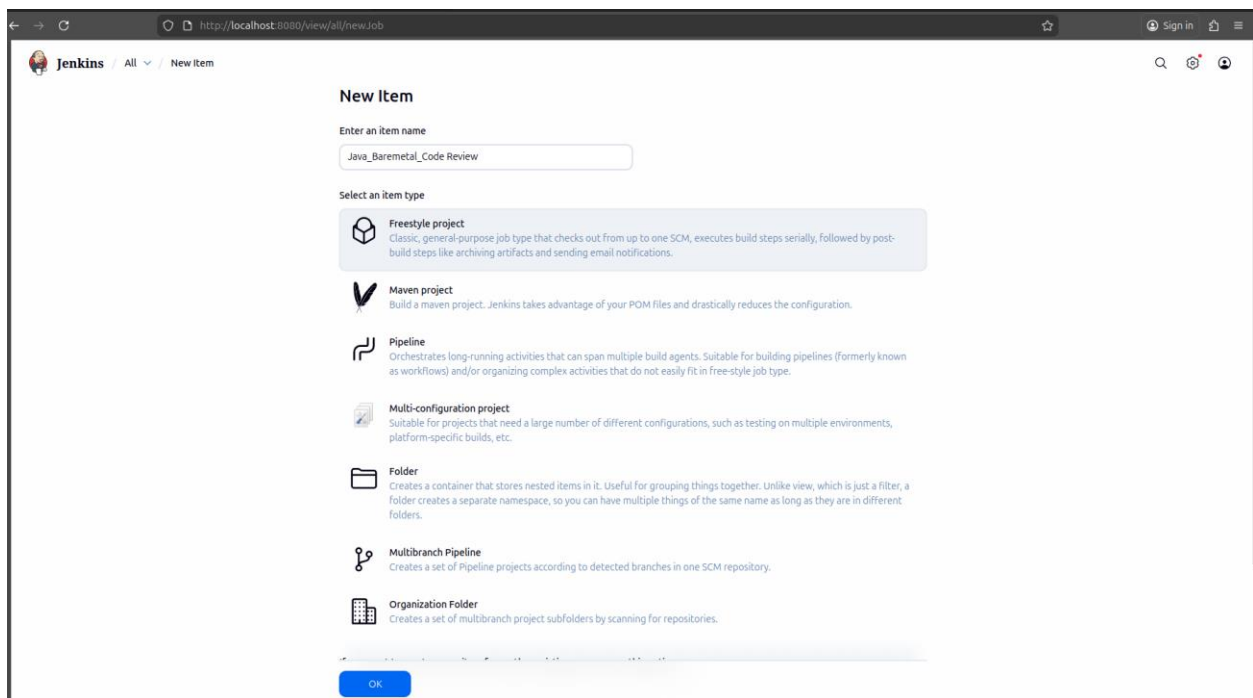
The screenshot shows the Jenkins 'Configuration' page for the job 'Java_Baremetal_Compile'. The browser address bar indicates the URL is `http://localhost:8080/job/Java_Baremetal_Compile/configure`. The Jenkins logo and 'Java_Baremetal_Compile / Configuration' breadcrumb are at the top left. The page title is 'Configure'. On the left, a sidebar lists configuration sections: General, Source Code Management (selected), Triggers, Environment, Build Steps, and Post-build Actions. The main content area is titled 'Source Code Management' and includes the instruction 'Connect and manage your code repository to automatically pull the latest code for your builds.' Below this, there are two radio buttons: 'None' and 'Git' (selected). Under the 'Git' section, there is a 'Repositories' section with a dashed border. It contains a 'Repository URL' input field with the value `https://github.com/sreepathysois/java-tomcat-maven-example.git`, a 'Credentials' dropdown menu set to '- none -', an '+ Add' button, and an 'Advanced' dropdown. Below the 'Repositories' section is an '+ Add Repository' button. There is also a 'Branches to build' section with a dashed border, containing a 'Branch Specifier (blank for 'any')' input field with the value `*/master`, and an '+ Add Branch' button. At the bottom, there is a 'Repository browser' dropdown menu. At the very bottom of the page are 'Save' and 'Apply' buttons.

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

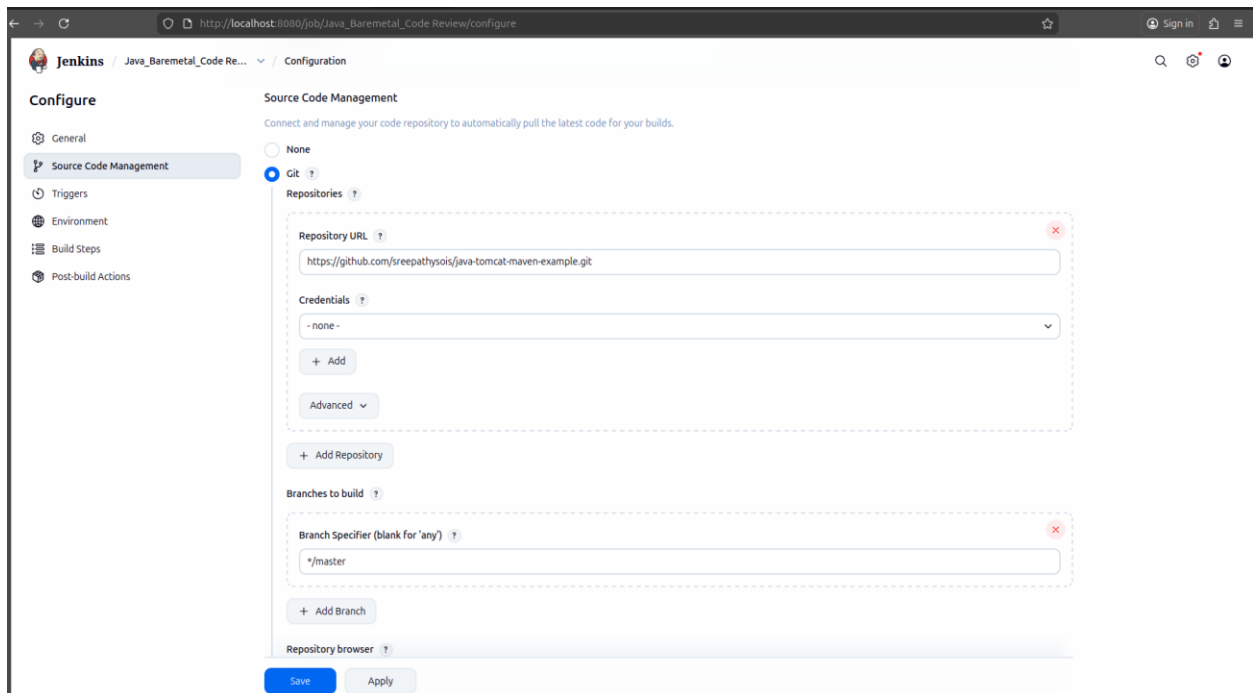


Step2: Code review

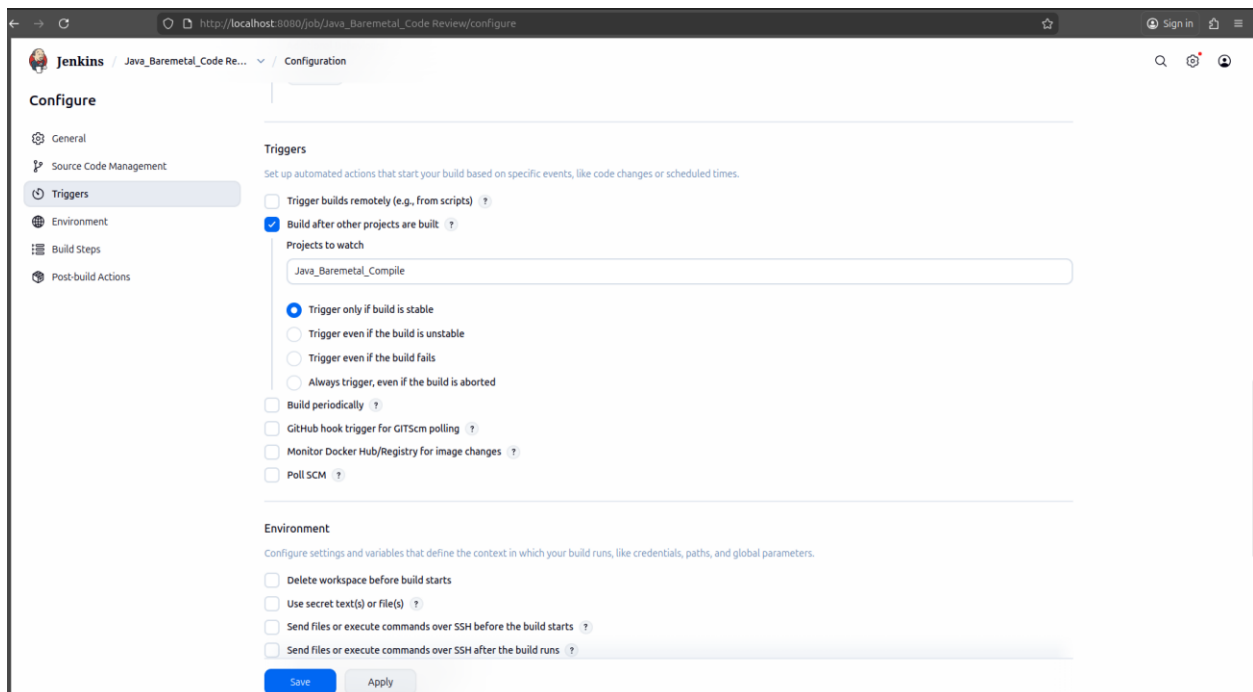
Create a Freestyle Jenkins project. Provide a name for the project, for example: `Java_Baremetal_CodeReview`, then click OK.



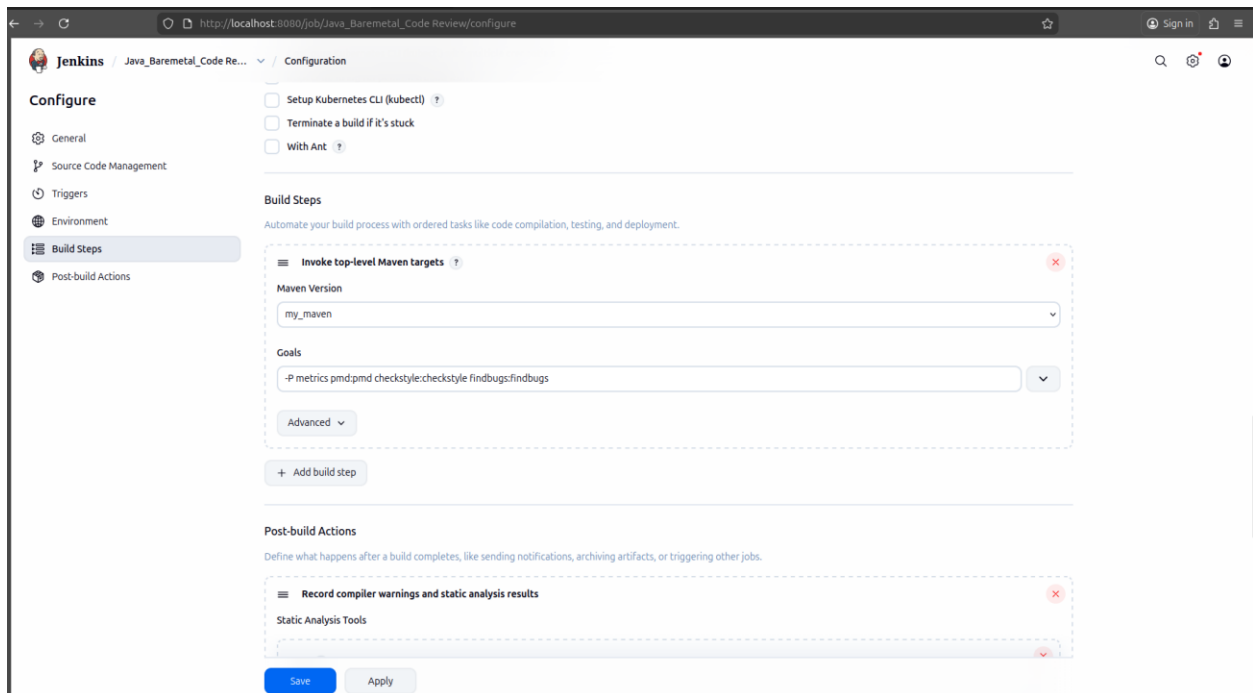
On the next page, select Source Code Management as Git, then provide the GitHub repository URL under Repository URL and specify the branch.



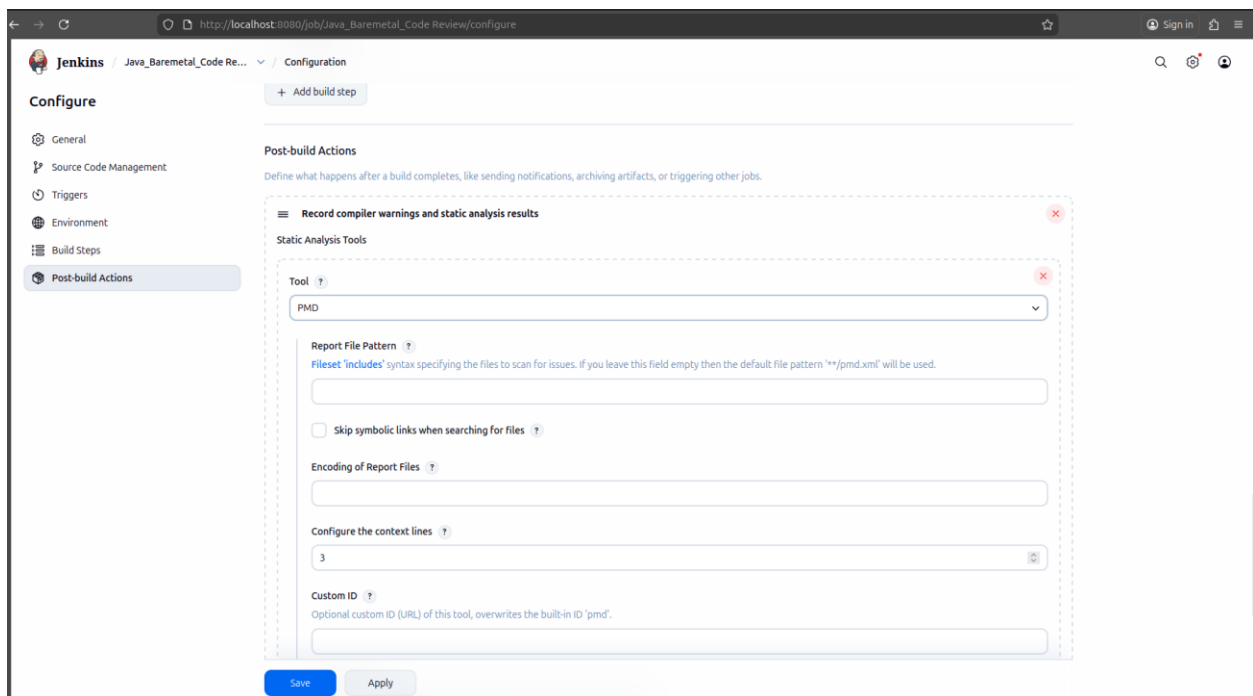
Under Triggers, select “Build after other projects are built,” and under Projects to watch, enter the previous item name (Java_Baremetal_Compile). Then select “Trigger only if build is stable.”



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 '**/findbugs.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 ?

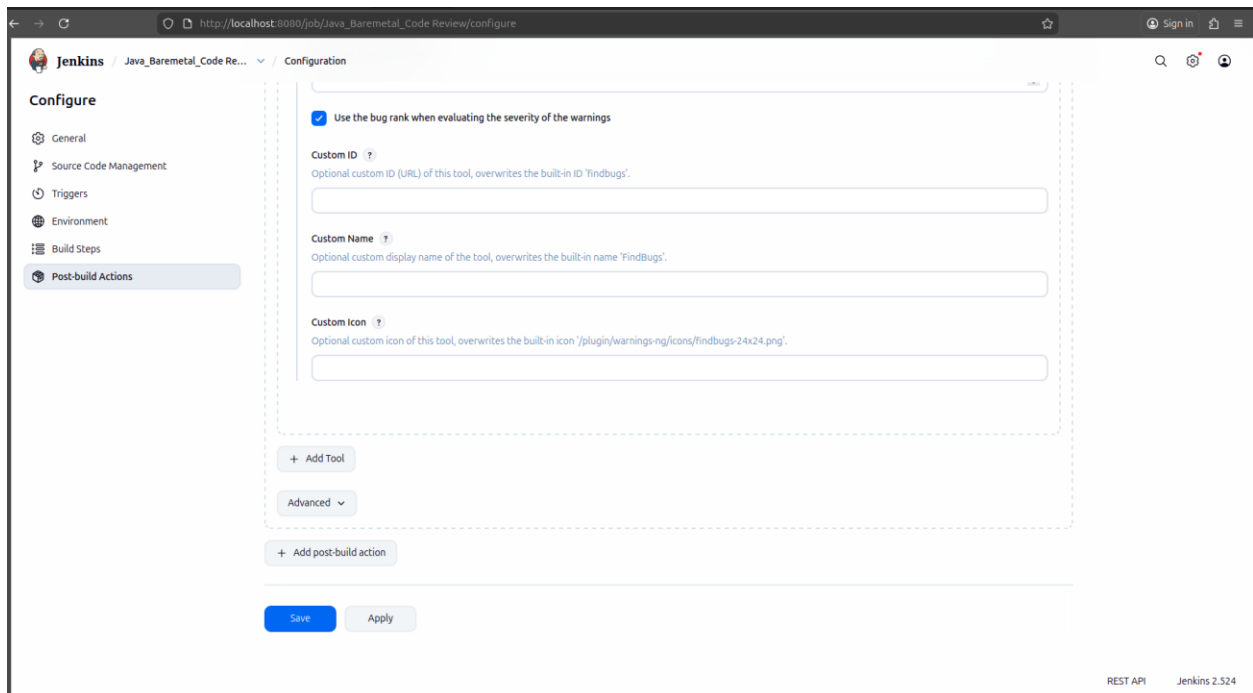
Optional custom ID (URL) of this tool, overwrites the built-in ID 'FindBugs'.

Custom Name ?

Optional custom display name of the tool, overwrites the built-in name 'FindBugs'.

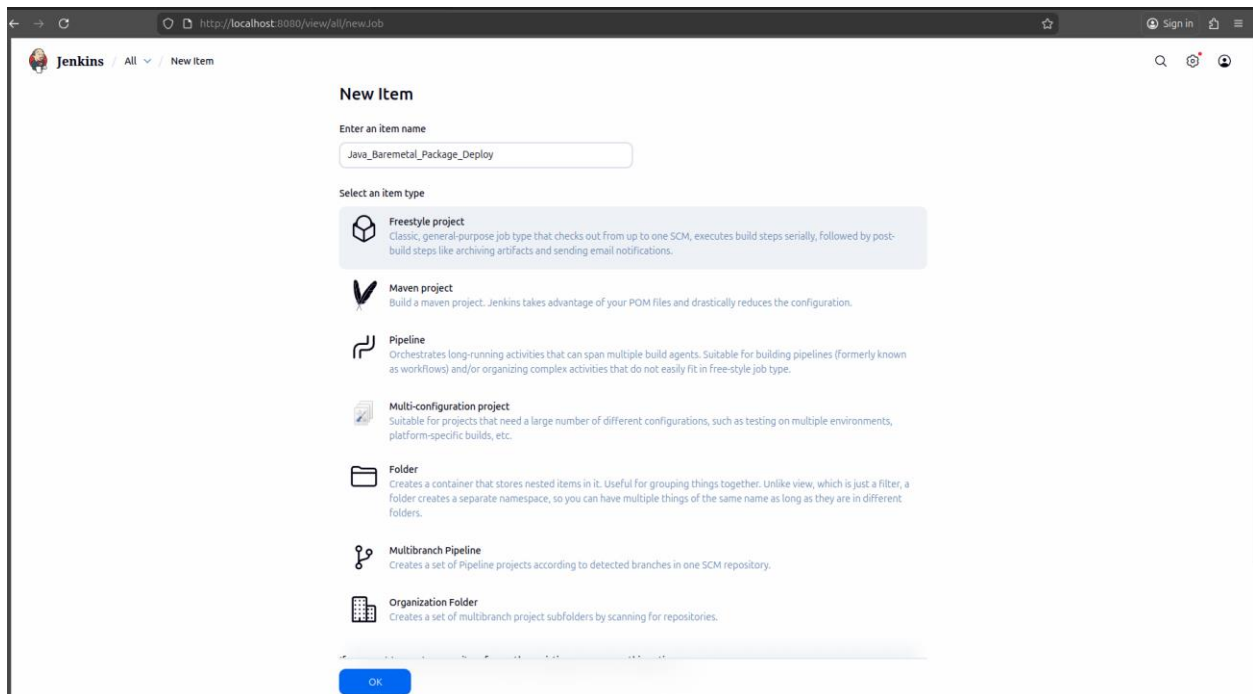
Save

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**.



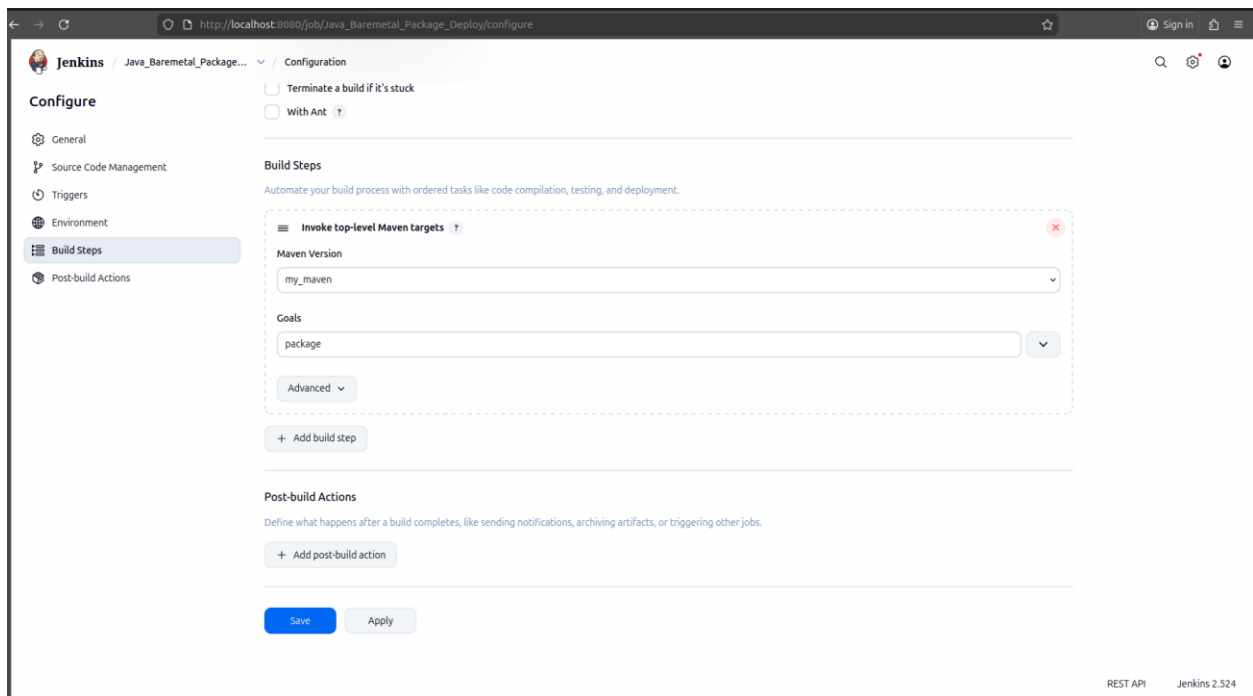
On the next page, select **Git** under **Source Code Management**, then enter the GitHub repository URL in the **Repository URL** field and specify the branch.

The screenshot shows the Jenkins Configuration page for the job 'Java_Baremetal_Package_Deploy'. The 'Source Code Management' section is active. The 'Git' option is selected under 'Source Code Management'. The 'Repository URL' is set to 'https://github.com/sreepathysis/java-tomcat-maven-example.git'. The 'Credentials' dropdown is set to '- none -'. The 'Branches to build' section has a 'Branch Specifier (blank for \'any\')' set to '*/master'. The 'Save' button is highlighted in blue.

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.”**

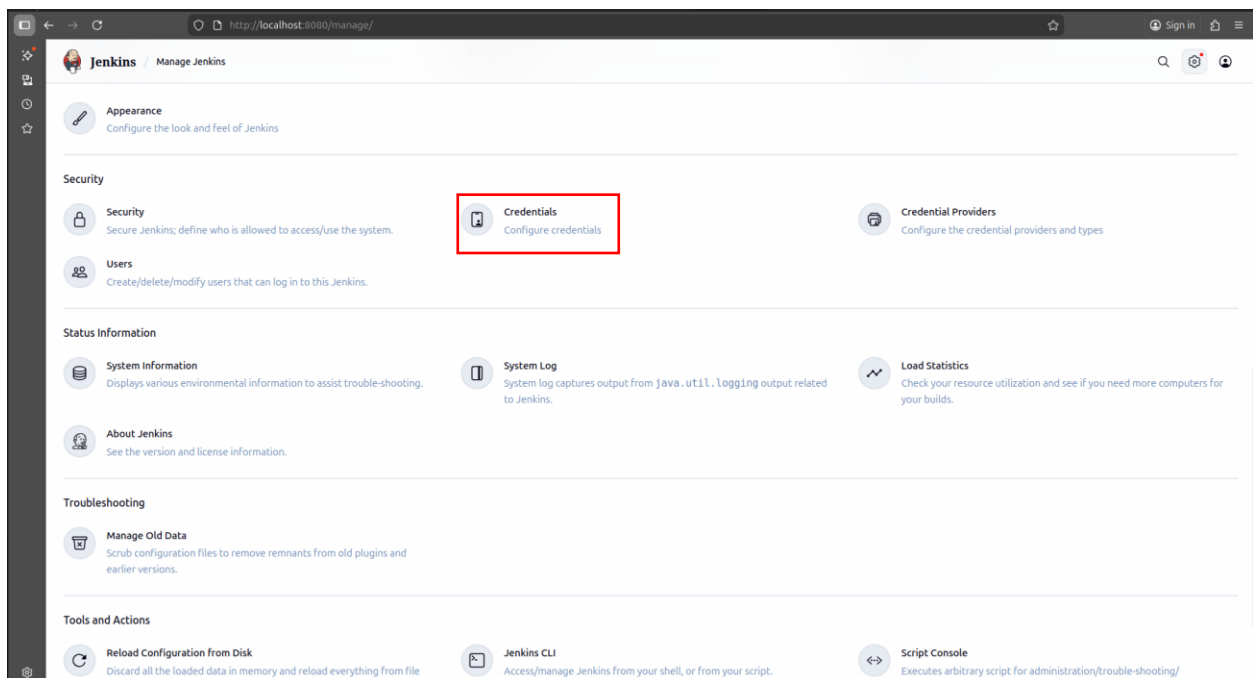
The screenshot shows the Jenkins Configuration page for the job 'Java_Baremetal_Package_Deploy'. The 'Triggers' section is active. The 'Build after other projects are built' checkbox is checked. The 'Projects to watch' field is set to 'Java_Baremetal_Code Review'. The 'Trigger only if build is stable' radio button is selected. The 'Save' button is highlighted in blue.

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.

Credentials

T	P	Store	Domain	ID	Name
		System	(global)	tomcat_manager_script_cred_38	msis/***** (tomcat_manager_script_cred_38)
		System	(global)	Docker	sheetalshetty/*****
		System	(global)	Kubernetes	Kubernetes
		System	(global)	test	test
		System	(global)	minikube	minikube
		System	(global)	minikube2	minikube2
		System	(global)	minikube3	minikube3

Stores scoped to Jenkins

P	Store	Domains
	System	(global)
	Kubernetes	(global)

Icons: S M L

REST API Jenkins 2.524

Click on Add Credential

Global credentials (unrestricted)

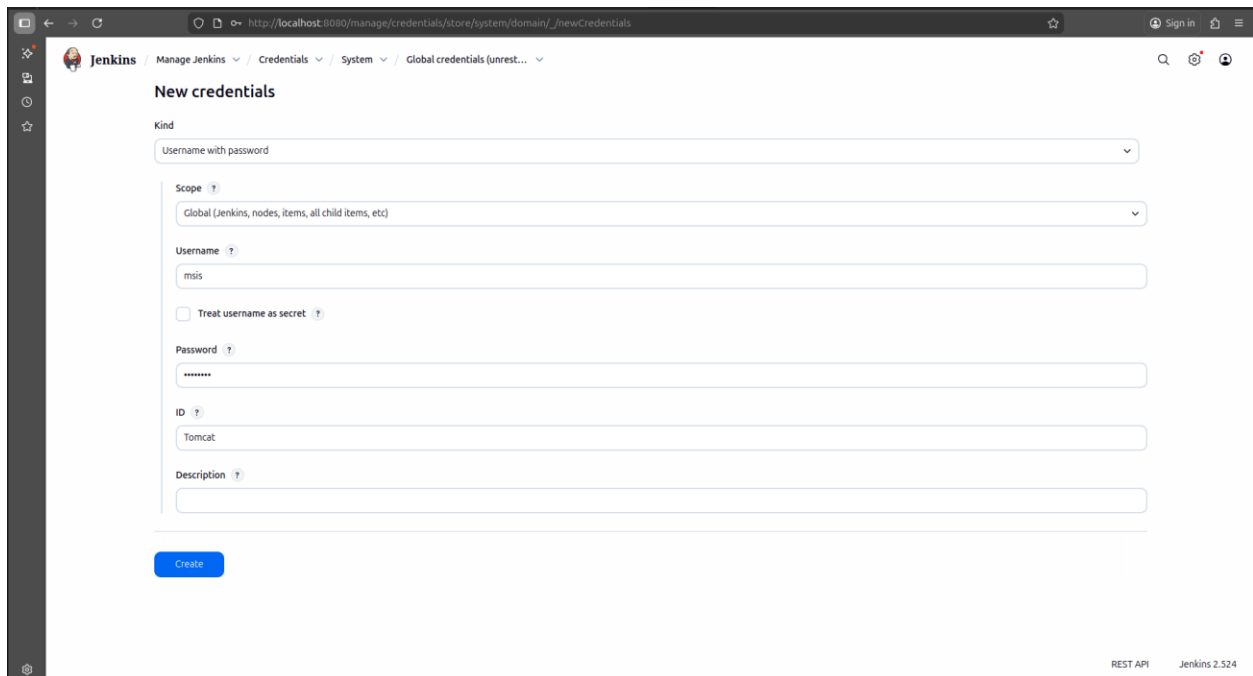
Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
tomcat_manager_script_cred_38	msis/***** (tomcat_manager_script_cred_38)	Username with password	tomcat_manager_script_cred_38
Docker	sheetalshetty/*****	Username with password	
Kubernetes	Kubernetes	Kubernetes configuration (kubeconfig)	
test	test	Kubernetes configuration (kubeconfig)	
minikube	minikube	Kubernetes configuration (kubeconfig)	
minikube2	minikube2	Kubernetes configuration (kubeconfig)	
minikube3	minikube3	Kubernetes configuration (kubeconfig)	

Icons: S M L

REST API Jenkins 2.524

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



The screenshot shows the Jenkins 'New credentials' form. The 'Kind' dropdown is set to 'Username with password'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'msis'. The 'Treat username as secret' checkbox is unchecked. The 'Password' field contains eight asterisks. The 'ID' field contains 'Tomcat'. The 'Description' field is empty. A blue 'Create' button is at the bottom left. The top navigation bar shows 'Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrest...)'. The bottom right corner shows 'REST API' and 'Jenkins 2.524'.

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: msis

☐ Treat username as secret

Password: *****

ID: Tomcat

Description:

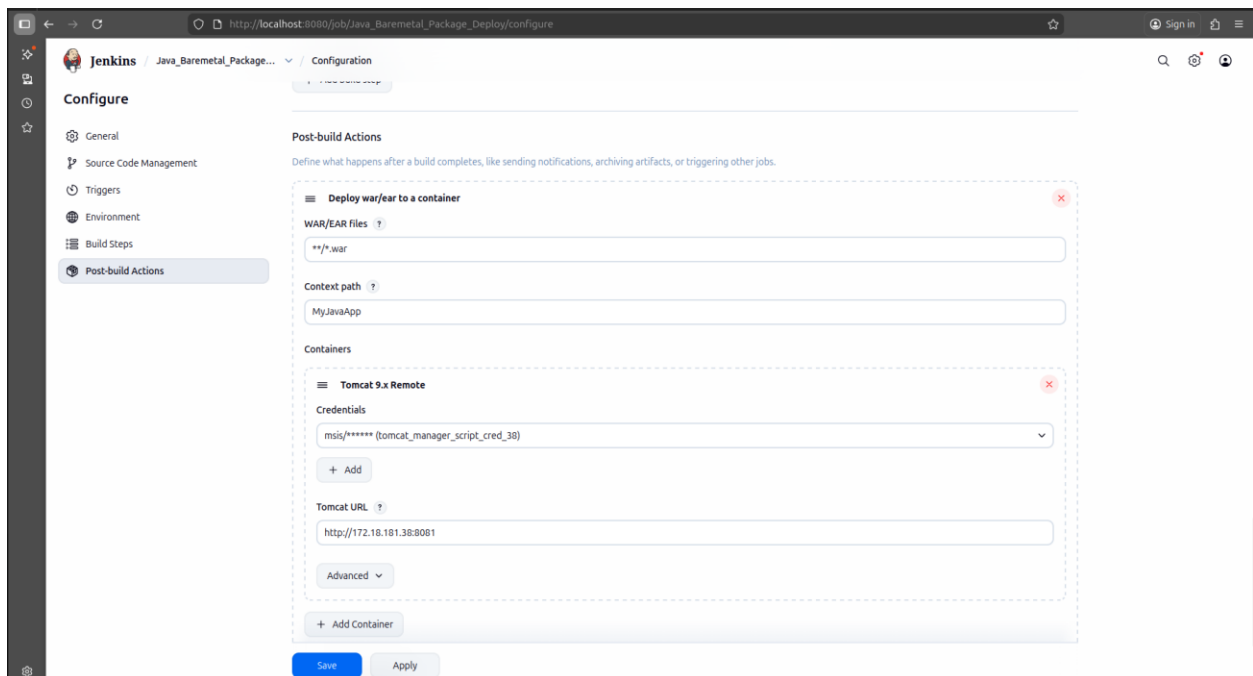
Create

REST API Jenkins 2.524

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**.



The screenshot shows the Jenkins 'Configure' page for the job 'Java_Baremetal_Package_Deploy'. The 'Post-build Actions' section is expanded, showing the 'Deploy war/ear to a container' action. The 'WAR/EAR Files' field contains '**/*.war'. The 'Context path' field contains 'MyJavaApp'. The 'Containers' section shows 'Tomcat 9.x Remote' selected. The 'Credentials' dropdown is set to 'msis/***** (tomcat_manager_script_cred_30)'. The 'Tomcat URL' field contains 'http://172.18.181.38:8081'. There are '+ Add' buttons for credentials and containers, and an 'Advanced' dropdown. At the bottom are 'Save' and 'Apply' buttons. The left sidebar shows the 'Configure' menu with 'Post-build Actions' selected. The top navigation bar shows 'Jenkins / Java_Baremetal_Package... / Configuration'.

Configure

General

Source Code Management

Triggers

Environment

Build Steps

Post-build Actions

Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

Deploy war/ear to a container

WAR/EAR Files: **/*.war

Context path: MyJavaApp

Containers

Tomcat 9.x Remote

Credentials: msis/***** (tomcat_manager_script_cred_30)

+ Add

Tomcat URL: http://172.18.181.38:8081

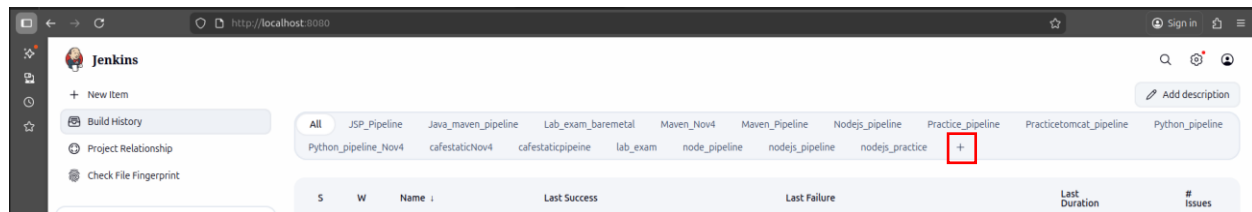
Advanced

+ Add Container

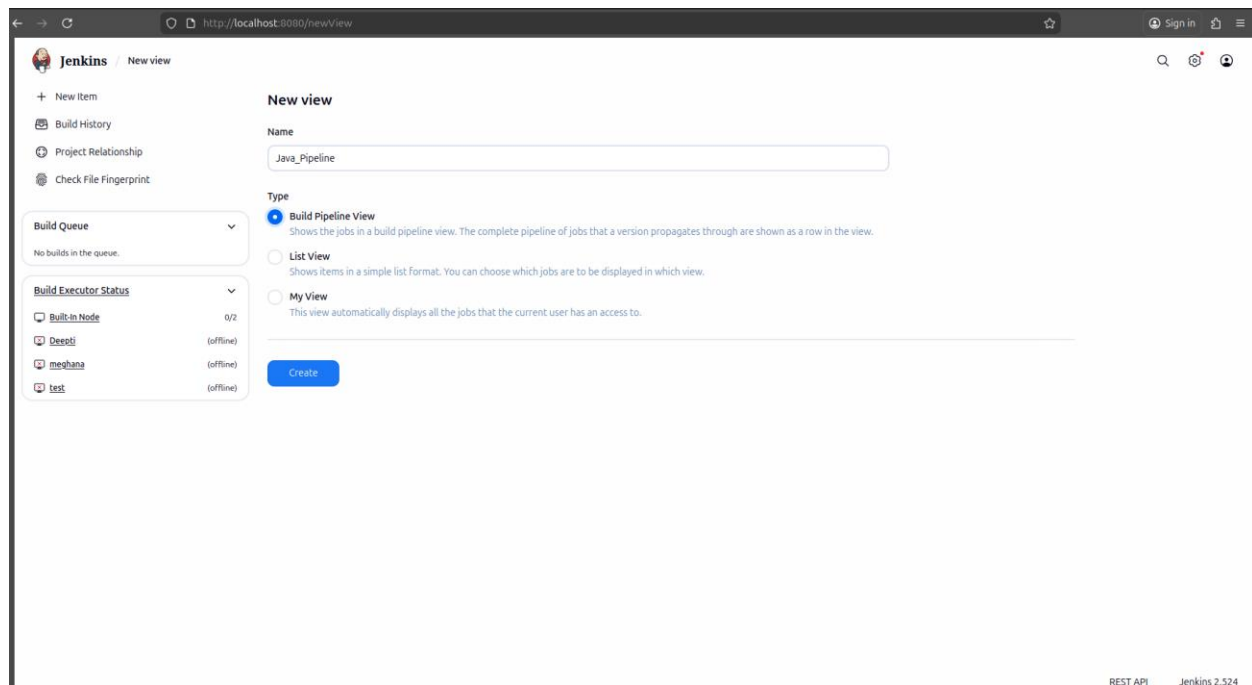
Save Apply

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**.

← → ↻ http://localhost:8080/view/Java_Pipeline/configure Sign in

Jenkins / Java_Pipeline / Edit View

Layout

Based on upstream/downstream relationship

This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config

Select Initial Job ?

Java_Baremetal_Compile

Trigger Options

Build Cards

Standard build card

Use the default build cards

Restrict triggers to most recent successful builds ?

☐ Yes

☒ No

Always allow manual trigger on pipeline steps ?

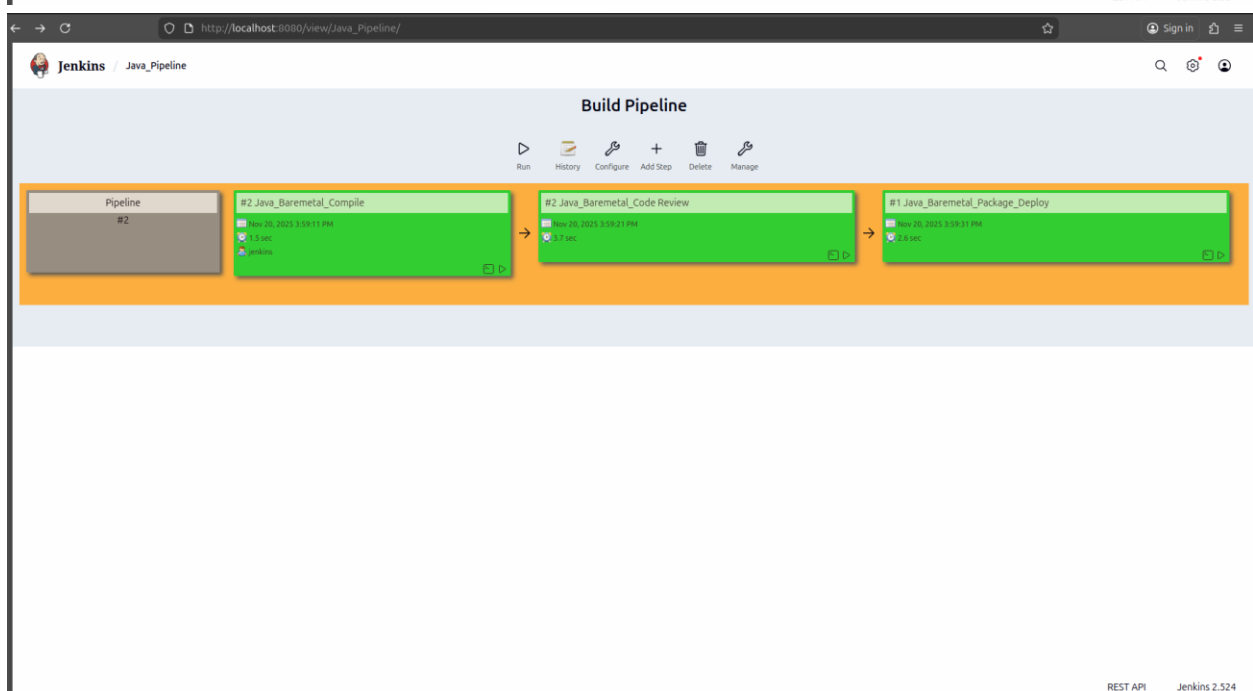
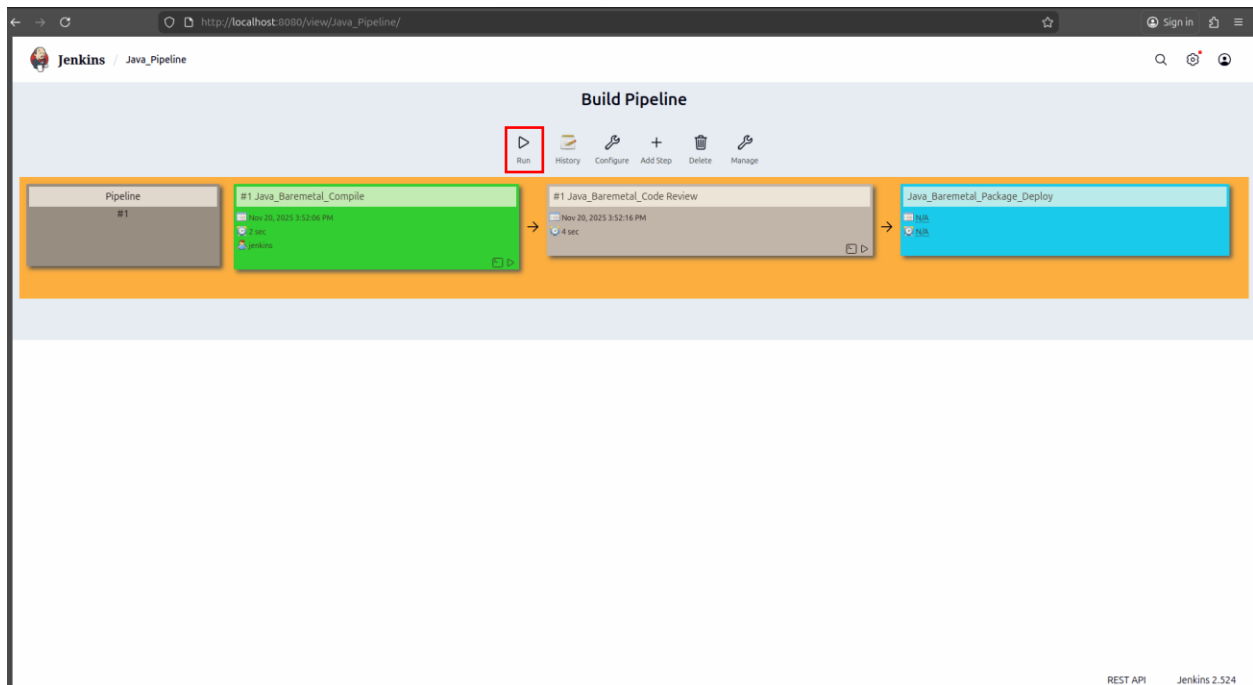
☐ Yes

☒ No

Display Options

Save Apply

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/](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
```

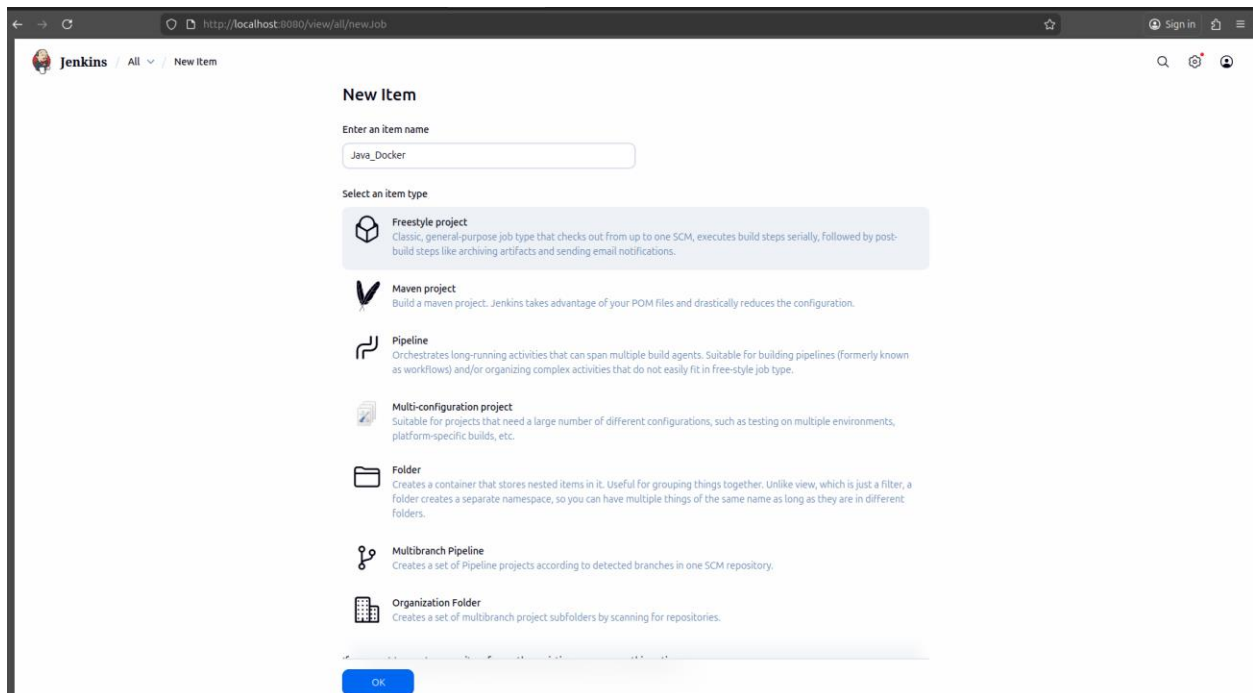
```
EXPOSE 8080
```

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

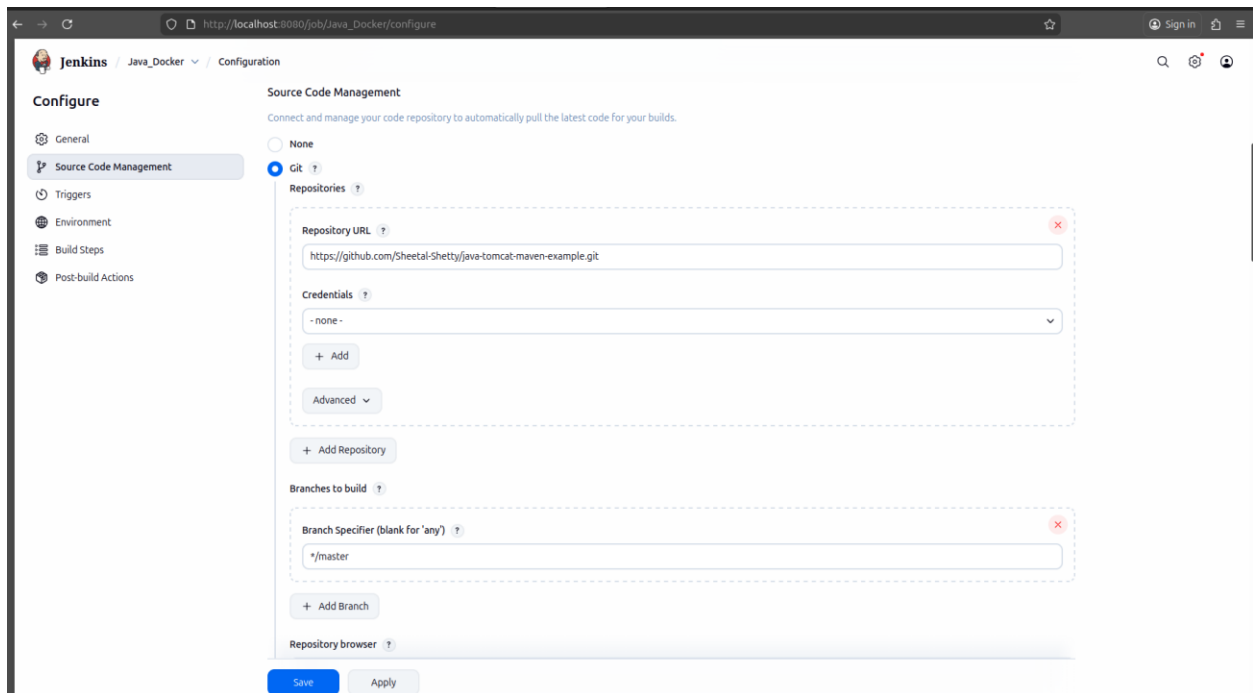
Pushed sir's code along with the Dockerfile to the new repository.

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

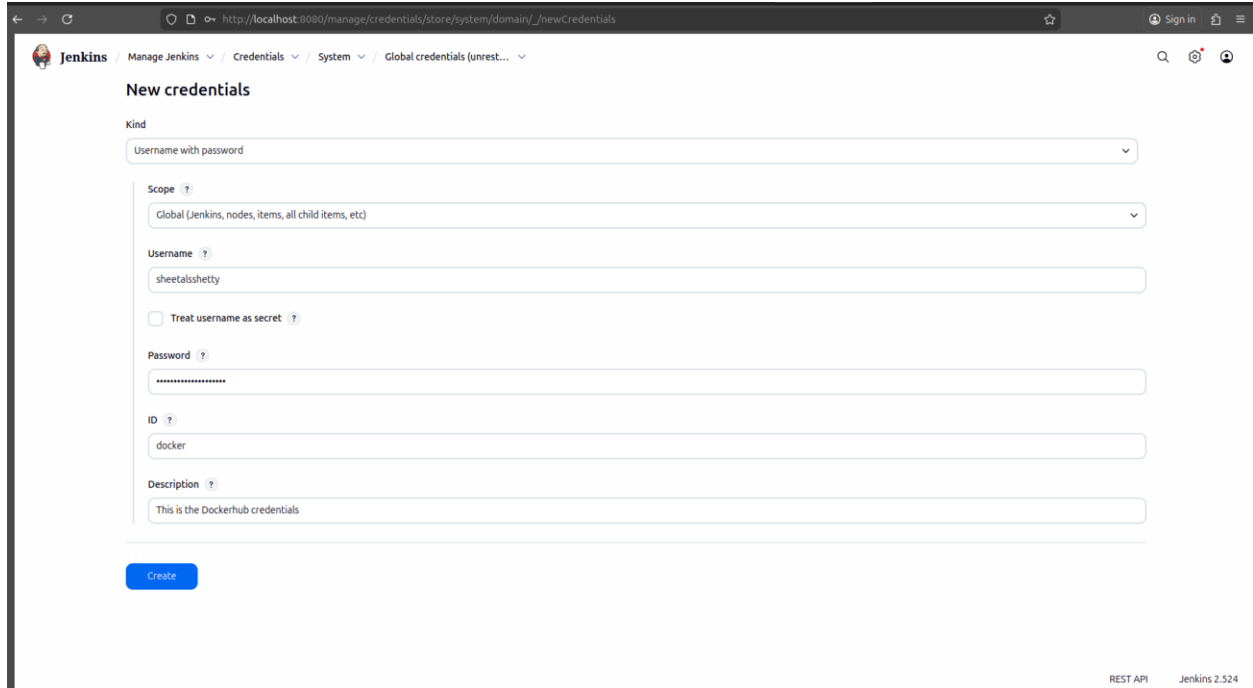
Create a **Freestyle Jenkins Project** and provide a name for the project, for example **Java_Docker**, then click **OK**.



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.

The screenshot shows the Jenkins configuration page for a job named 'Java_Docker'. The left sidebar contains a 'Configure' menu with options: General, Source Code Management, Triggers, Environment, Build Steps, and Post-build Actions. The 'Build Steps' section is active, showing a 'Docker Build and Publish' step. The configuration for this step includes: Repository Name 'sheetalshetty/java', Tag 'v3', Docker Host URI (empty), Server credentials (set to 'none'), Docker registry URL (empty), and Registry credentials (set to 'sheetalshetty/*****'). There are 'Save' and 'Apply' buttons at the bottom.

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.**”

The screenshot shows the Jenkins configuration page for the same 'Java_Docker' job, but now with an additional 'Execute Docker command' step. The 'Docker command' is set to 'Remove container(s)'. The 'Container ID(s)' field contains 'java3'. Below this, there are two checkboxes that are checked: 'Ignore if not found' and 'Force remove'. There are also 'Save' and 'Apply' buttons at the bottom.

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).

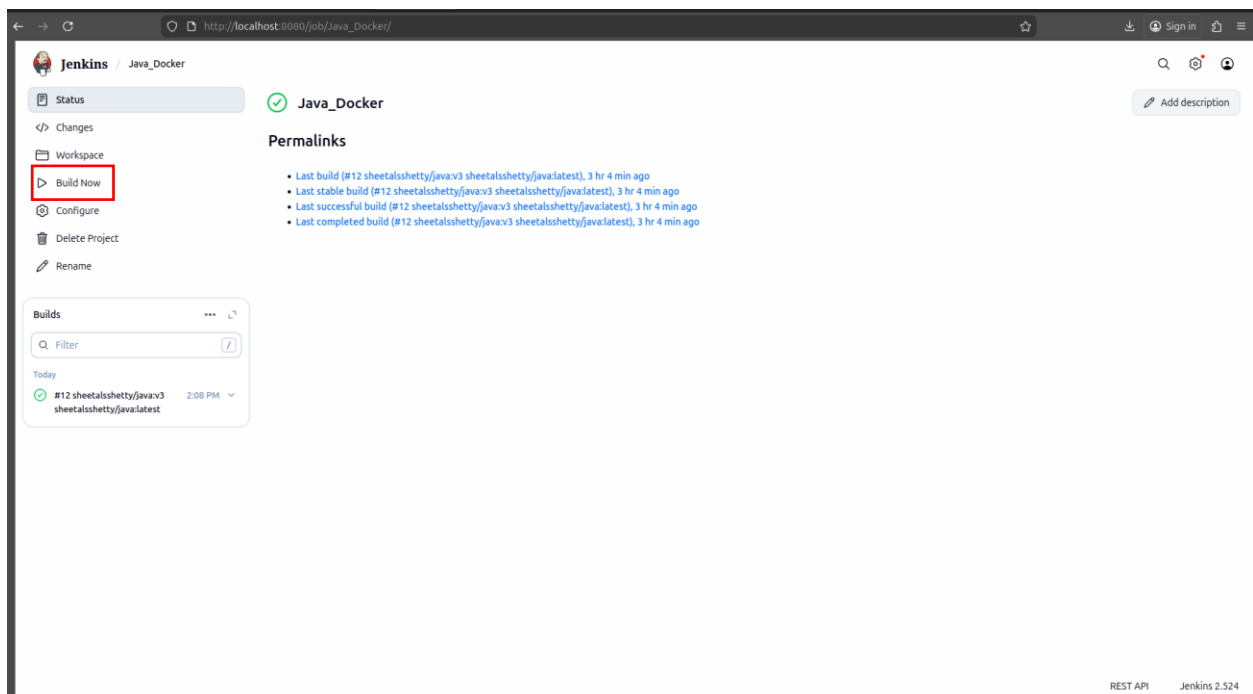
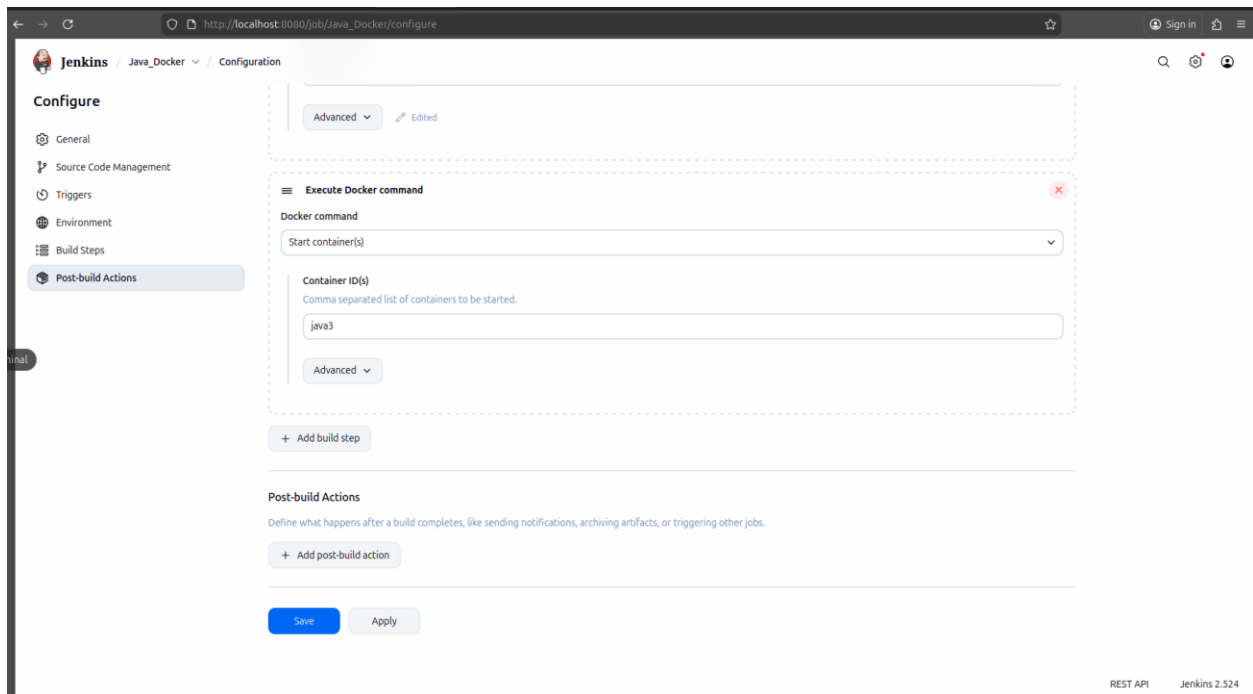
Click **Advanced**, select “**Publish all ports,**” and under **Port bindings**, map the ports in the format **Host_port:Container_port** (e.g., 8089:8080).

The image displays two screenshots of the Jenkins configuration interface for the 'Execute Docker command' build step.

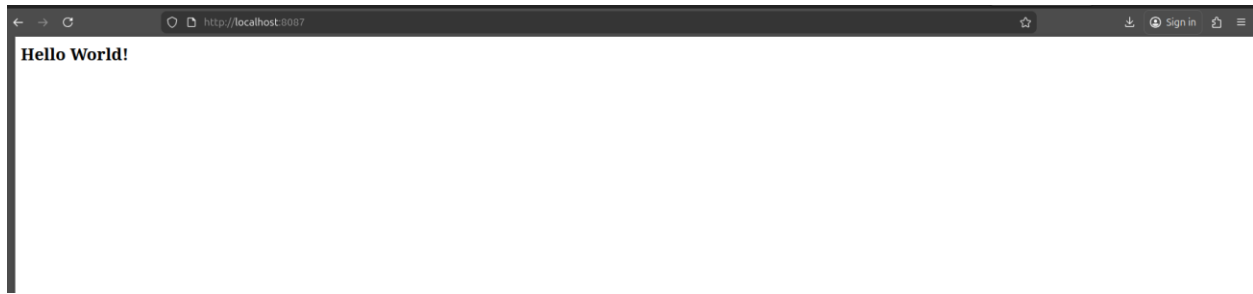
Top Screenshot: The 'Execute Docker command' section is expanded. The 'Docker command' dropdown is set to 'Create container'. The 'Image name' field contains 'sheetalsshetty/java:v3'. The 'Container name' field contains 'java3'. The 'Advanced' button is visible at the bottom of the section.

Bottom Screenshot: The 'Advanced' options are expanded. The 'Option --network:Network mode to use for this container' field is empty. The 'Publish all ports' checkbox is checked. The 'Port bindings' field contains '8087:8080'. The 'Bind mounts' field is empty. The 'Extended privileges' and 'always restart' checkboxes are unchecked.

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**.



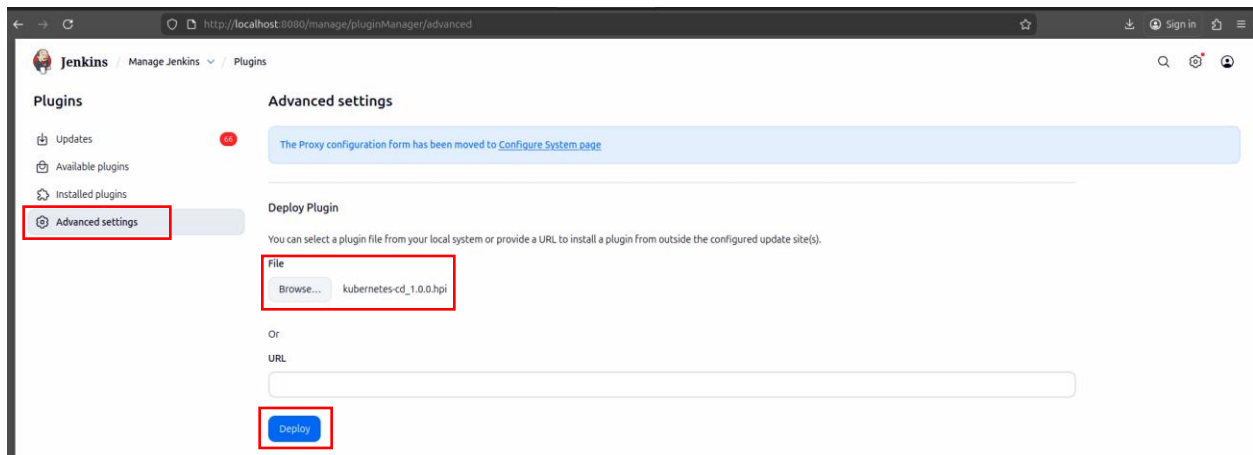
we can access the web page at <http://localhost:<Host Port>/>



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).

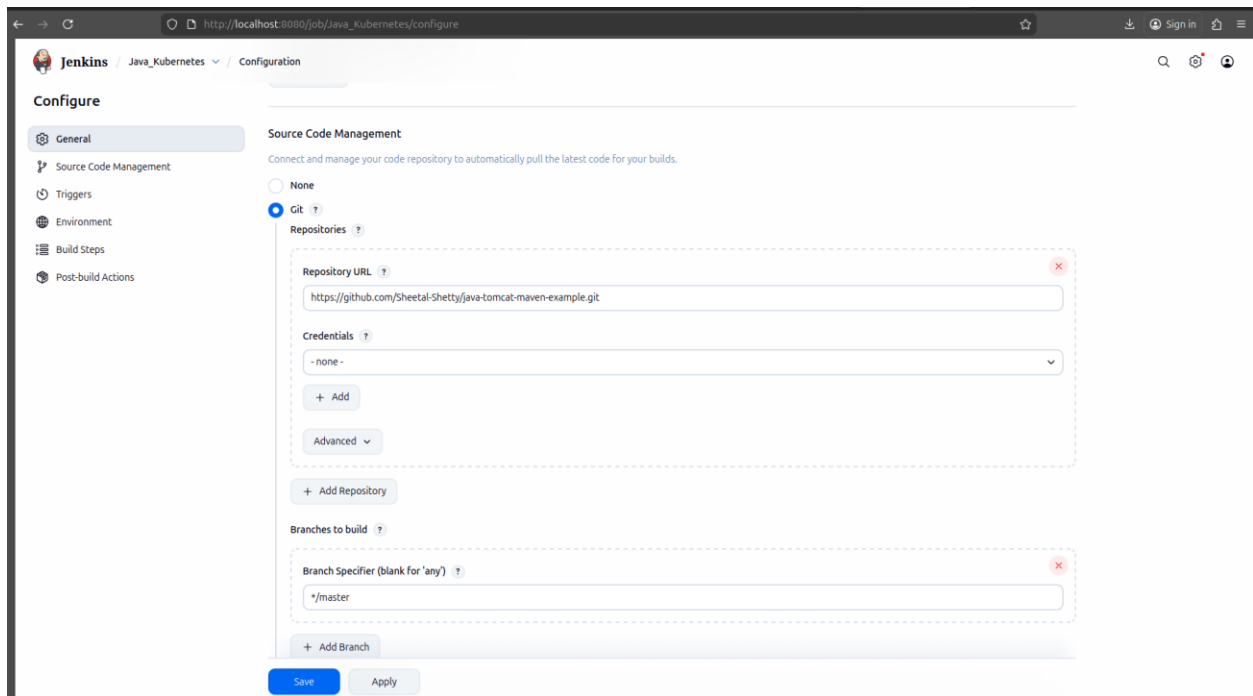
Go to **Manage Jenkins** → **Plugins** → **Advanced**, browse for the .hpi file, and click **Deploy**.



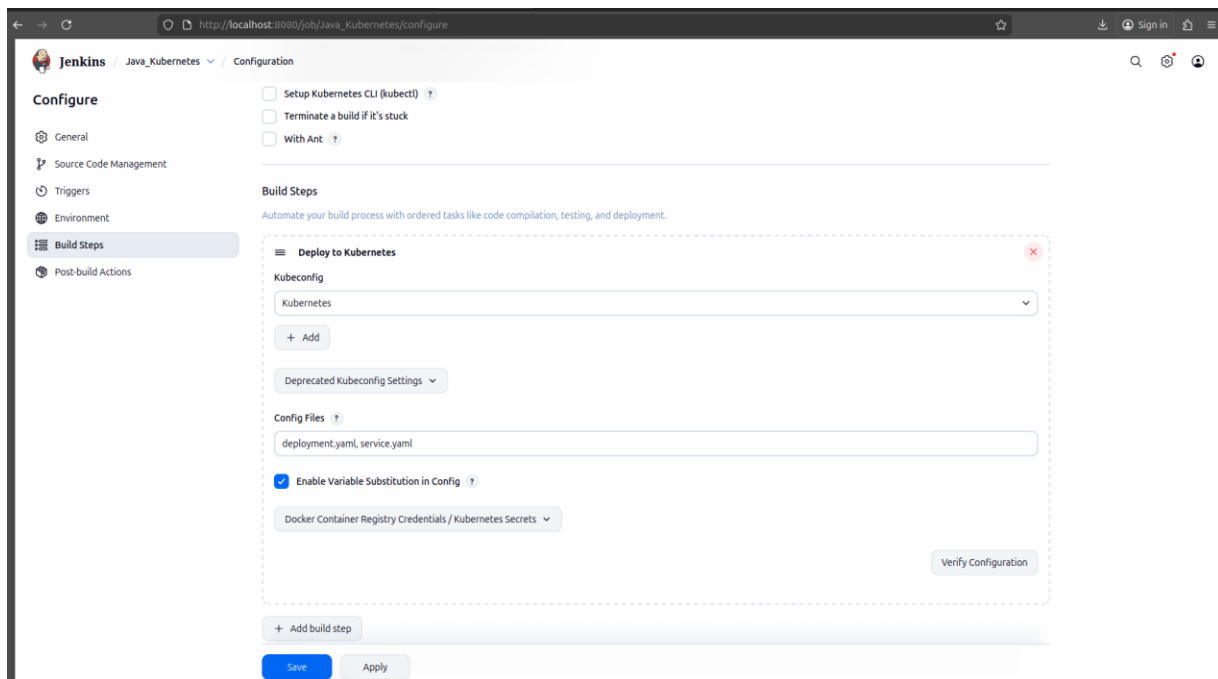
Created the Kubernetes manifest files (deployment.yaml and service.yaml) and added them to the GitHub repository.

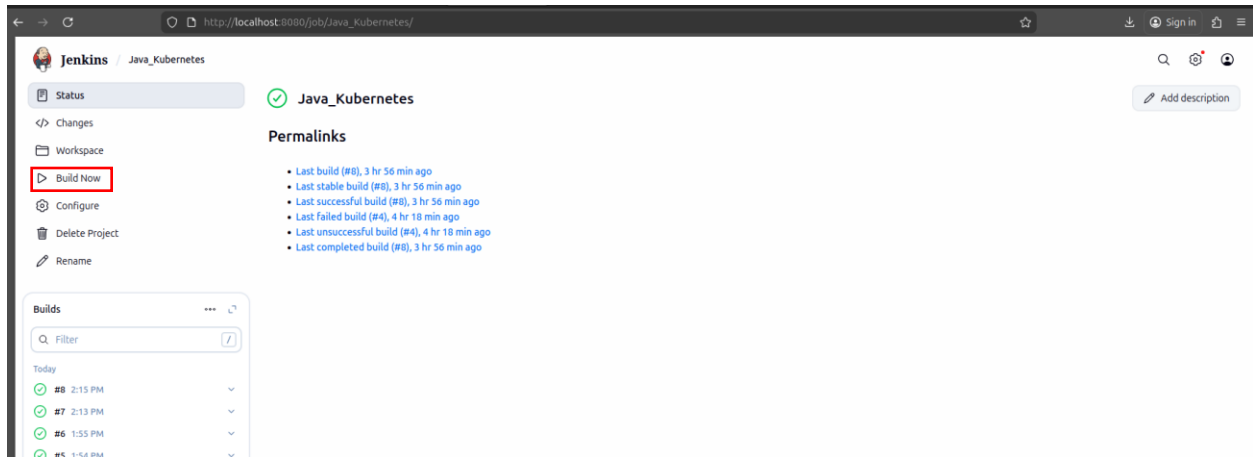
GitHub Repo: <https://github.com/Sheetal-Shetty/java-tomcat-maven-example.git>

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**.



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**.





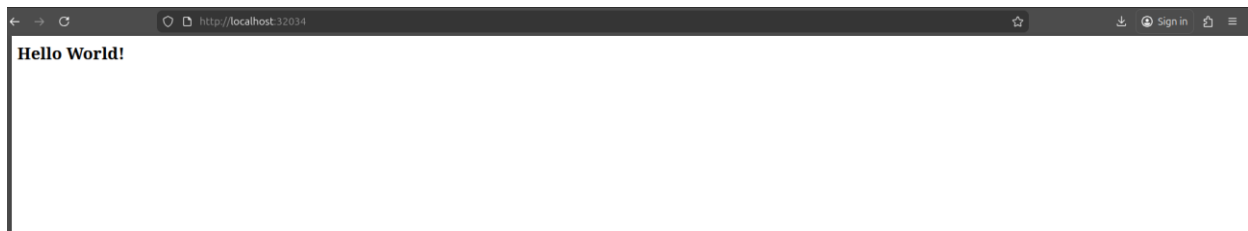
In terminal type `kubectl get svc`

```
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>`

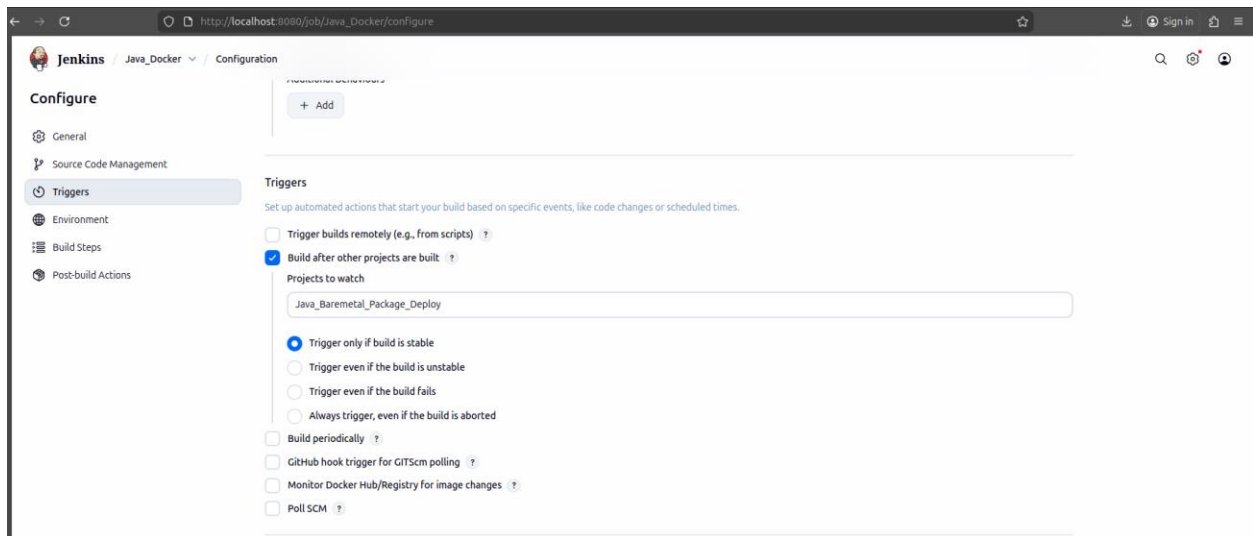
<http://localhost:32034/>



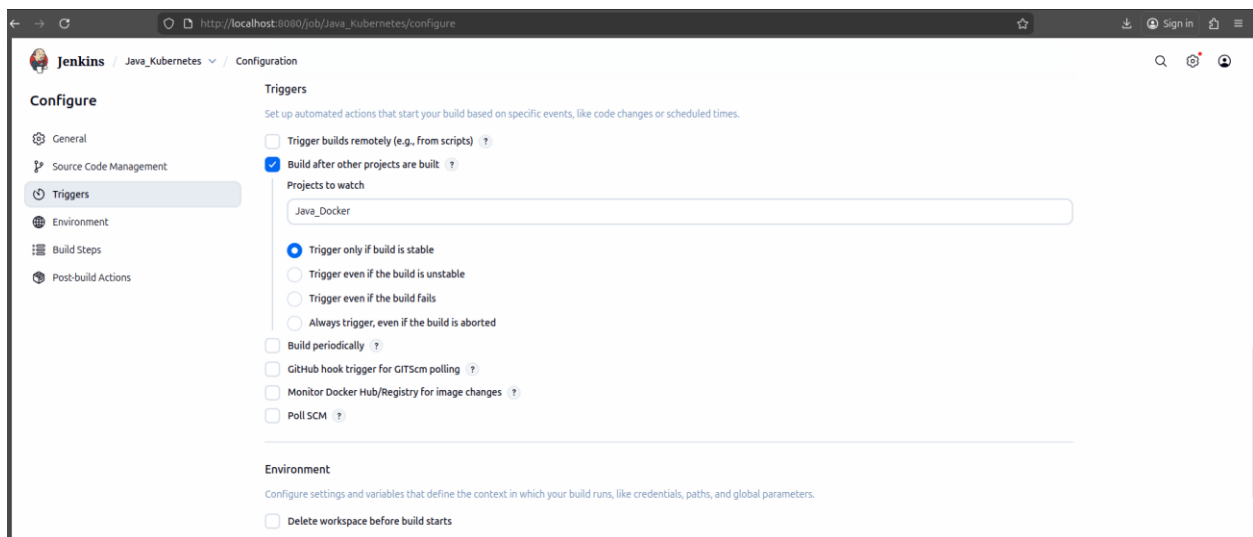
Create Pipeline for all these items (Baremetal->Docker->Kubernetes)

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).

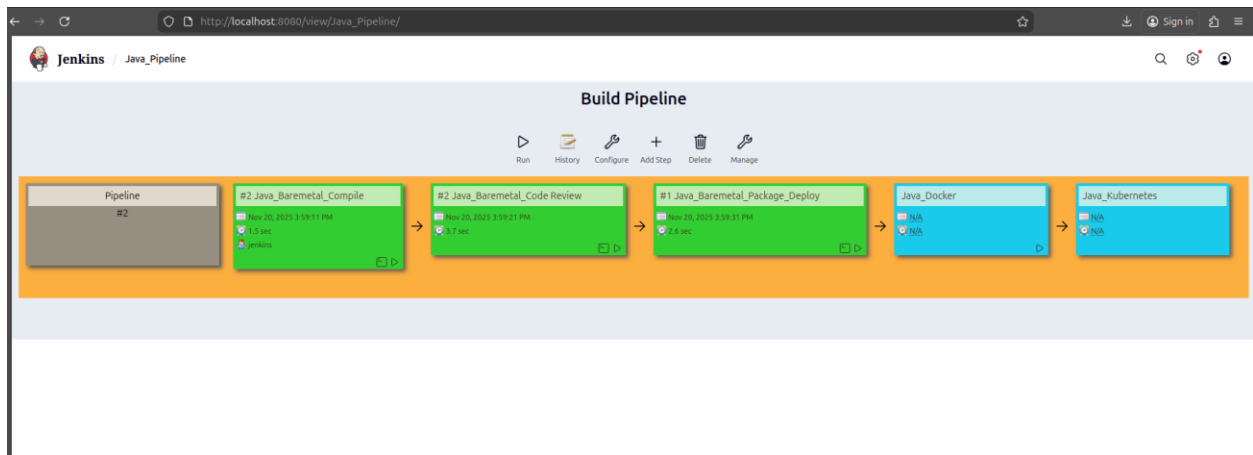
Go to the item named Java_Docker. Under Triggers, select “Build after other projects are built.” Then, under Projects to watch, select Java_Baremetal_Package_Deploy.



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.

