**DEVOPS Theory Assignment**

**Setup a Jenkins and Docker on Ubuntu 22.04**

**Jenkins Installation**

**To set up Jenkins on Ubuntu 22.04, we first need to install Java as it's a prerequisite for Jenkins:**

sudo apt-get update

sudo apt install openjdk-17-jdk

**After installing Java, we'll add the Jenkins repository and install Jenkins:**

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc <https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key>

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" <https://pkg.jenkins.io/debian-stable> binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install Jenkins

**Once installed, check the Jenkins service status:**

sudo systemctl status jenkins.service

**To access Jenkins for the first time, you'll need the initial admin password:**

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

**This command displays a one-time auto-generated password created during Jenkins installation that you'll use for the initial setup.**

**Open up any browser on your ubuntu system and hit the URL localhost:8080 or your systems IP address:8080 example 172.16.50.85:8080 to access Jenkins.**

**Docker Installation**

**For Docker installation, run the following commands:**

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable docker

sudo systemctl start docker

**Verify Docker is running properly:**

sudo systemctl status docker

docker –version

**Apache and PHP Installation**

**To set up Apache web server with PHP support:**

sudo apt update

sudo apt install apache2 -y

sudo systemctl status apache2

sudo apt install php libapache2-mod-php php-mysql -y

sudo systemctl restart apache2

**Tomcat Installation**

**For Tomcat 9 installation and configuration:**

sudo apt-get update

sudo apt install tomcat9

sudo apt install tomcat9-admin

**Check Tomcat service status:**

sudo systemctl status tomcat9.service

**Configure Tomcat by editing the server.xml file:**

sudo nano /etc/tomcat9/server.xml

**Add or modify the connector configuration:**

<Connector **port="8000"** protocol="HTTP/1.1"**change the port number to 8000 or any other**

**address="172.16.50.85" Add the system IP Address**

connectionTimeout="20000"

redirectPort="8443" />

**Set up a Tomcat user for Jenkins access:**

sudo nano /etc/tomcat9/tomcat-users.xml

**Add the following user configuration:**

<user username="**jenkins**" password="**Tomcat@123**" roles="**manager-script**"/>

**Add the above line in the last line of the tomcat-users.xml file**

**Restart and check the status of the tomcat server**

sudo systemctl restart tomcat9.service

sudo systemctl status tomcat9.service

**If required Install JDK and restart Tomcat:**

sudo apt install default-jdk –y

sudo systemctl restart tomcat9.service

sudo systemctl status tomcat9.service

# **Java-Tomcat-Maven-Example CI-CD Pipeline**

**Prerequisites**

* Jenkins server up and running
* Required plugins installed: Git, Maven Integration, Docker, Kubernetes, and Deploy to Container (for Tomcat)
* Access to a Tomcat server (local or remote) for deployments
* Docker and Kubernetes cluster access if those steps are needed



**Pipeline Steps Overview**

* Compile
* Review
* Deploy
* Docker Build and Push, Run
* Kubernetes Deploy

# **Java-Tomcat-Maven-Example-Compile**

**Source Code Management**

Select Git.

Enter the repo URL: https://github.com/Nikhil-SG/java-tomcat-maven-example.git

Set branch to build: \*/master.



**Build Steps**

Add a build step: Invoke top-level Maven targets.

Choose Maven version: MyMaven.

Set goals: clean compile.

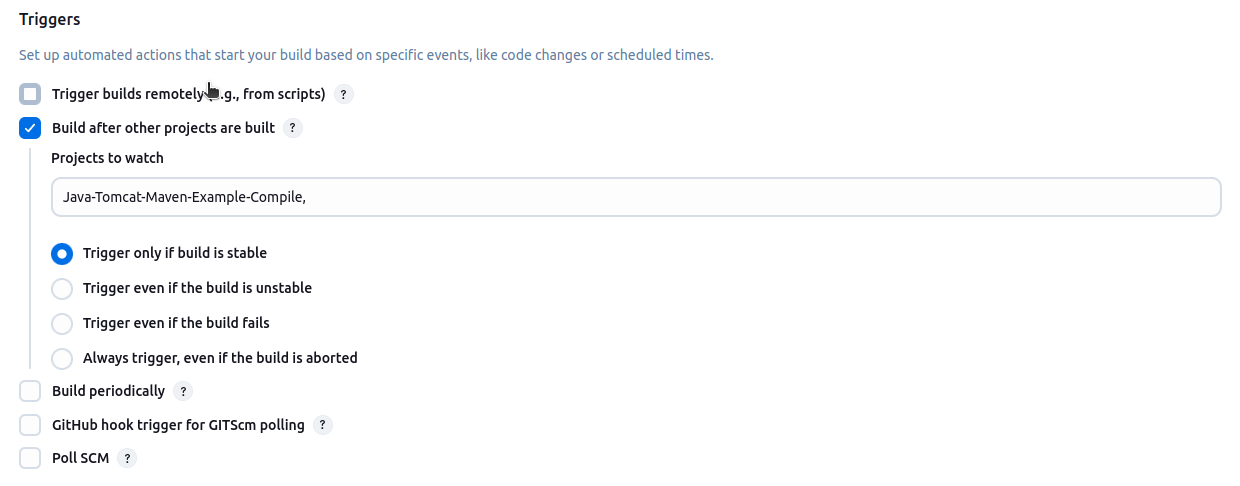


**This setup will pull the latest code from the master branch and compile it using Maven.**

# **Java-Tomcat-Maven-Example-Review**

**Build Triggers**

This job will start automatically after the Java-Tomcat-Maven-Example-Compile job finishes and only if that build was successful.



**Build Steps**

Uses Maven (MyMaven) to run code quality checks.

Maven goals:

pmd:pmd findbugs:findbugs checkstyle:checkstyle validate

This runs PMD, FindBugs, and Checkstyle tools to check for code issues and validate the project.



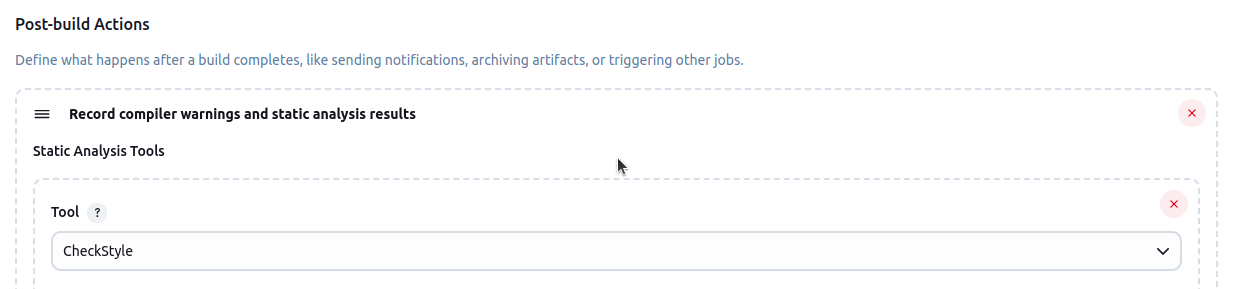
**Post-build Actions**

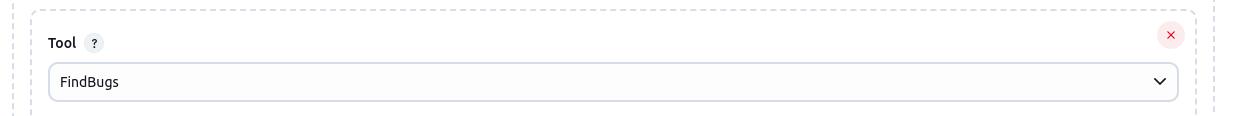
After the build, Jenkins collects and records results from static analysis tools:

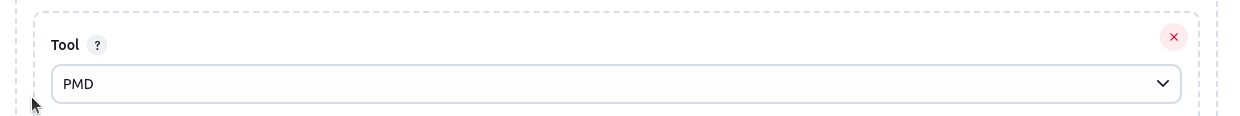
CheckStyle (checks code style)

FindBugs (finds the bugs in the code)

PMD (finds code problems)





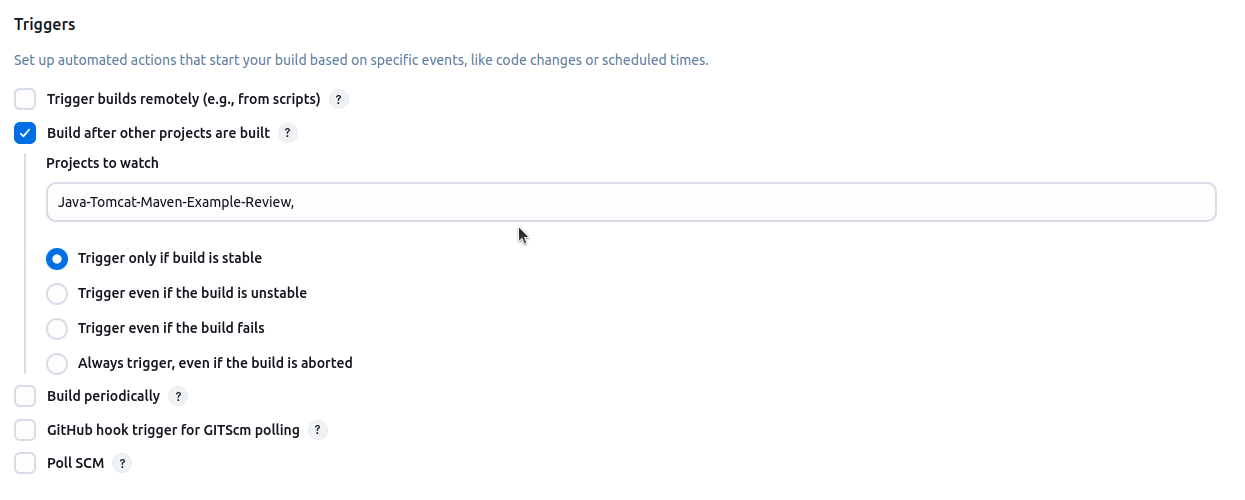


# **This job runs code quality tools (PMD, FindBugs, Checkstyle) after a successful compile, and then records their results for you to review.**

# **Java-Tomcat-Maven-Example-Deploy**

**Build Triggers**

This job runs automatically after the Java-Tomcat-Maven-Example-Review job finishes successfully.



**Build Steps:**

Maven Build:

Uses Maven (MyMaven) to run install package goals.

This builds the project and creates a WAR file ready for deployment



**Post-build Actions**

**Deploy WAR to Tomcat:**

Finds the WAR file (\*\*/\*.war) and deploys it to a remote Tomcat server.

Sets the application context path as MyMavenApp.

Uses saved Jenkins credentials for Tomcat access.

Deploys to the Tomcat server at http://172.16.50.85:8081



# **This job builds the project and automatically deploys the generated WAR file to a remote Tomcat**

# **server after the review stage is successful.**

# **Java-Tomcat-Maven-Example-Docker**

**Build Triggers**

This job runs automatically after the Java-Tomcat-Maven-Example-Deploy job finishes

successfully.

**Build Steps:**

**Maven Build**

Uses Maven (MyMaven) with the goal:

**package**

**This command packages your Java app into a WAR file.**

**Docker Build and Publish**

Builds a Docker image using the generated WAR file.

Repository name: sgnikhil/javatomcatexample2

Tag: v2

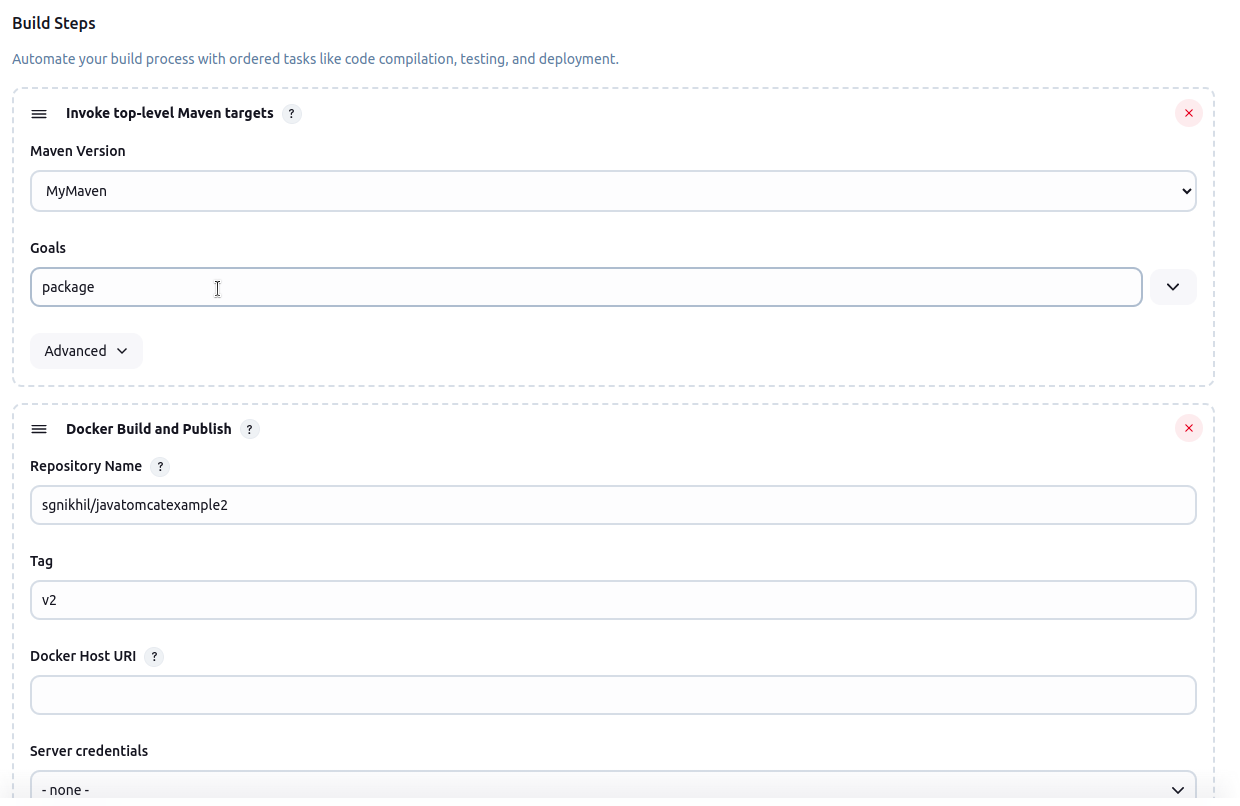
**Uses Docker credentials to push the image to Docker Hub.**

**Execute Shell**

**Runs a shell command to start a container from the new image**

**This job packages your app, builds a Docker image, pushes it to Docker Hub, and then runs the**

**app in a Docker container.**



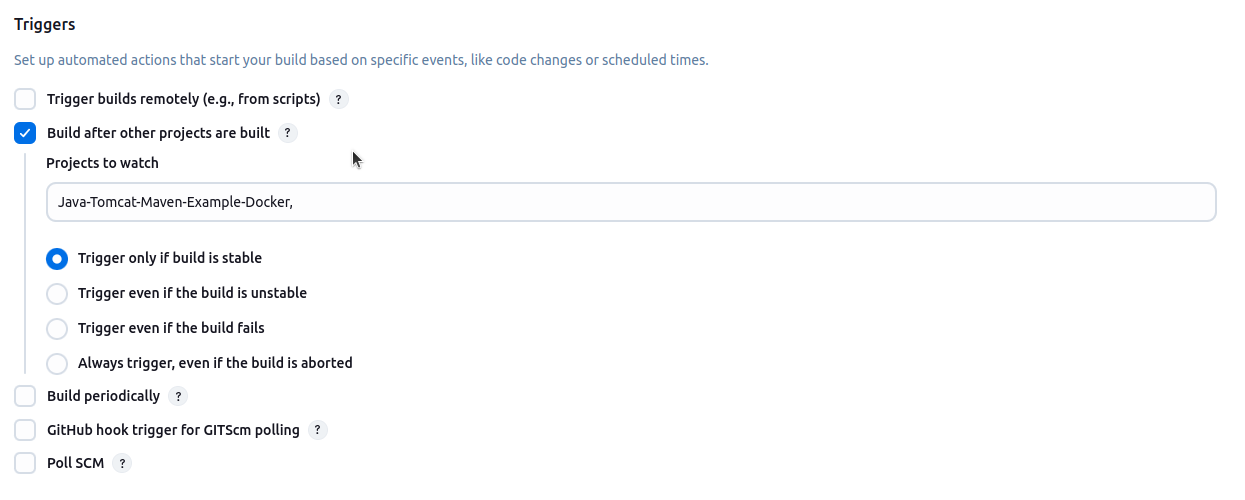


# **Java-Tomcat-Maven-Example-Kube**

**Build Triggers**

This job runs automatically after the Java-Tomcat-Maven-Example-Docker job finishes

successfully



**Build Steps:**

**Deploy to Kubernetes Build Step**

**Kubeconfig:**

Select the Kubernetes cluster credentials (here, "**47\_49 (Master 47 Slave 49)**") so Jenkins can connect to your Kubernetes cluster.

**Config Files:**

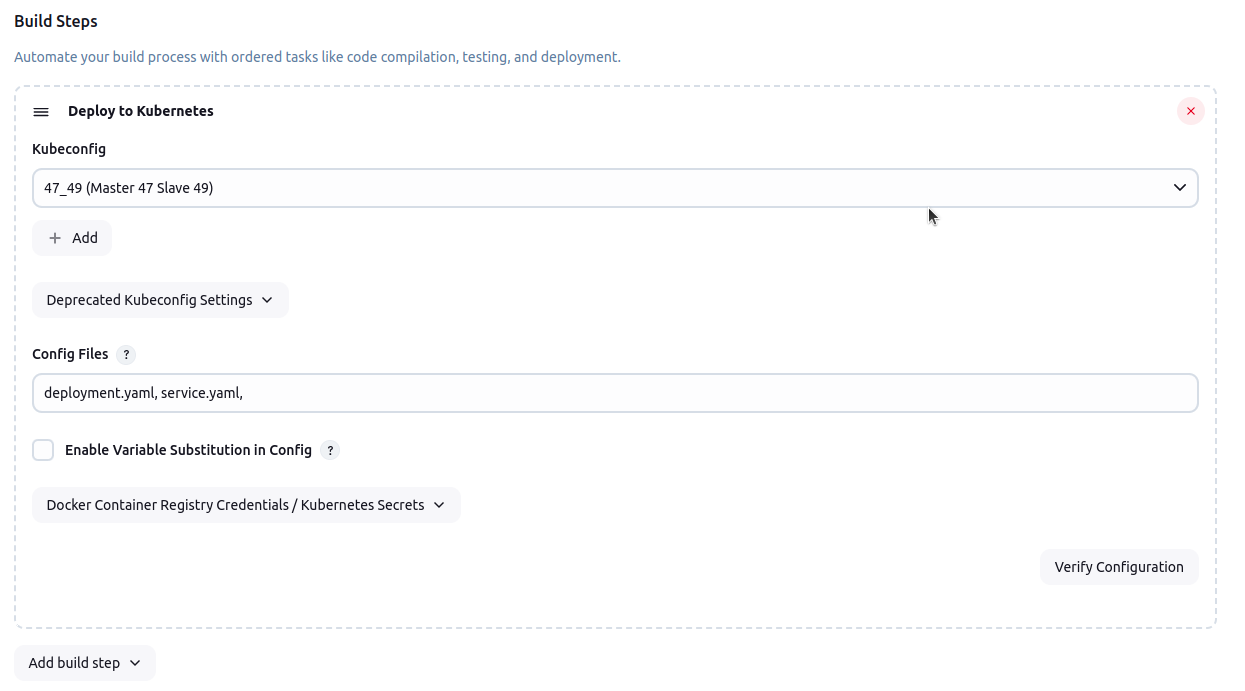
Lists the Kubernetes YAML files to apply:

deployment.yaml, service.yaml

These files define how your app will run and be exposed in the cluster.

**Docker Container Registry Credentials / Kubernetes Secrets:**

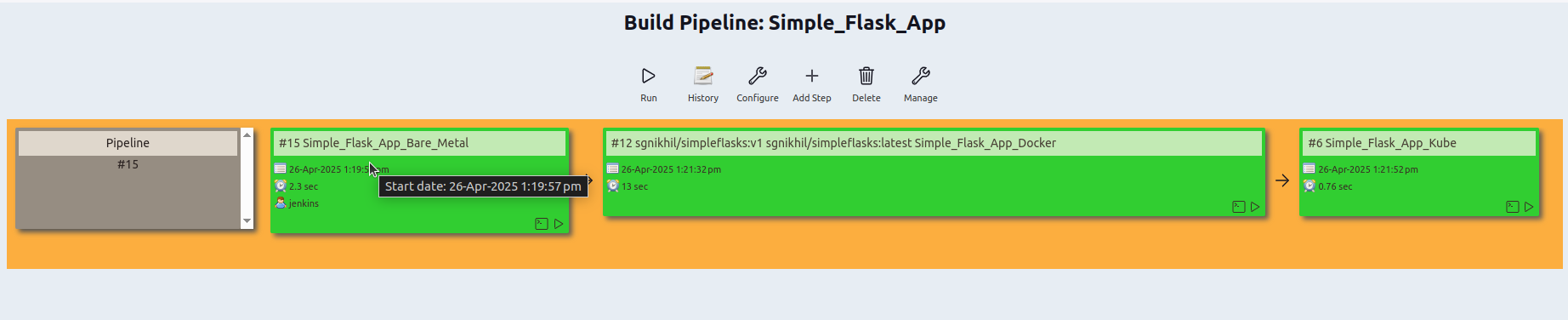
Let Jenkins use stored credentials if your deployment needs to pull images from a private Docker registry.



# **Simple\_Flask\_App CI-CD Pipeline**

**Prerequisites**

* Jenkins server is up and running
* Required plugins installed: Git, Python, Docker, Kubernetes
* Docker and Kubernetes cluster access



**Pipeline Steps Overview**

* Bare MetalDeploy
* Docker Build and Push, Run
* Kubernetes Deploy

# **Simple\_Flask\_App\_Bare\_Metal**

**Source Code Management**

Select Git.

Enter the repo URL: https://github.com/Nikhil-SG/Simple-Flask-App-Docker.git

Set branch to build: \*/main.



**Build Steps:**

Checks Python version.

Creates a virtual environment called FLasks.

Activates the virtual environment.

Installs all dependencies from requirements.txt.

Starts the Flask app in the background with python3 app.py&



# **Jenkins downloads your Flask app code, sets up Python, installs needed packages, and runs the**

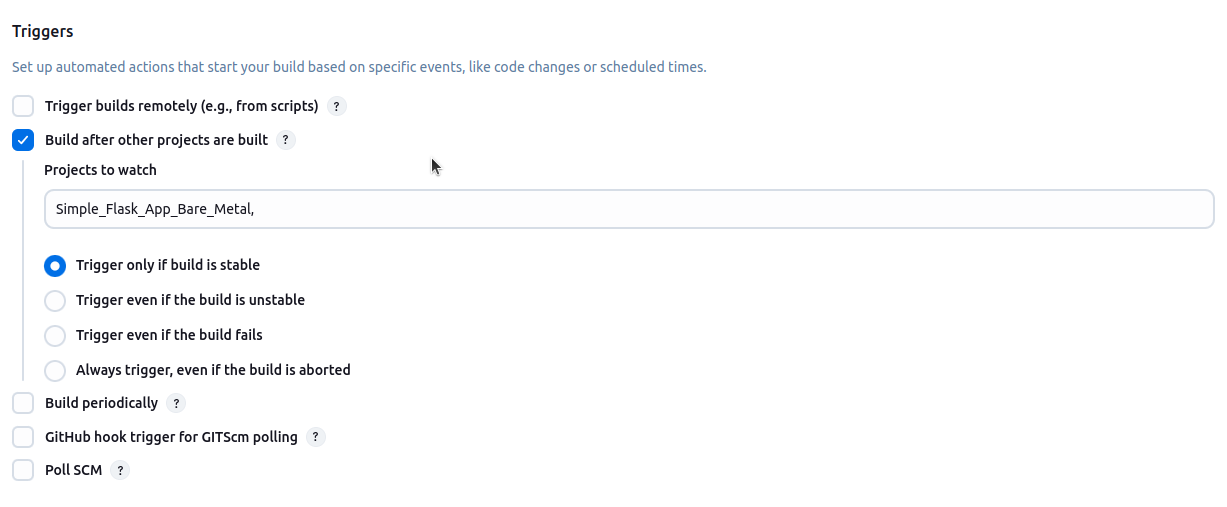
# **app directly on the server.**

# **Simple\_Flask\_App\_Docker**

**Build Triggers**

This job runs automatically after the Simple\_Flask\_App\_Bare\_Metal job finishes

successfully



**Build Steps:**

**Docker Build and Publish**

Builds a Docker image using the Dockerfile for gitrepo.

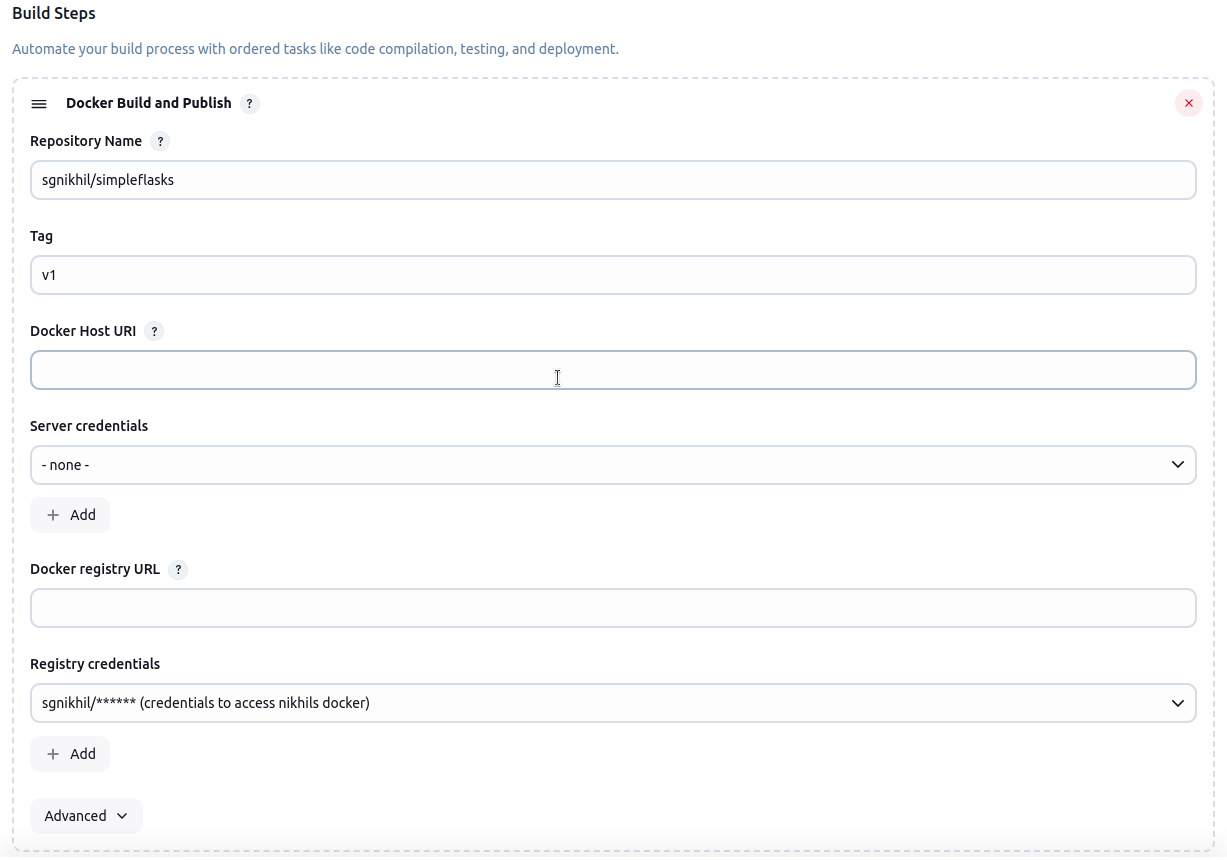
Repository name: sgnikhil/simpleflasks

Tag: v1

**Uses Docker credentials to push the image to Docker Hub.**

**Execute Shell**

**Runs a shell command to start a container from the new image**





**This job starts your Flask app in a Docker container on port 5070, right after the Bare Metal job**

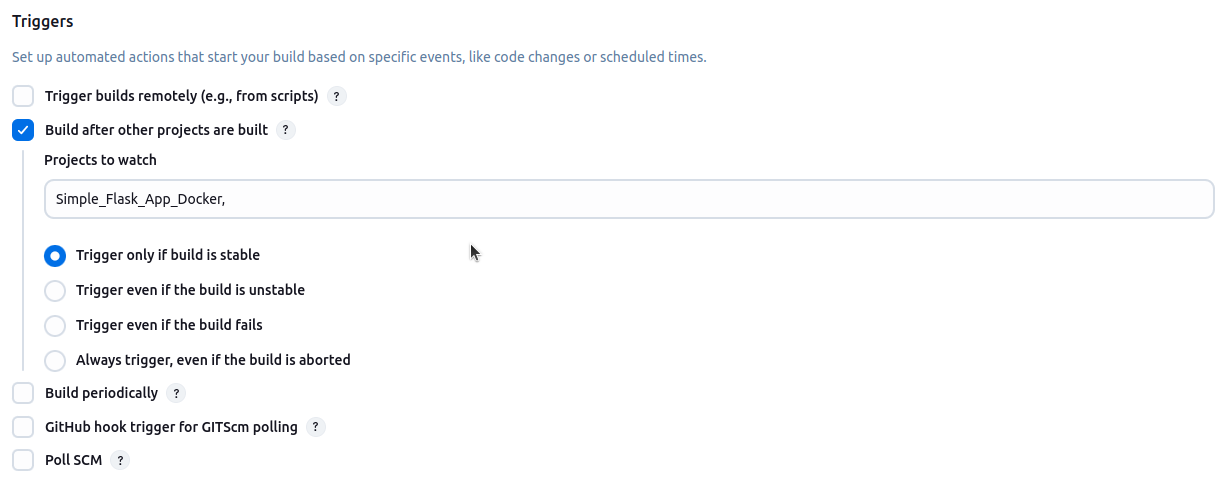
**finishes successfully.**

# **Simple\_Flask\_App\_Kube**

**Build Triggers**

This job runs automatically after the Simple\_Flask\_App\_Docker job finishes

successfully



**Build Steps:**

**Deploy to Kubernetes Build Step**

**Kubeconfig:**

Select the Kubernetes cluster credentials (here, "**47\_49 (Master 47 Slave 49)**") so Jenkins can connect to your Kubernetes cluster.

**Config Files:**

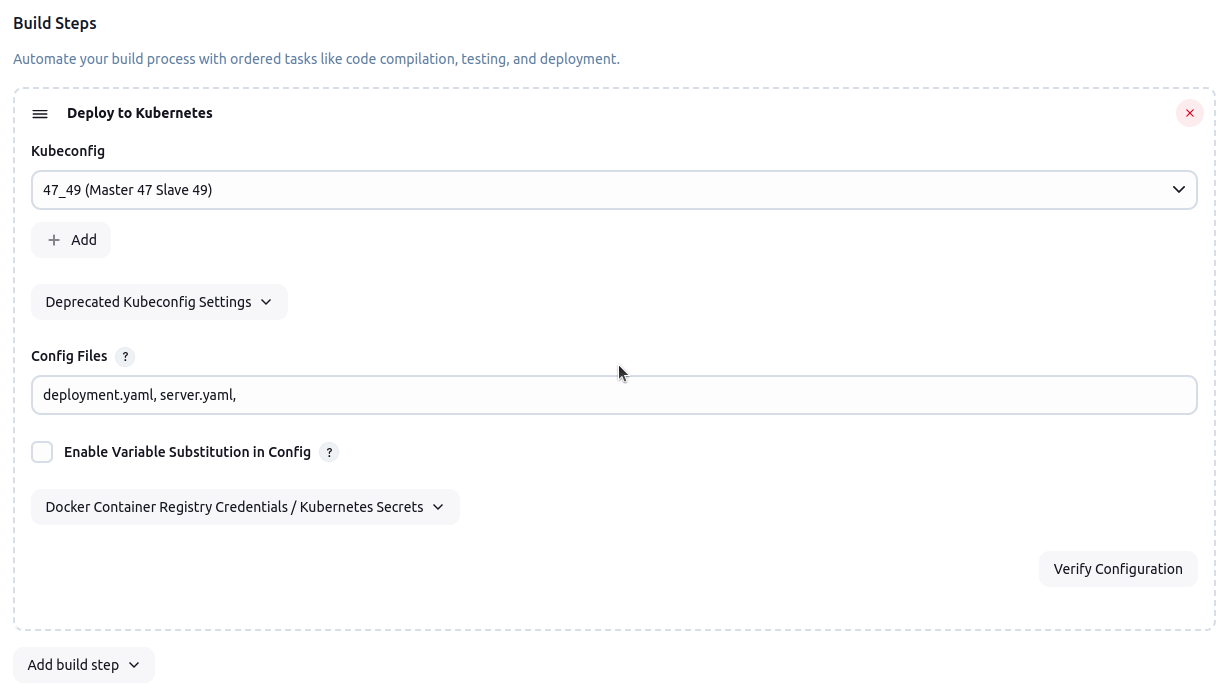
Lists the Kubernetes YAML files to apply:

deployment.yaml, service.yaml

These files define how your app will run and be exposed in the cluster.

**Docker Container Registry Credentials / Kubernetes Secrets:**

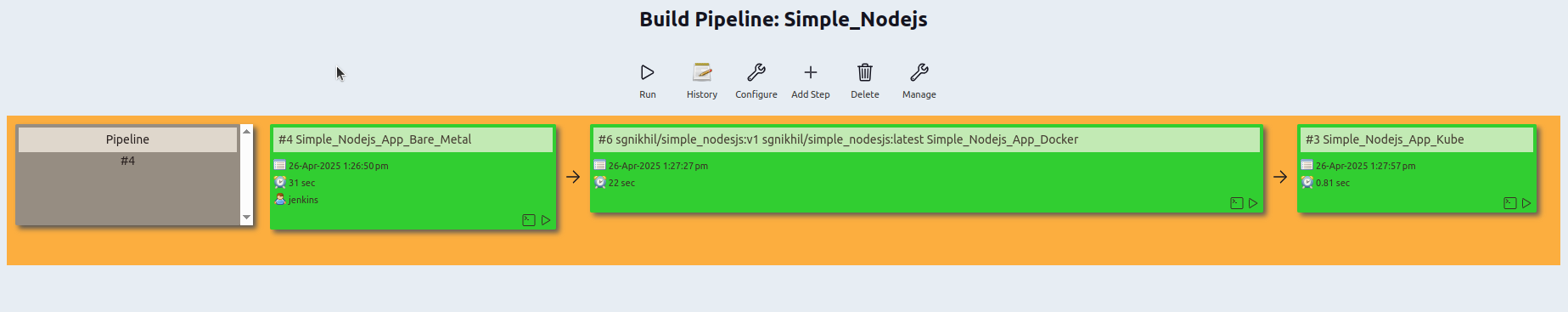
Let Jenkins use stored credentials if your deployment needs to pull images from a private Docker registry.



# **Simple\_Nodejs CI-CD Pipeline**

**Prerequisites**

* Jenkins server is up and running
* Required plugins installed: Git, Node.js, npm, Docker, Kubernetes
* Docker and Kubernetes cluster access



**Pipeline Steps Overview**

* Bare MetalDeploy
* Docker Build and Push, Run
* Kubernetes Deploy

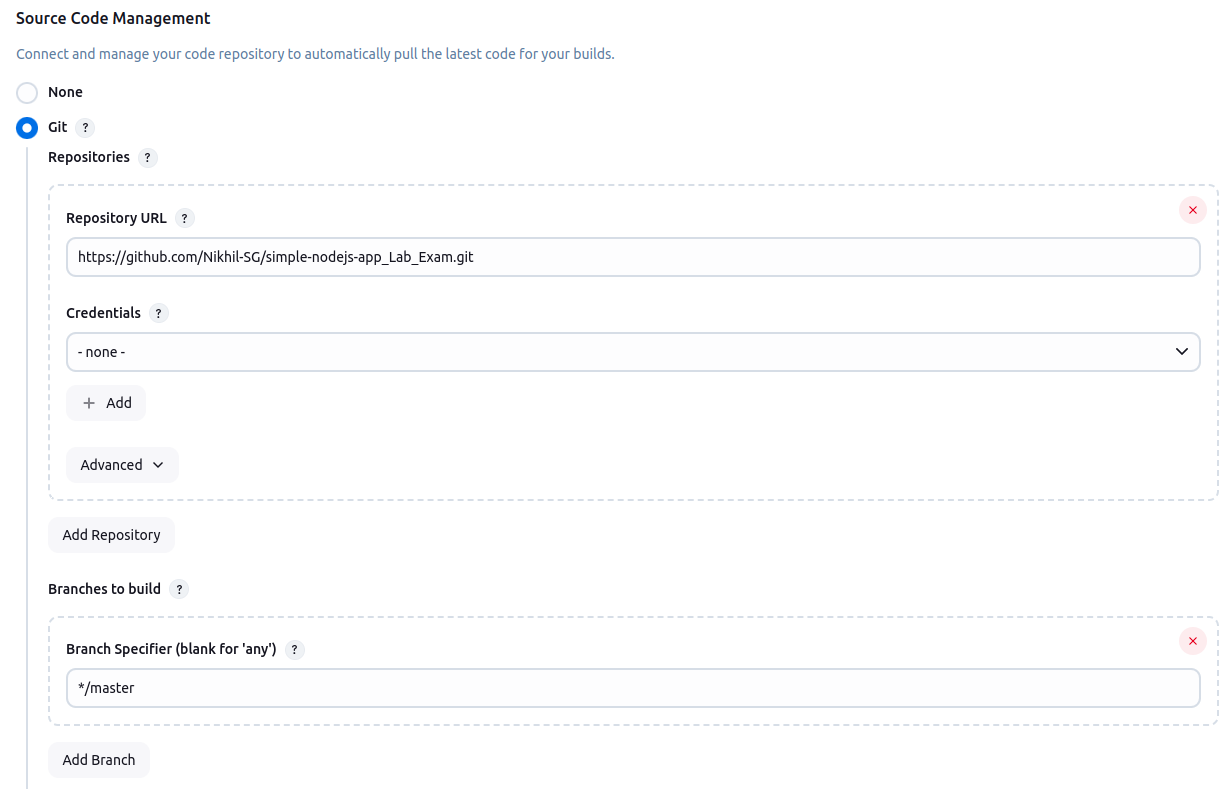
# **Simple\_Nodejs\_App\_Bare\_Metal**

**Source Code Management**

Select Git.

Enter the repo URL: https://github.com/Nikhil-SG/Simple-Flask-App-Docker.git

Set branch to build: \*/master.



**Build Steps:**

**Execute Shell**

Updates the server’s package list.

Removes any old Node.js and npm versions.

Installs Node.js 18.x (latest stable version).

Installs all Node.js dependencies with npm install.

Installs pm2 globally (a Node.js process manager).

Restarts the app using pm2 with index.js as the entry point



# **Jenkins downloads your Node.js app, sets up the latest Node.js, installs dependencies, and runs the app using pm2 for better process management.**

# **Simple\_Nodejs\_App\_Docker**

**Build Triggers**

This job runs automatically after the Simple\_Nodejs\_App\_Bare\_Metal job finishes

successfully



**Build Steps:**

**Docker Build and Publish**

Builds a Docker image using the Dockerfile for gitrepo.

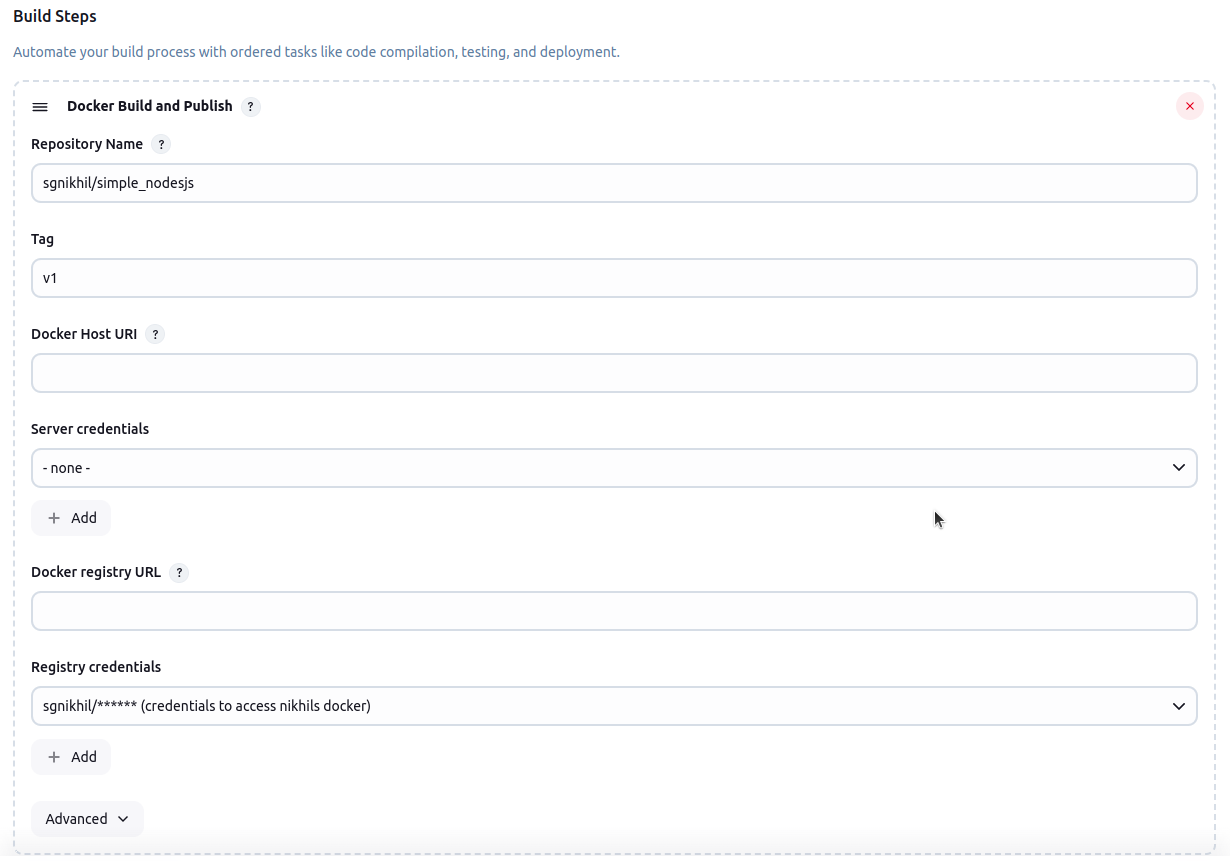
Repository name: sgnikhil/simple\_nodejs

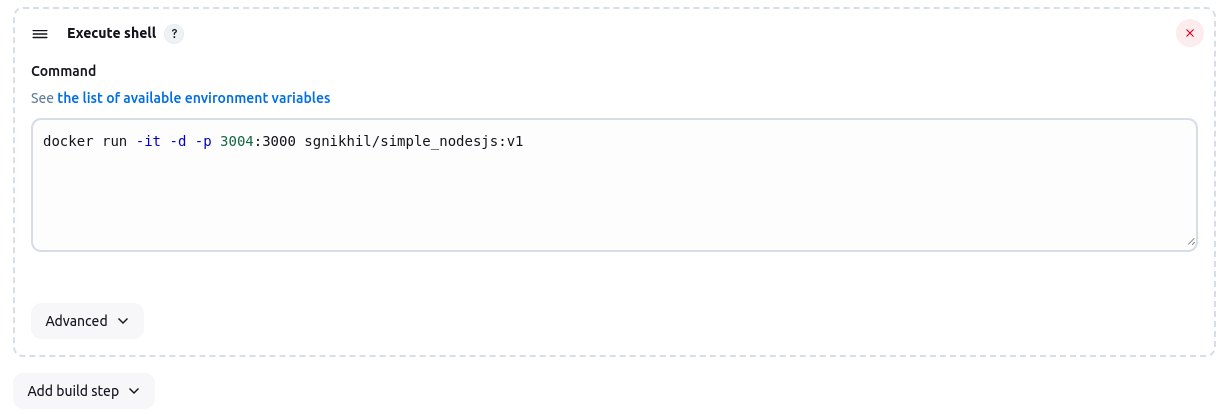
Tag: v1

**Uses Docker credentials to push the image to Docker Hub.**

**Execute Shell**

**Runs a shell command to start a container from the new image**





**This job starts your Node.js application in a Docker container on port 5070, right after the Bare Metal job finishes successfully.**

# **Simple\_Nodejs\_App\_Kube**

**Build Steps:**

**Deploy to Kubernetes Build Step**

**Kubeconfig:**

Select the Kubernetes cluster credentials (here, "**47\_49 (Master 47 Slave 49)**") so Jenkins can connect to your Kubernetes cluster.

**Config Files:**

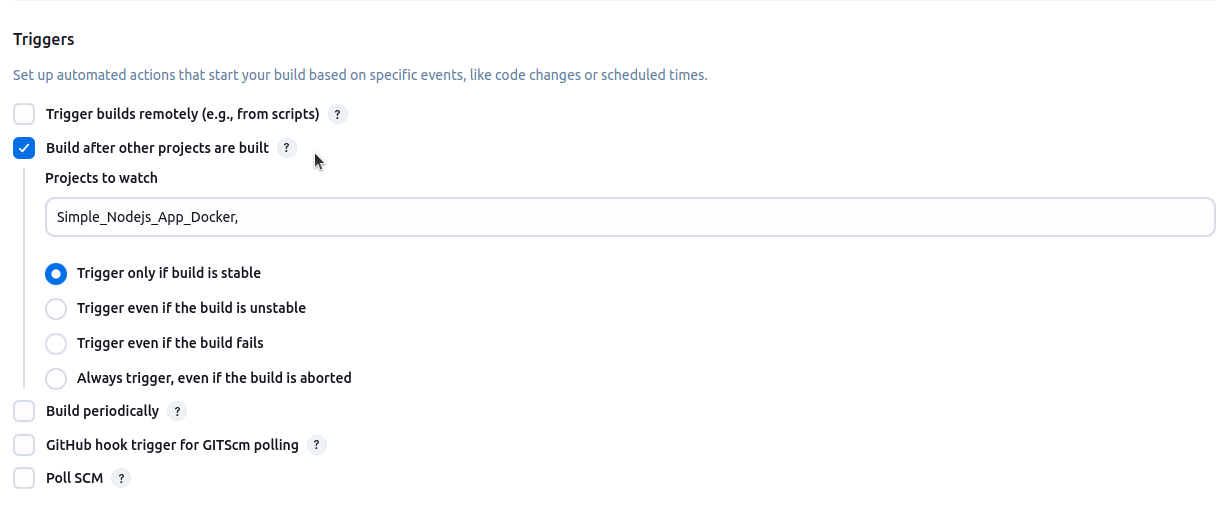
Lists the Kubernetes YAML files to apply:

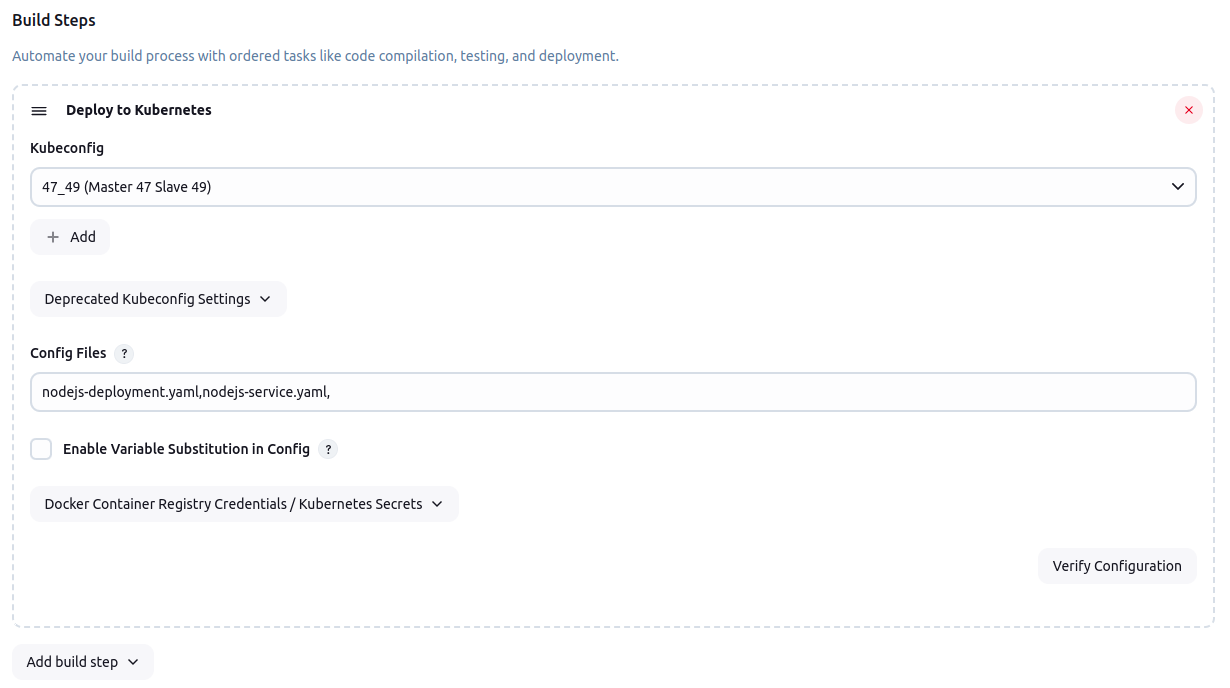
deployment.yaml, service.yaml

These files define how your app will run and be exposed in the cluster.

**Docker Container Registry Credentials / Kubernetes Secrets:**

Let Jenkins use stored credentials if your deployment needs to pull images from a private Docker registry.



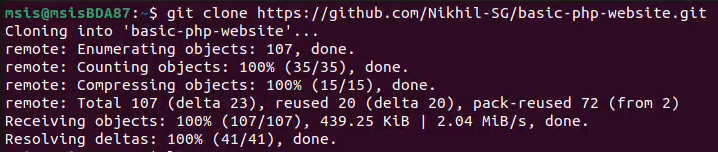


**Bookalbum\_php\_CICD\_Manual\_Pipeline**

**Deploying on a Local Machine (Bare Metal)**

**Clone the Repository:**

Download the PHP website project from GitHub to your local machine.

**View Files:**

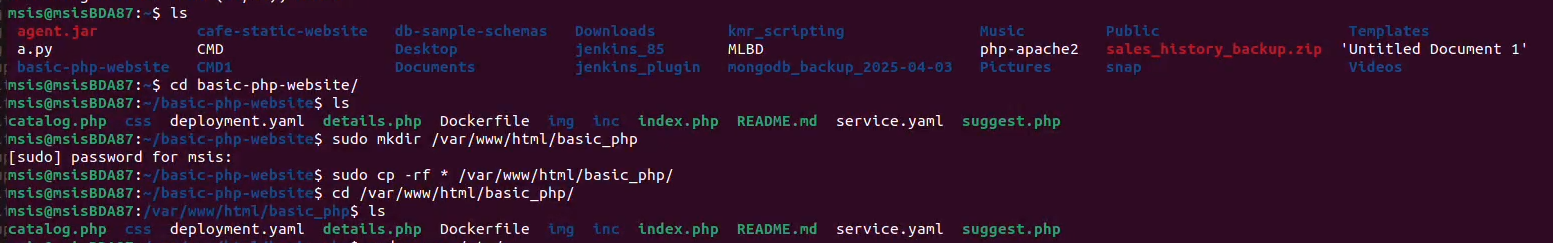
Use ls to list files and folders.

**Prepare Web Directory:**

Create a new directory under /var/www/html/ for your project using sudo mkdir.

**Copy Project Files:**

Copy all files from basic-php-website to your new directory in /var/www/html/ using sudo cp.



**Configure Apache:**

Open the Apache config file (/etc/apache2/sites-available/000-default.conf) with sudo nano and update the DocumentRoot to point to your new project folder.





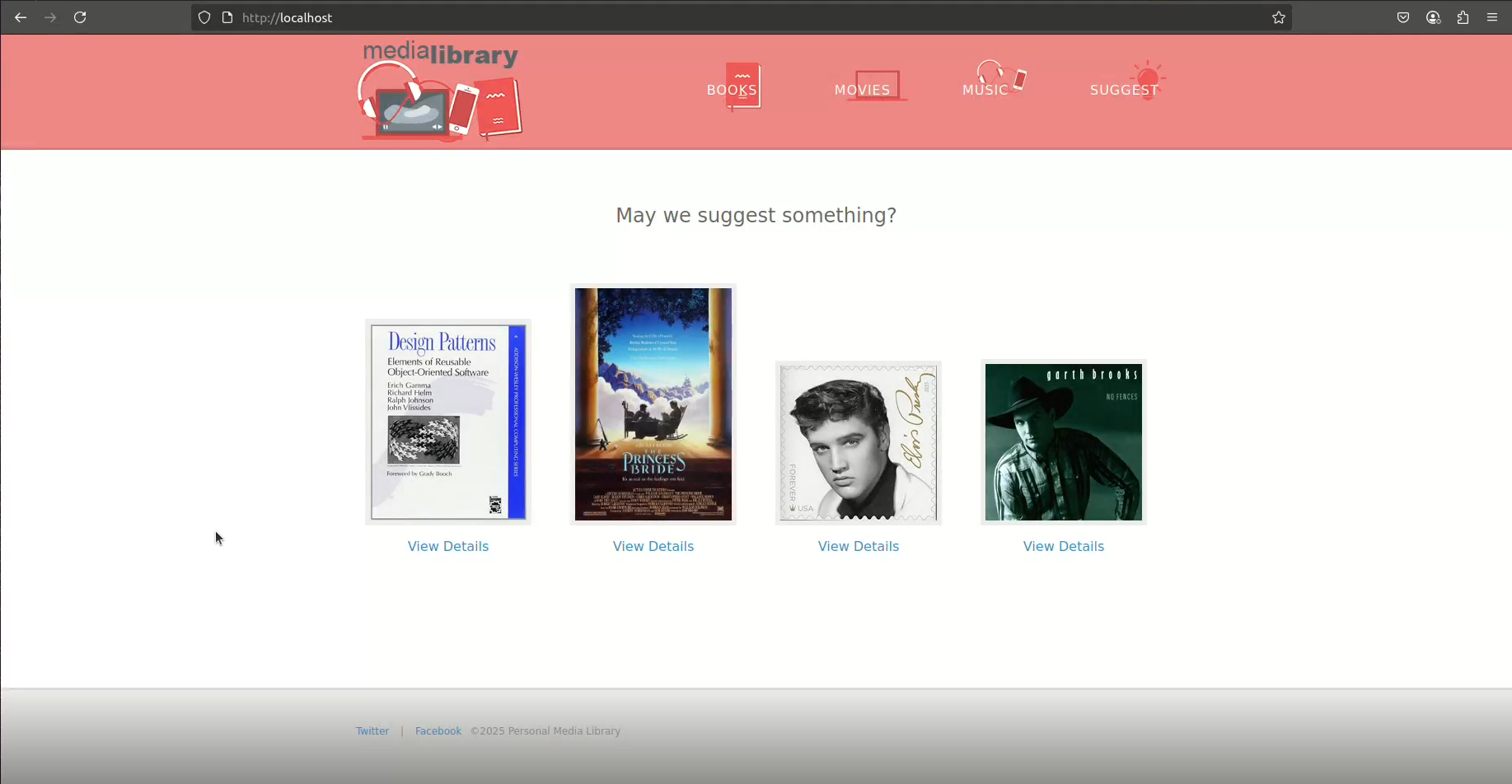
**Enable Site & Restart Apache:**

Enable the site with a2ensite and restart Apache with sudo systemctl restart



**Access the Website:**

Open a browser and go to localhost:80 or your system’s IP address to view the website.



**Deploying with Docker**

**Log in to Docker:**

Authenticate to your Docker account. cd - change directory to basic-php-website folder

**Build the Docker Image:**

Navigate to the project folder, ensure a Dockerfile exists, and build the image using:

sudo docker build -t your\_docker\_username/item\_name:version .



**Run the Docker Container:**

Start the container with:

sudo docker run -it -d -p host\_port:container\_port your\_docker\_username/item\_name:version

**Push Image to Docker Hub:**

Upload your image with:

sudo docker push your\_docker\_username/item\_name:version



**Access the Website:**

In your browser, go to localhost:host\_port or your\_ip:host\_port.

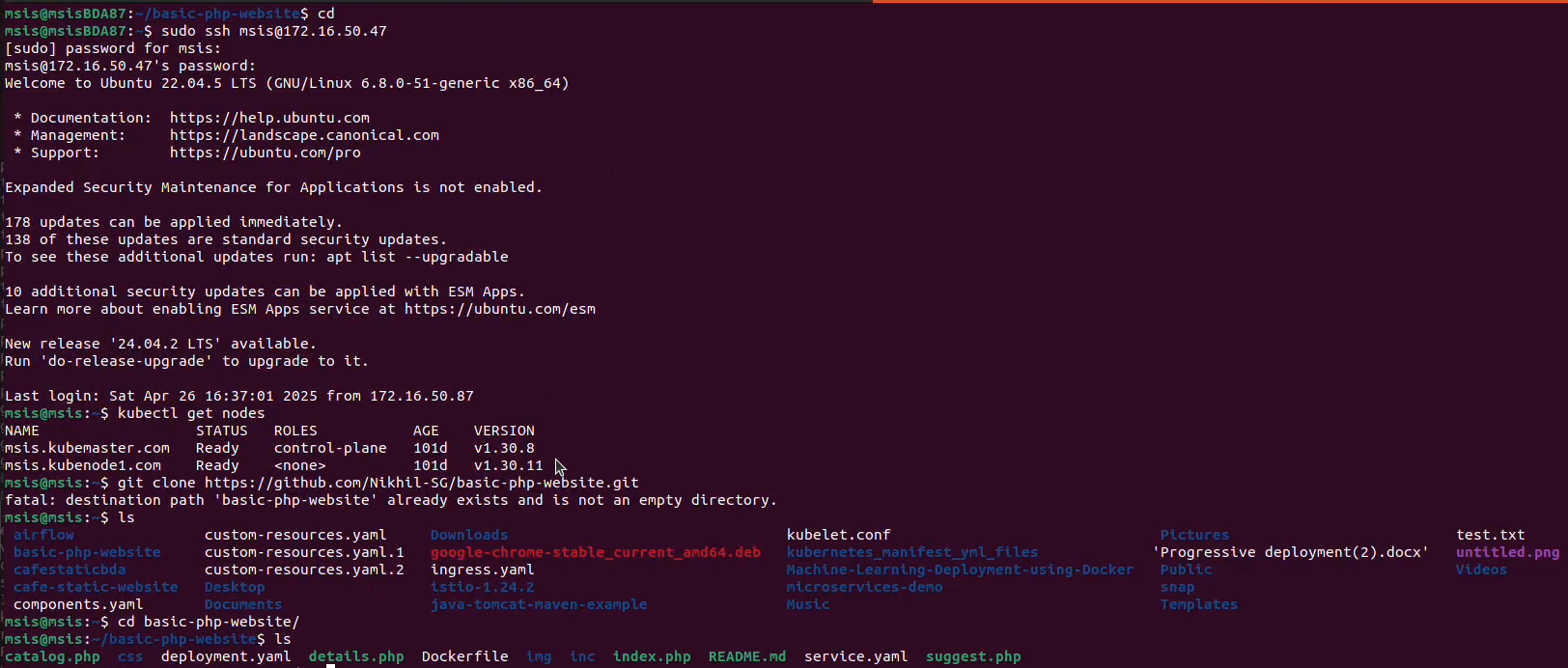
**Deploying on Kubernetes**

**Access Kubernetes Master:**

Log in and ensure all nodes are ready with kubectl get nodes.

**Clone the Project:**

Download your repository and check for necessary files (deployment.yaml, service.yaml).

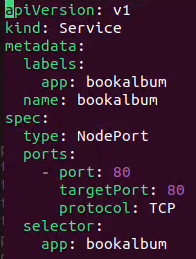
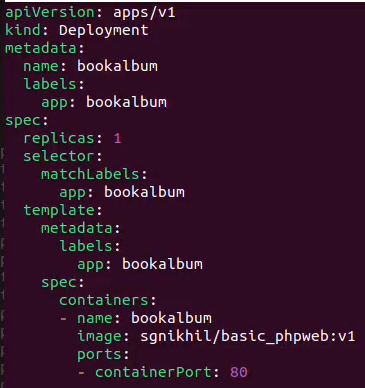


**Update Kubernetes Manifests:**

Edit deployment.yaml and service.yaml to confirm the correct Docker image and container port.







**Deploy to Cluster:**

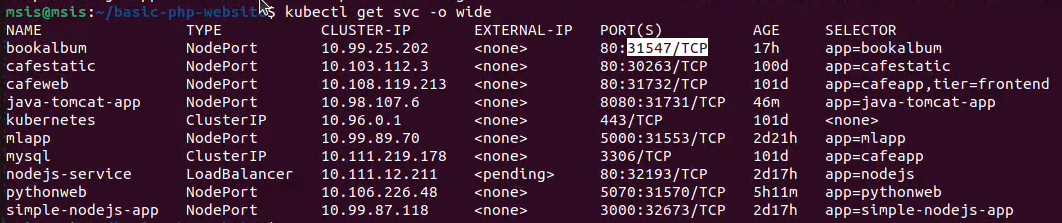
Apply the manifests using:

kubectl apply -f deployment.yaml kubectl apply -f service.yaml

**Verify Deployment:**

Check pods: kubectl get pods -o wide, Check service and port: kubectl get svc -o wide





**Access the Website:**

Open a browser and go to node\_ip:service\_port to view your PHP website running on the cluster.

