

## 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="" roles="manager-gui"/>
<user username="robot" password="" 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="" roles="tomcat"/>
<user username="both" password="" roles="tomcat,role1"/>
<user username="role1" password="" 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-C 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
-->

^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-6 Copy

```

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.

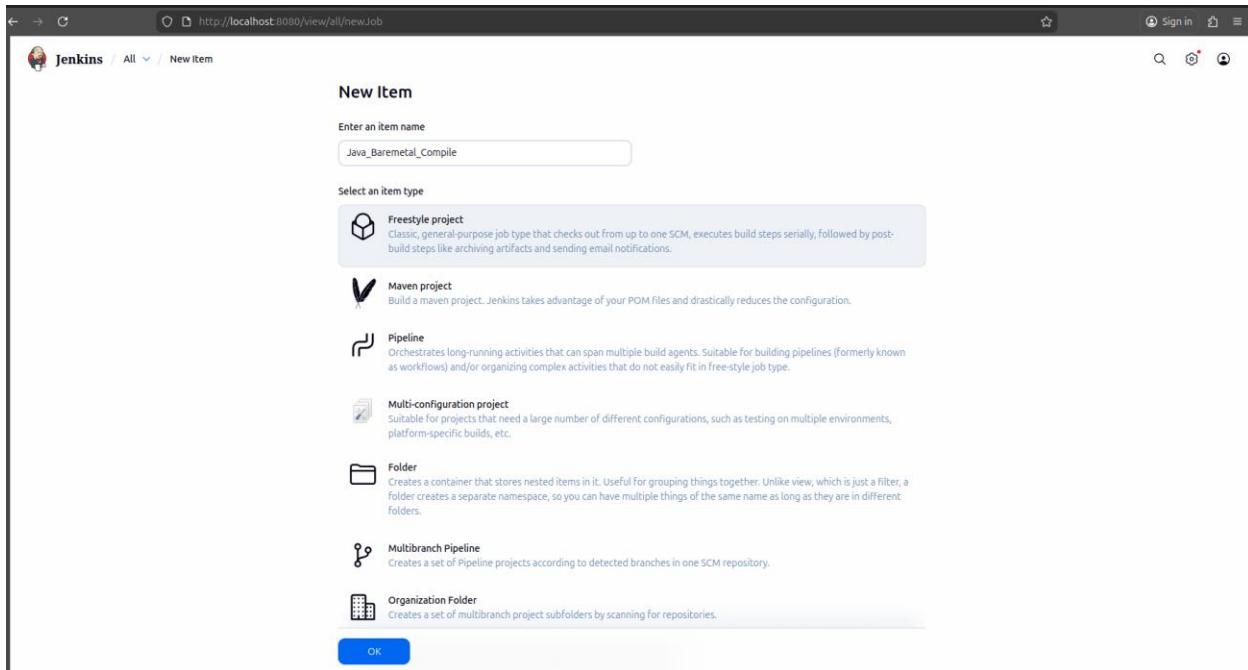
GitHub Repo: <https://github.com/sreepathysois/java-tomcat-maven-example.git>

Log in to Jenkins Server.

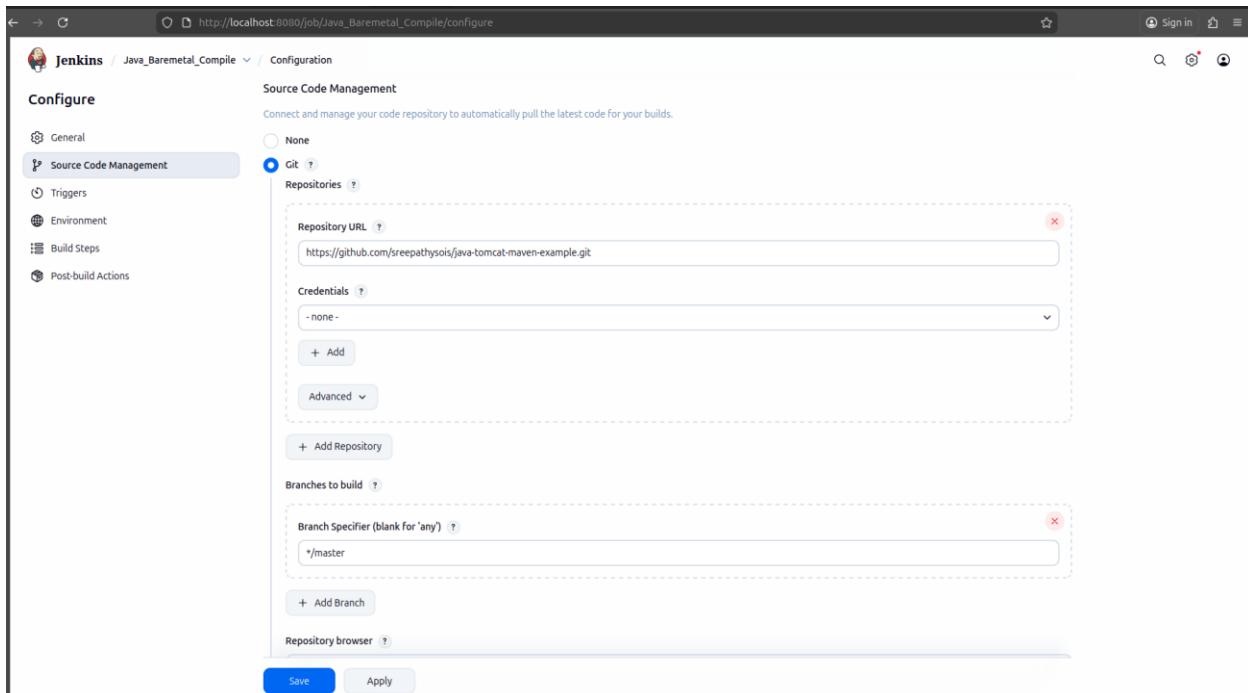
Step1: Compile

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.

Jenkins / Java\_Baremetal\_Compiled / Configuration

**Configure**

- General
- Source Code Management
- Triggers
- Environment
- Build Steps**
- Post-build Actions

**Build Steps**

Automate your build process with ordered tasks like code compilation, testing, and deployment.

**Invoke top-level Maven targets**

Maven Version: my\_maven

Goals: validate compile

**Post-build Actions**

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

+ Add post-build action

**Save** **Apply**

## Step2: Code review

Create a Freestyle Jenkins project. Provide a name for the project, for example: Java\_Baremetal\_CodeReview, then click OK.

Jenkins / All / New Item

**New Item**

Enter an item name: Java\_Baremetal\_Code Review

Select an item type:

- 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.

**OK**

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 'Java\_Baremetal\_Code Review' job. Under the 'Source Code Management' section, 'Git' is selected. The 'Repository URL' is set to `https://github.com/seepathyiso/java-tomcat-maven-example.git`. The 'Branch Specifier' is set to `*/master`. There are 'Save' and 'Apply' buttons at the bottom.

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.”

The screenshot shows the Jenkins configuration page for the 'Java\_Baremetal\_Code Review' job. Under the 'Triggers' section, 'Build after other projects are built' is selected. In the 'Projects to watch' list, 'Java\_Baremetal\_Compile' is listed. Under 'Trigger only if build is stable', the radio button is selected. There are 'Save' and 'Apply' buttons at the bottom.

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.

The screenshot shows the Jenkins job configuration page for 'Java\_Baremetal\_Code Review'. The 'Build Steps' section contains a step named 'Invoke top-level Maven targets' with 'my\_maven' as the Maven Version and goals '-P metrics pmd:pmd checkstyle:checkstyle findbugs:findbugs'. The 'Post-build Actions' section contains a step named 'Record compiler warnings and static analysis results' using 'Static Analysis Tools'.

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

The screenshot shows the Jenkins job configuration page with the 'Post-build Actions' section expanded. Under 'Record compiler warnings and static analysis results', the 'Tool' dropdown is set to 'PMD'. The 'Report File Pattern' field is empty. The 'skip symbolic links when searching for files' checkbox is unchecked. The 'Encoding of Report Files' field is empty. The 'Configure the context lines' field contains the value '3'. The 'Custom ID' field is empty. At the bottom are 'Save' and 'Apply' buttons.

Jenkins / Java\_Baremetal\_Code Review / 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**

Jenkins / Java\_Baremetal\_Code Review / 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**

The screenshot shows the Jenkins configuration interface for a job named 'Java\_Baremetal\_Code Review'. The left sidebar has a 'Configure' section with links for General, Source Code Management, Triggers, Environment, Build Steps, and Post-build Actions. The 'Post-build Actions' link is highlighted. The main panel shows a 'Post-build Actions' configuration section with three input fields: 'Custom ID', 'Custom Name', and 'Custom Icon', each with a placeholder and a question mark icon. Below these fields are three buttons: '+ Add Tool', 'Advanced', and '+ Add post-build action'. At the bottom are 'Save' and 'Apply' buttons. The status bar at the bottom right indicates 'REST API' and 'Jenkins 2.524'.

### 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**.

The screenshot shows the Jenkins 'New Item' dialog. The 'Enter an item name' field contains 'Java\_Baremetal\_Package\_Deploy'. The 'Select an item type' section lists several options: 'Freestyle project' (selected and highlighted), 'Maven project', 'Pipeline', 'Multi-configuration project', 'Folder', 'Multibranch Pipeline', and 'Organization Folder'. Each option has a small icon and a brief description. At the bottom is an 'OK' button.

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 job configuration page for 'Java\_Baremetal\_Package\_Deploy'. The 'Source Code Management' section is active, showing a 'Git' repository configuration. The 'Repository URL' is set to 'https://github.com/sreepathysois/java-tomcat-maven-example.git'. The 'Branch Specifier' is set to '\*/\*master'. There are 'Save' and 'Apply' buttons at the bottom.

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 job configuration page for 'Java\_Baremetal\_Package\_Deploy'. The 'Triggers' section is active, showing the 'Build after other projects are built' option selected. The 'Projects to watch' field contains 'Java\_Baremetal\_Code Review'. The 'Trigger only if build is stable' option is selected. There are 'Save' and 'Apply' buttons at the bottom.

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

The screenshot shows the Jenkins configuration page for a job named "Java\_Baremetal\_Package...". The left sidebar includes sections for General, Source Code Management, Triggers, Environment, Build Steps (which is selected and highlighted in grey), and Post-build Actions. The main content area is titled "Build Steps" and contains a sub-section "Invoke top-level Maven targets". Inside this section, the "Maven Version" dropdown is set to "my\_maven" and the "Goals" dropdown is set to "package". There is also an "Advanced" button and a "+ Add build step" link. Below this, the "Post-build Actions" section has a "+ Add post-build action" link. At the bottom are "Save" and "Apply" buttons. The footer indicates "REST API" and "Jenkins 2.524".

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

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

The screenshot shows the "Manage Jenkins" page. On the left, there are several navigation links: Appearance, Security, Users, Status Information, About Jenkins, Troubleshooting, Tools and Actions, and Global (which is highlighted with a red box). The main content area contains sections for System Information, System Log, Load Statistics, and Jenkins CLI. The "Credentials" link under the Security section is also highlighted with a red box.

Click on Global.

The screenshot shows the Jenkins 'Credentials' management page. At the top, there's a header with the Jenkins logo and navigation links. Below it, a table lists credentials with columns for ID, Name, and Kind. A second section titled 'Stores scoped to Jenkins' shows a list of stores with their domains. The 'System' store has '(global)' selected, which is highlighted with a red box.

| ID                            | Name                                       | Kind                                  |
|-------------------------------|--|---------------------------------------|
| tomcat_manager_script_cred_38 | msis/***** (tomcat_manager_script_cred_38) | Username with password                |
| 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) |

Stores scoped to Jenkins

| Store      | Domains  |
|------------|----------|
| System     | (global) |
| Kubernetes | (global) |

Click on Add Credential

The screenshot shows the Jenkins 'Global credentials (unrestricted)' page. It displays a list of credentials with columns for ID, Name, Kind, and Description. An 'Add Credentials' button is highlighted with a red box in the top right corner.

| 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) |                               |

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

New credentials

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**.

Configure

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\_38)

Tomcat URL: http://172.18.181.38:8081

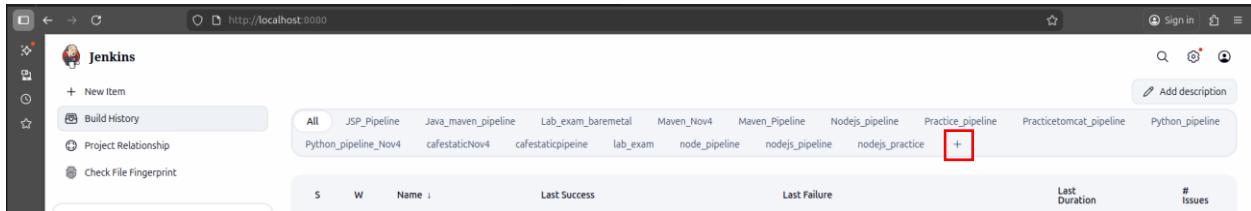
Advanced

+ Add Container

Save Apply

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

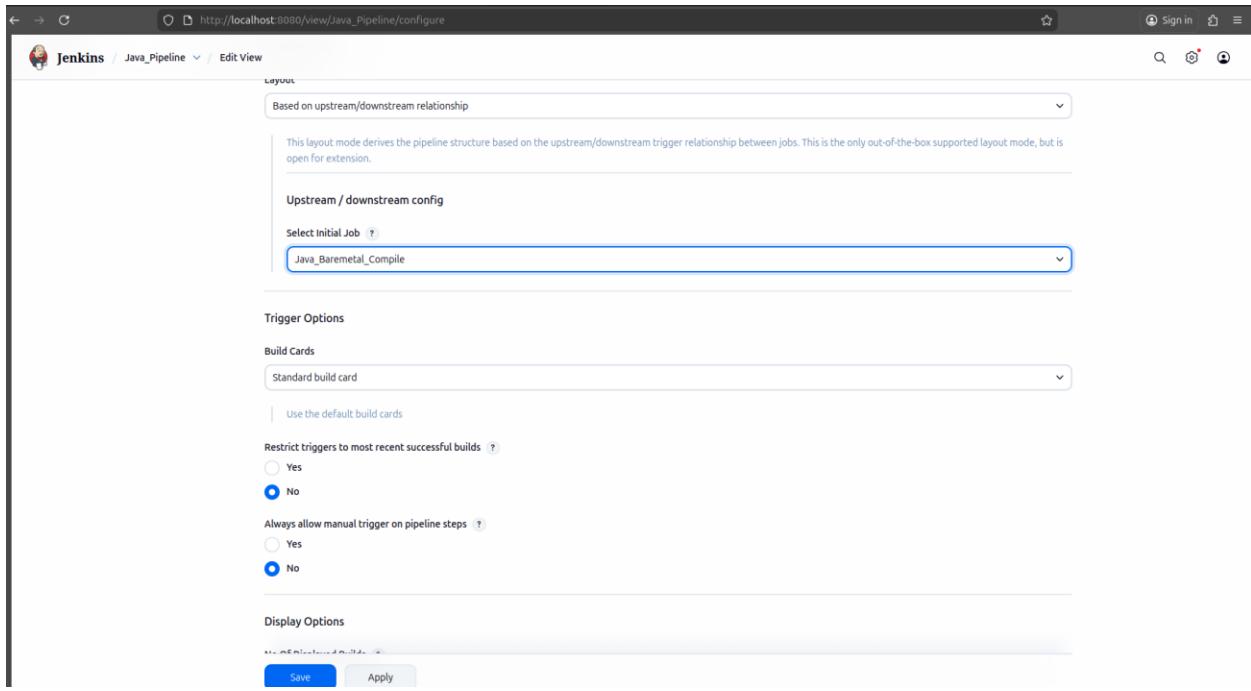
Navigate to the Jenkins home page and click on the “+” icon.



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

A screenshot of a web browser displaying the 'New view' configuration page at http://localhost:8080/newView. The page has a left sidebar with 'New Item', 'Build History', 'Project Relationship', and 'Check File Fingerprint'. The main area is titled 'New view' and contains a 'Name' input field with 'Java\_Pipeline' typed in. Below it is a 'Type' section with three options: 'Build Pipeline View' (selected, indicated by a blue radio button), 'List View' (unselected, indicated by an empty radio button), and 'My View' (unselected, indicated by an empty radio button). A note below the first option says: 'Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.' A 'Create' button is at the bottom of the form. On the right side of the page, there are links for 'REST API' and 'Jenkins 2.524'.

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**.



The screenshot shows the Jenkins Pipeline configuration page for a view named "Java\_Pipeline". The URL is [http://localhost:8080/view/Java\\_Pipeline/configure](http://localhost:8080/view/Java_Pipeline/configure). The page has a header with Jenkins logo, user sign-in, and search/filter icons.

**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

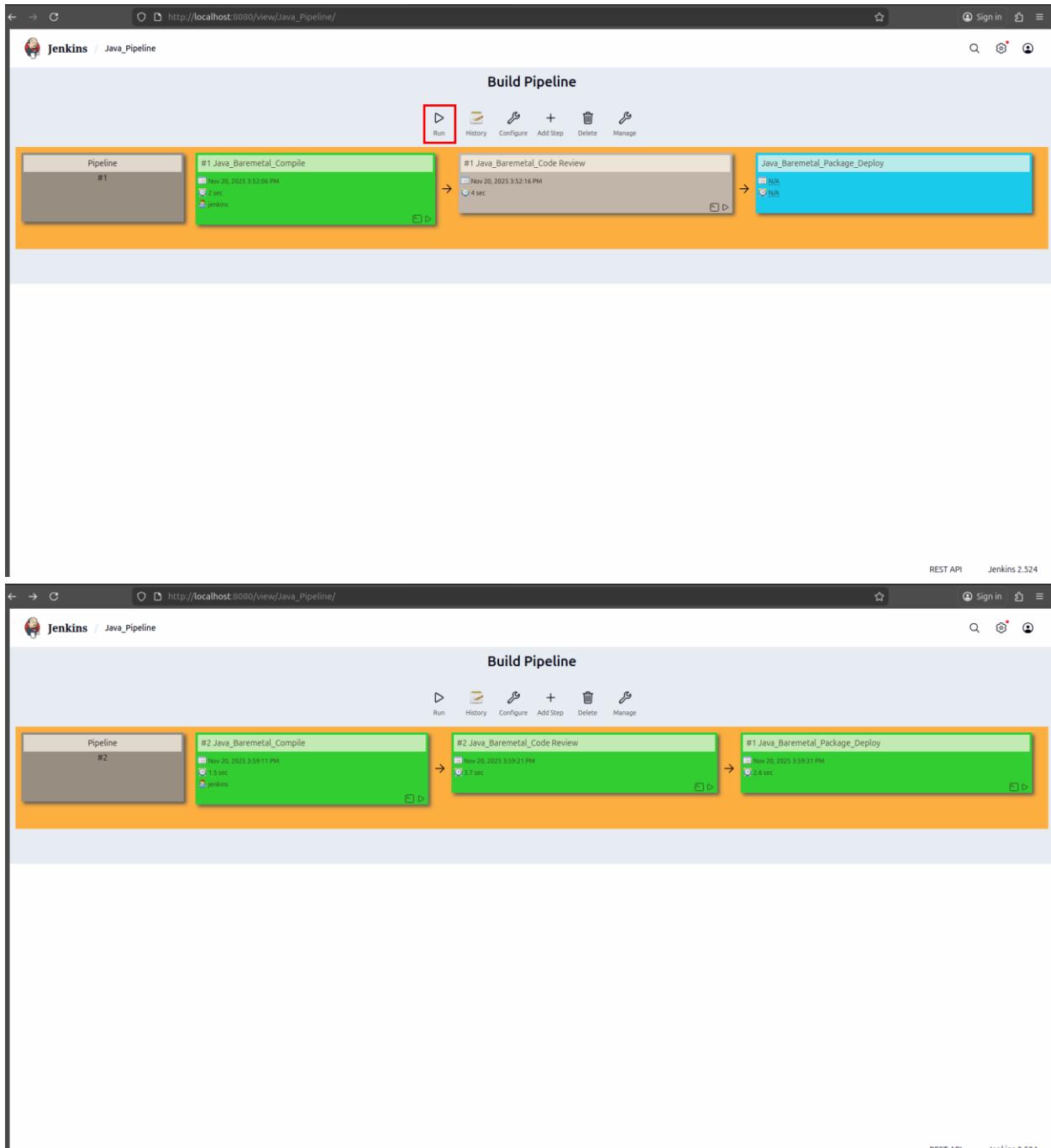
Restrict triggers to most recent successful builds: No

Always allow manual trigger on pipeline steps: 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`)



```
*****
```

## 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
```

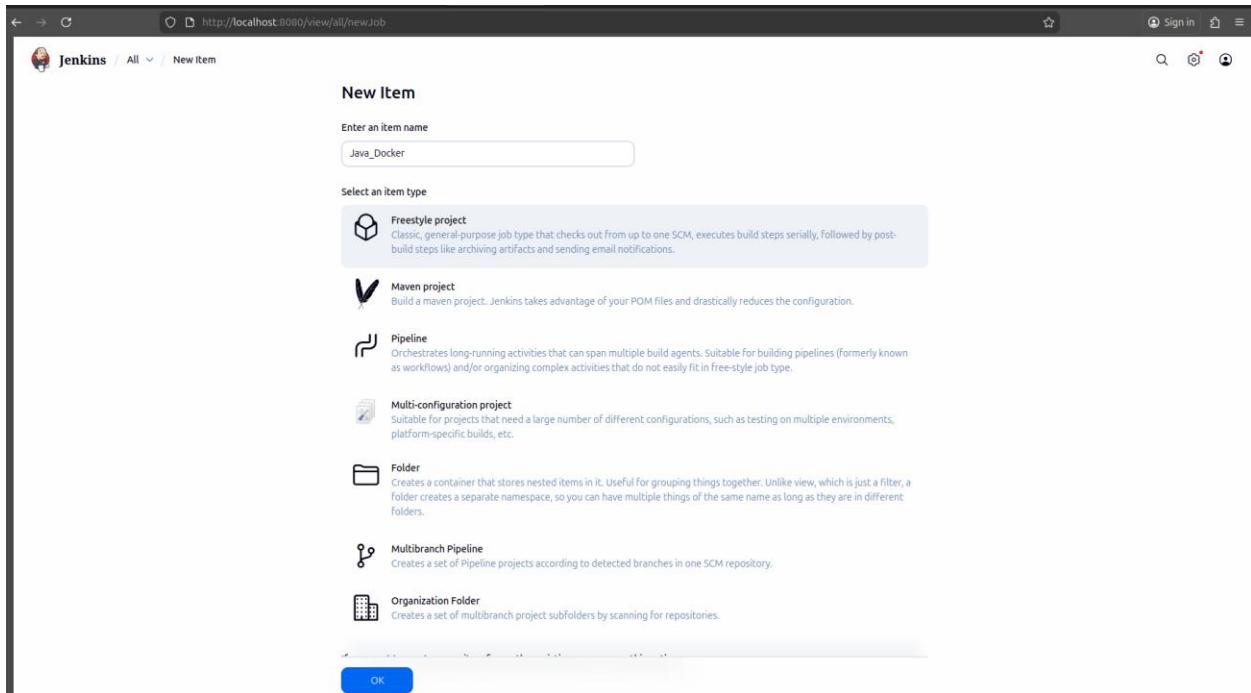
```
EXPOSE 8080
```

```
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.

The screenshot shows the Jenkins job configuration page for 'Java\_Docker'. The left sidebar has 'Configure' selected. Under 'Source Code Management', 'Git' is chosen, and the 'Repositories' section is expanded. It shows a repository URL of 'https://github.com/Sheetal-Shetty/java-tomcat-maven-example.git' and a branch specifier of '\*/\*master'. There are 'Save' and 'Apply' buttons at the bottom.

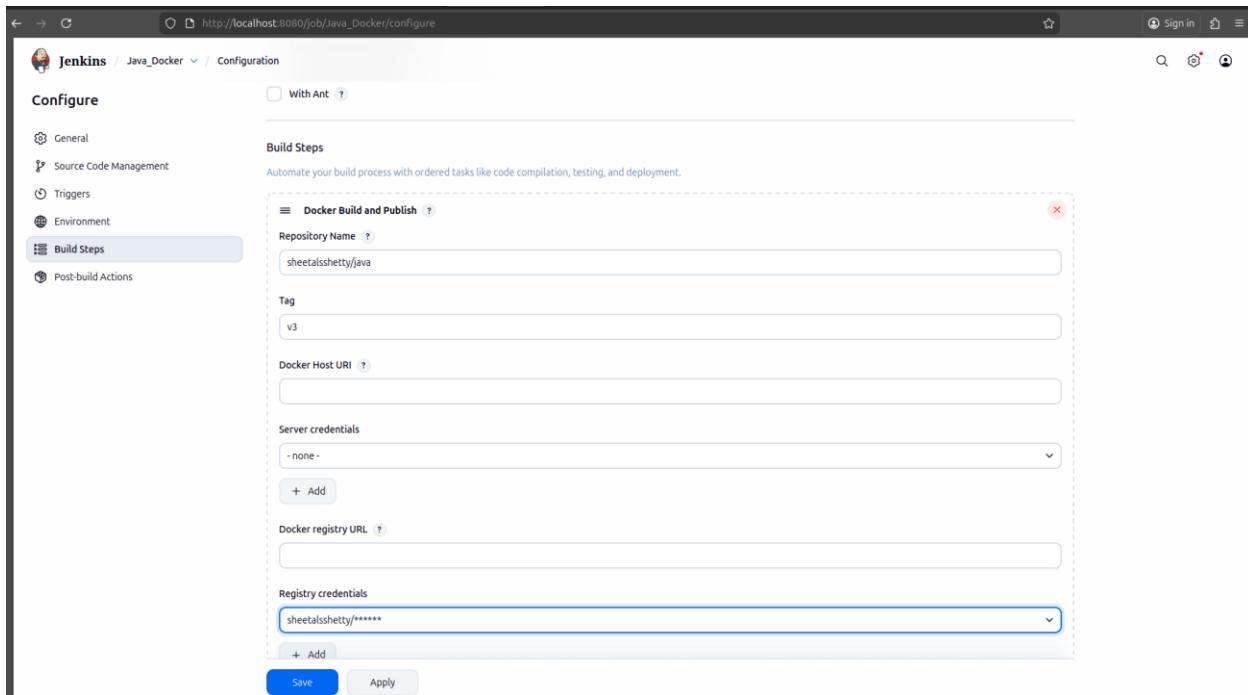
Add Docker credentials to Jenkins by navigating to **Manage Jenkins** → **Credentials**. Use your Docker Hub username as the **Username** and your Docker Hub access token as the **Password**.

The screenshot shows the 'New credentials' screen under 'Manage Jenkins' → 'Credentials'. A 'Username with password' credential is being created. The 'Kind' is set to 'Username with password', 'Scope' is 'Global', 'Username' is 'sheetalshetty', and 'Password' is a masked string. The 'ID' is 'docker' and the 'Description' is 'This is the Dockerhub credentials'. A 'Create' button is at the bottom.

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

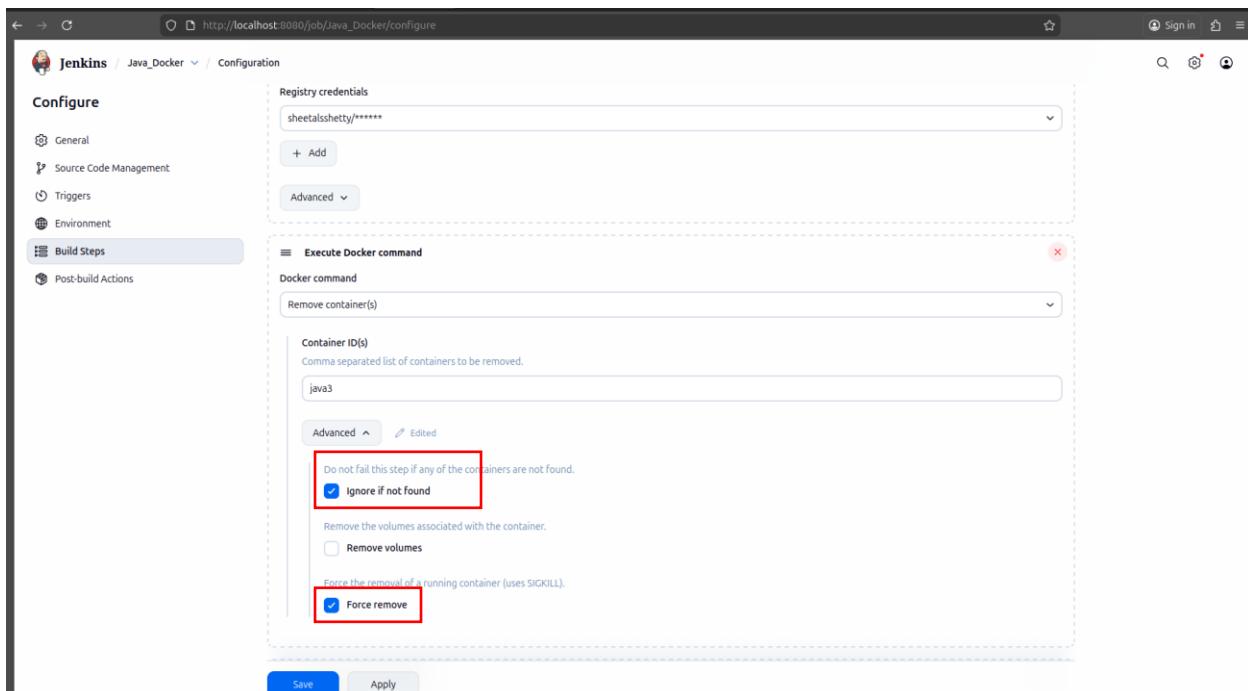
For **Repository Name**, enter <DockerHub\_Username>/<image\_name>.

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 job configuration page for 'Java\_Docker'. The 'Build Steps' section is expanded, showing a 'Docker Build and Publish' step. In this step, the 'Repository Name' is set to 'sheetalshetty/java', the 'Tag' is set to 'v3', and the 'Docker Host URI' field is empty. Under 'Server credentials', there is a dropdown menu currently set to '-none-' with a '+ Add' button below it. In the 'Registry credentials' section, a dropdown menu is open, showing 'sheetalshetty/\*\*\*\*' selected. At the bottom of the step, there are 'Save' and 'Apply' buttons.

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 job configuration page for 'Java\_Docker'. The 'Build Steps' section is expanded, showing an 'Execute Docker command' step. In this step, the 'Docker command' is set to 'Remove container(s)'. Below this, the 'Container ID(s)' field contains 'java3'. Under the 'Advanced' section, two checkboxes are highlighted with red boxes: 'Ignore if not found' and 'Force remove'. There is also a checkbox for 'Remove volumes' which is not highlighted. At the bottom of the step, there are 'Save' and 'Apply' buttons.

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 consists of two screenshots of the Jenkins configuration interface for a job named "Java\_Docker".

**Screenshot 1 (Top): Configuration - Post-build Actions**

- The left sidebar shows "Post-build Actions" selected.
- A modal dialog titled "Execute Docker command" is open, set to "Create container".
- Fields: "Image name" (sheetalsshetty/java:v3), "Container name" (java3).
- Buttons: "Advanced" (selected), "Save", "Apply".

**Screenshot 2 (Bottom): Configuration - Advanced**

- The left sidebar shows "Post-build Actions" selected.
- A modal dialog titled "Execute Docker command" is open, set to "Docker command".
- Fields: "Publish all ports" (checked), "Port bindings" (8087:8080).
- Buttons: "Check syntax", "Save", "Apply".

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**.

The screenshot shows the Jenkins job configuration page for 'Java\_Docker'. The 'Post-build Actions' section is selected. A new build step is being added, specifically an 'Execute Docker command' step. In the 'Docker command' dropdown, 'Start container(s)' is selected. The 'Container ID(s)' field contains 'java3'. The 'Save' button is visible at the bottom.

The screenshot shows the Jenkins job status page for 'Java\_Docker'. The 'Status' tab is selected. The 'Build Now' button is highlighted with a red box. The page also displays the history of builds, including the most recent build (#12) which was completed 3 hours and 4 minutes ago.

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

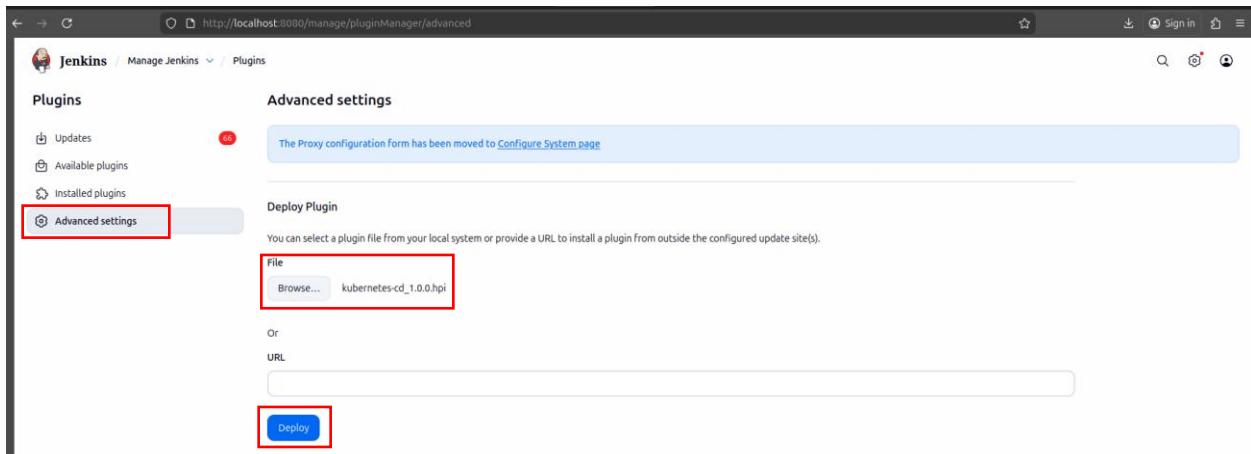


\*\*\*\*\*

### 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**.

The screenshot shows the Jenkins 'New credentials' configuration page. The 'Kind' dropdown is set to 'Kubernetes configuration (kubeconfig)'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'ID' field contains 'kubernetes'. The 'Description' field is empty. Under 'Kubeconfig', the 'Enter directly' radio button is selected, and a large text area displays a complex JSON/YAML configuration block. Below this, there are two additional options: 'From a File on the Jenkins master' and 'From a File on the Kubernetes master node', both of which are unselected. At the bottom left is a blue 'Create' button.

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

The screenshot shows the Jenkins 'New Item' creation page. In the 'Enter an item name' field, 'Java\_Kubernetes' is typed. The 'Select an item type' section shows several options: 'Freestyle project' (selected), 'Maven project', 'Pipeline', 'Multi-configuration project', 'Folder', 'Multibranch Pipeline', and 'Organization Folder'. Each option has a small icon and a brief description. At the bottom left is a blue 'OK' button.

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 job configuration page for a job named "Java\_Kubernetes". The left sidebar lists configuration sections: General, Source Code Management, Triggers, Environment, Build Steps, and Post-build Actions. The "Source Code Management" section is active, showing a "Git" repository configuration. It includes fields for "Repository URL" (https://github.com/Sheetal-Shetty/java-tomcat-maven-example.git), "Credentials" (set to "-none-"), and "Branches to build" (set to "\*/\*"). Buttons at the bottom are "Save" and "Apply".

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**.

The screenshot shows the Jenkins job configuration page for the same job. The "Build Steps" section is active, displaying a "Deploy to Kubernetes" step. Under "Kubeconfig", the value "Kubernetes" is selected. In the "Config Files" section, the values "deployment.yaml, service.yaml" are listed. A checked checkbox "Enable Variable Substitution in Config" is present. A "Verify Configuration" button is located at the bottom right of the step's configuration area. Buttons at the bottom are "+ Add build step", "Save", and "Apply".

The screenshot shows the Jenkins interface for the 'Java\_Kubernetes' project. The 'Build Now' button is highlighted with a red box. The page displays a list of recent builds:

- Last build (#8), 3 hr 56 min ago
- Last stable build (#8), 3 hr 56 min ago
- Last successful build (#8), 3 hr 56 min ago
- Last failed build (#4), 4 hr 18 min ago
- Last unsuccessful build (#4), 4 hr 18 min ago
- Last completed build (#8), 3 hr 56 min ago

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/)

<http://localhost:32034/>

The screenshot shows a web browser displaying the output of the Java application. The page content is "Hello World!".

\*\*\*\*\*

### 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).

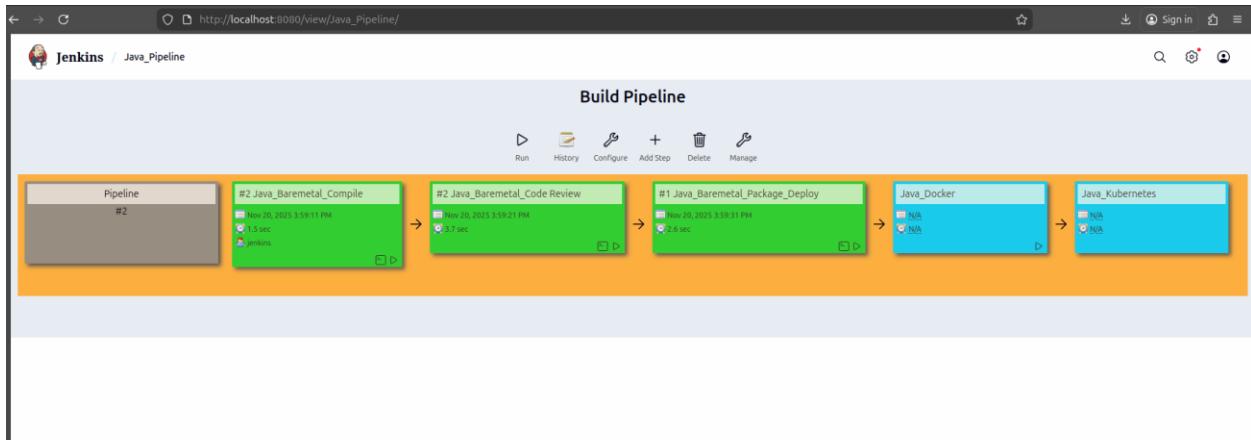
Go to the item named Java\_Docker. Under Triggers, select “Build after other projects are built.” Then, under Projects to watch, select Java\_Baremetal\_Package\_Deploy.

The screenshot shows the Jenkins configuration page for the 'Java\_Docker' job. The 'Triggers' section is selected. Under 'Build after other projects are built', the 'Build after other projects are built' checkbox is checked. In the 'Projects to watch' dropdown, 'Java\_Baremetal\_Package\_Deploy' is selected. Below this, the 'Trigger only if build is stable' radio button is selected.

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.

The screenshot shows the Jenkins configuration page for the 'Java\_Kubernetes' job. The 'Triggers' section is selected. Under 'Build after other projects are built', the 'Build after other projects are built' checkbox is checked. In the 'Projects to watch' dropdown, 'Java\_Docker' is selected. Below this, the 'Trigger only if build is stable' radio button is selected.

Now, the pipeline creation is complete.



Run the Pipeline.

