Installing Docker

Step 1: Update System Packages

Run the following command to update your system's package list:

```
sudo apt update
```

Step 2: Install Docker

Install Docker using the following command:

```
sudo apt install -y docker.io
```

Step 3: Enable and Start Docker Service

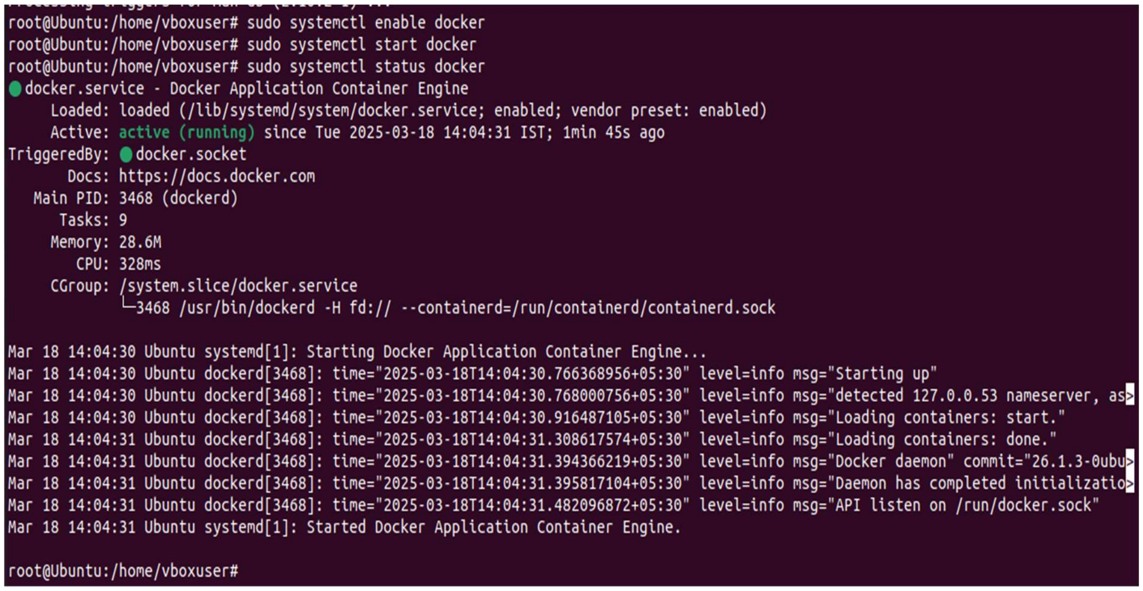
Enable Docker to start at boot and then start the Docker service:

```
sudo systemctl enable docker
sudo systemctl start docker
```

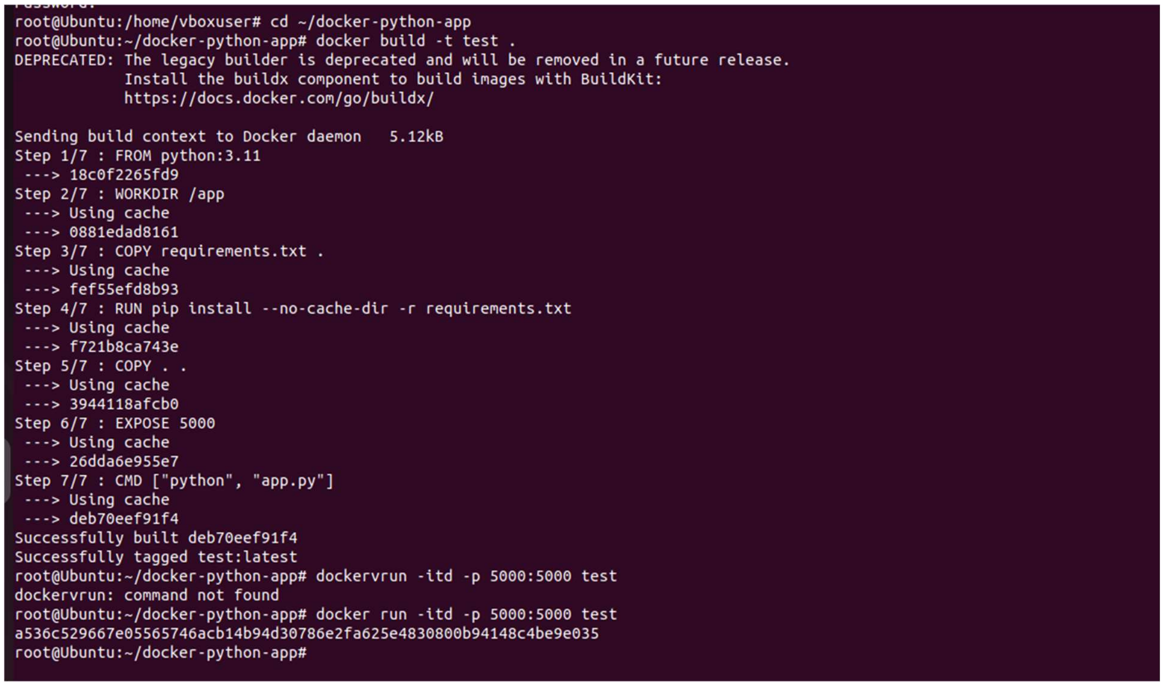
Step 4: Verify Installation

To ensure that Docker is installed successfully, check its version:

```
docker –version
```



```
root@Ubuntu:/home/vboxuser# sudo systemctl enable docker
root@Ubuntu:/home/vboxuser# sudo systemctl start docker
root@Ubuntu:/home/vboxuser# sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2025-03-18 14:04:31 IST; 1min 45s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 3468 (dockerd)
      Tasks: 9
     Memory: 28.6M
        CPU: 328ms
     CGroup: /system.slice/docker.service
             └─3468 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Mar 18 14:04:30 Ubuntu systemd[1]: Starting Docker Application Container Engine...
Mar 18 14:04:30 Ubuntu dockerd[3468]: time="2025-03-18T14:04:30.766368956+05:30" level=info msg="Starting up"
Mar 18 14:04:30 Ubuntu dockerd[3468]: time="2025-03-18T14:04:30.768000756+05:30" level=info msg="detected 127.0.0.53 nameserver, as>
Mar 18 14:04:30 Ubuntu dockerd[3468]: time="2025-03-18T14:04:30.916487105+05:30" level=info msg="Loading containers: start."
Mar 18 14:04:31 Ubuntu dockerd[3468]: time="2025-03-18T14:04:31.308617574+05:30" level=info msg="Loading containers: done."
Mar 18 14:04:31 Ubuntu dockerd[3468]: time="2025-03-18T14:04:31.394366219+05:30" level=info msg="Docker daemon" commit="26.1.3-0ubu>
Mar 18 14:04:31 Ubuntu dockerd[3468]: time="2025-03-18T14:04:31.395817104+05:30" level=info msg="Daemon has completed initializatio>
Mar 18 14:04:31 Ubuntu dockerd[3468]: time="2025-03-18T14:04:31.482096872+05:30" level=info msg="API listen on /run/docker.sock"
Mar 18 14:04:31 Ubuntu systemd[1]: Started Docker Application Container Engine.

root@Ubuntu:/home/vboxuser#
```



```
root@Ubuntu:/home/vboxuser# cd ~/docker-python-app
root@Ubuntu:~/docker-python-app# docker build -t test .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon   5.12kB
Step 1/7 : FROM python:3.11
 ---> 18c0f2265fd9
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> 0881edad8161
Step 3/7 : COPY requirements.txt .
 ---> Using cache
 ---> fef55efd8b93
Step 4/7 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Using cache
 ---> f721b8ca743e
Step 5/7 : COPY . .
 ---> Using cache
 ---> 3944118afcb0
Step 6/7 : EXPOSE 5000
 ---> Using cache
 ---> 26dda6e955e7
Step 7/7 : CMD ["python", "app.py"]
 ---> Using cache
 ---> deb70eef91f4
Successfully built deb70eef91f4
Successfully tagged test:latest
root@Ubuntu:~/docker-python-app# dockervrun -itd -p 5000:5000 test
dockervrun: command not found
root@Ubuntu:~/docker-python-app# docker run -itd -p 5000:5000 test
a536c529667e05565746acb14b94d30786e2fa625e4830800b94148c4be9e035
root@Ubuntu:~/docker-python-app#
```

Installing Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. Follow these steps to install it:

Step 1: Install Curl

Ensure that `curl` is installed by running:

```
sudo apt install curl
```

Step 2: Download Docker Compose

Download the latest version of Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Step 3: Give Execution Permission

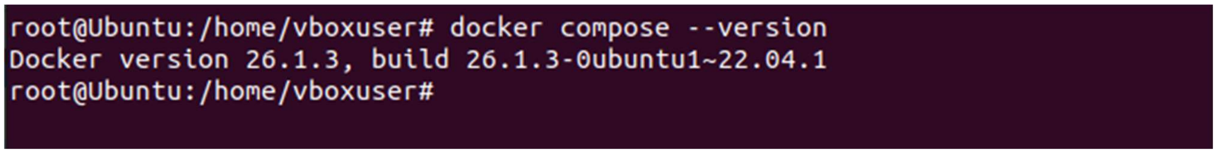
Make the downloaded file executable:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Step 4: Verify Installation

Check if Docker Compose is installed correctly:

```
docker-compose –version
```



```
root@Ubuntu:/home/vboxuser# docker compose --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~22.04.1
root@Ubuntu:/home/vboxuser#
```

Creating a Python "Hello World" Application

To demonstrate Docker, we will create a simple Python application using Flask.

Step 1: Create a Project Directory

```
mkdir ~/docker-python-app
cd ~/docker-python-app
```

Step 2: Create a Python Script

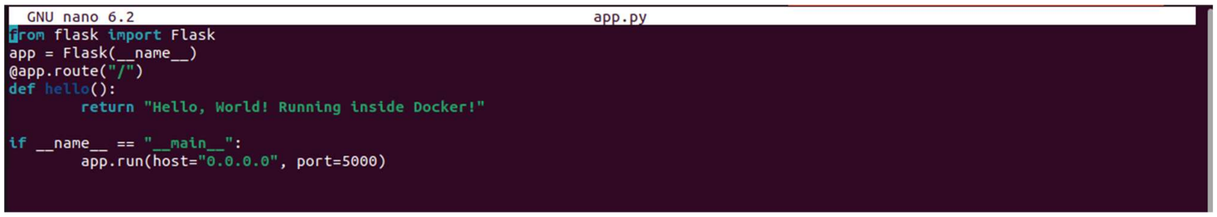
Create a file named `app.py`:

```
nano app.py
```

Step 3: Write Python Code

Add the following code inside `app.py` and save the file:

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello, World! Running inside Docker!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```



```
  GNU nano 6.2                                    app.py
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
        return "Hello, World! Running inside Docker!"

if __name__ == "__main__":
        app.run(host="0.0.0.0", port=5000)
```

Installing Dependencies

To ensure that the necessary dependencies are available inside the container, create a `requirements.txt` file.

Step 1: Create a Dependencies File

```
nano requirements.txt
```

Step 2: Add Required Package

Inside the file, add the following line and save it:

```
flask
```

Creating a Dockerfile

A Dockerfile contains instructions to build a Docker image.

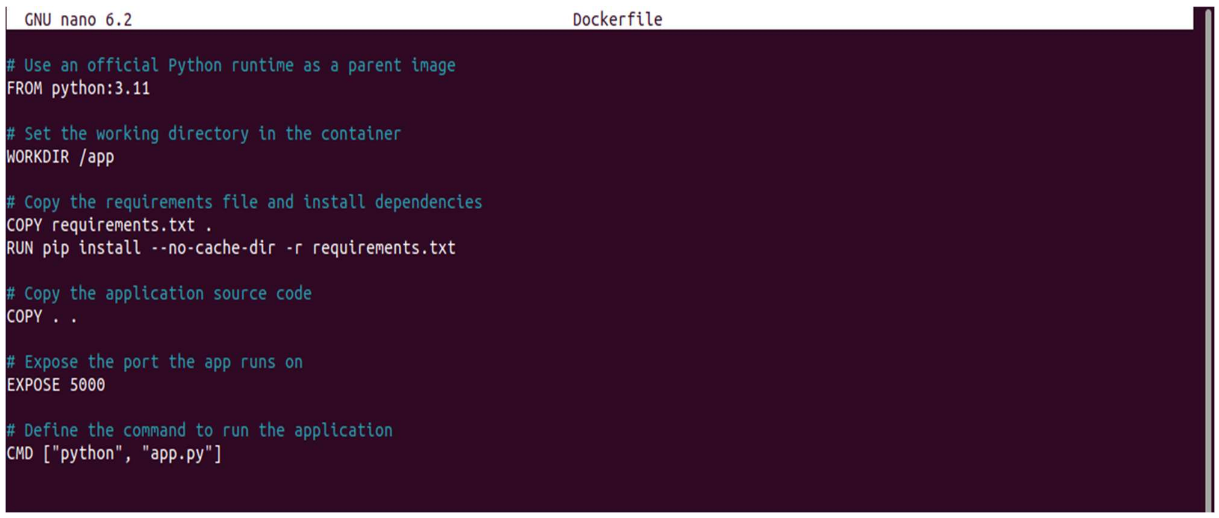
Step 1: Create a Dockerfile

```
nano Dockerfile
```

Step 2: Add Docker Instructions

Paste the following content into the file:

```
# Use an official Python runtime as a parent image
FROM python:3.11

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application source code
COPY . .

# Expose the port the app runs on
EXPOSE 5000

# Define the command to run the application
CMD ["python", "app.py"]
```



```
GNU nano 6.2                                   Dockerfile

# Use an official Python runtime as a parent image
FROM python:3.11

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy the application source code
COPY . .

# Expose the port the app runs on
EXPOSE 5000

# Define the command to run the application
CMD ["python", "app.py"]
```

Creating a Docker Compose File

Docker Compose allows you to define and run multiple containers as a single service.

Step 1: Create a Docker Compose File

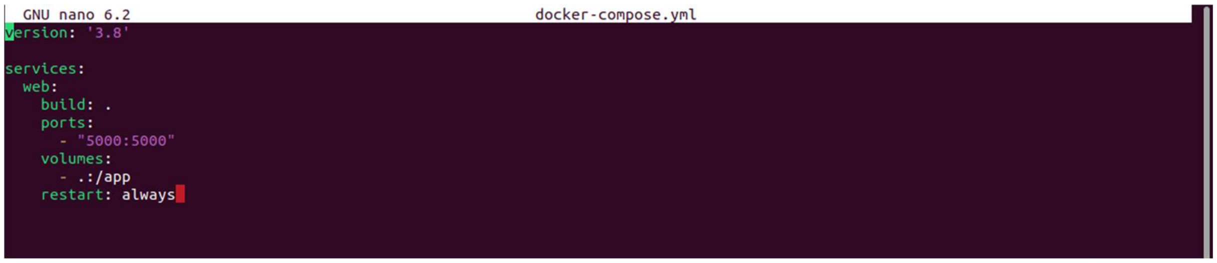
```
nano docker-compose.yml
```

Step 2: Add Configuration

Paste the following content into the file:

```
version: '3.8'
```

```
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    restart: always
```



```
  GNU nano 6.2                                         docker-compose.yml
version: '3.8'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/app
    restart: always
```

Building and Running the Docker Container

Now, we will build and run the application inside a Docker container.

Step 1: Build the Docker Image

```
sudo docker-compose build
```

Step 2: Start the Container

```
sudo docker-compose up -d
```

Verifying the Setup

Step 1: Check Docker Images

To list the available Docker images, run:

```
sudo docker images
```

Step 2: Build and Run Manually (Alternative Method)

```
docker build -t test .
docker run -itd -p 5000:5000 test
```

Step 3: Check Logs

To check if the container is running properly, use:

```
docker logs <container_id>
```

Step 4: Access the Application

Open a web browser and go to:

```
http://localhost:5000
```

You should see the output:

```
Hello, World! Running inside Docker!
```

Pushing the Project to GitHub

Step 1: Clone the Repository

```
git clone https://github.com/SujithaKC/jenkins-docker-demo.git
cd jenkins-docker-demo
```

Step 2: Move Files into Repository

```
mv ~/docker-python-app/Dockerfile ~/docker-python-app/requirements.txt ~/docker-
python-app/app.py ~/docker-python-app/docker-compose.yml .
```

Step 3: Add and Commit the Changes

```
git add --all
git commit -m "Initial commit for docker app"
```

Step 4: Push to GitHub

```
git push origin main
```

Configuring Jenkins Pipeline

Step 1: Create a `Jenkinsfile`

```
nano Jenkinsfile
```

Step 2: Add Jenkins Pipeline Code

Paste the following content into the file:

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = "sathiya9944/docker-app:latest"  // Change this to your
registry
        CONTAINER_NAME = "docker-running-app"
        REGISTRY_CREDENTIALS = "docker-hub-credentials"  // Jenkins credentials
ID
    }

    stages {
        stage('Checkout Code') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'sathiya',
usernameVariable: 'GIT_USER', passwordVariable: 'GIT_TOKEN')]) {
                    git url:
"https://$GIT_USER:$GIT_TOKEN@github.com/sathiya9944/jenkins-docker.git",
branch: 'main'
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                sh 'docker build -t $DOCKER_IMAGE .'
            }
        }

        stage('Login to Docker Registry') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'sathiya9944',
usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
                    sh 'echo $DOCKER_PASS | docker login -u $DOCKER_USER --
password-stdin'
                }
            }
        }

        stage('Push to Container Registry') {
            steps {
                sh 'docker push $DOCKER_IMAGE'
            }
        }

        stage('Stop & Remove Existing Container') {
            steps {
                script {
                    sh '''
                    if [ "$(docker ps -aq -f name=$CONTAINER_NAME)" ]; then
                        docker stop $CONTAINER_NAME || true
                        docker rm $CONTAINER_NAME || true
                    fi
                    '''
                }
            }
        }

        stage('Run Docker Container') {
            steps {
                sh 'docker run -d -p 5001:5000 --name $CONTAINER_NAME
$DOCKER_IMAGE'
            }
        }
    }

    post {
        success {
            echo "Build, push, and container execution successful!"
        }
        failure {
            echo "Build or container execution failed."
        }
    }
}
```

Running Jenkins Build

Step 1: Resolve Security Error

```
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

Step 2: Verify Jenkins Credentials

Ensure that the correct credentials are set in Jenkins before triggering the build.

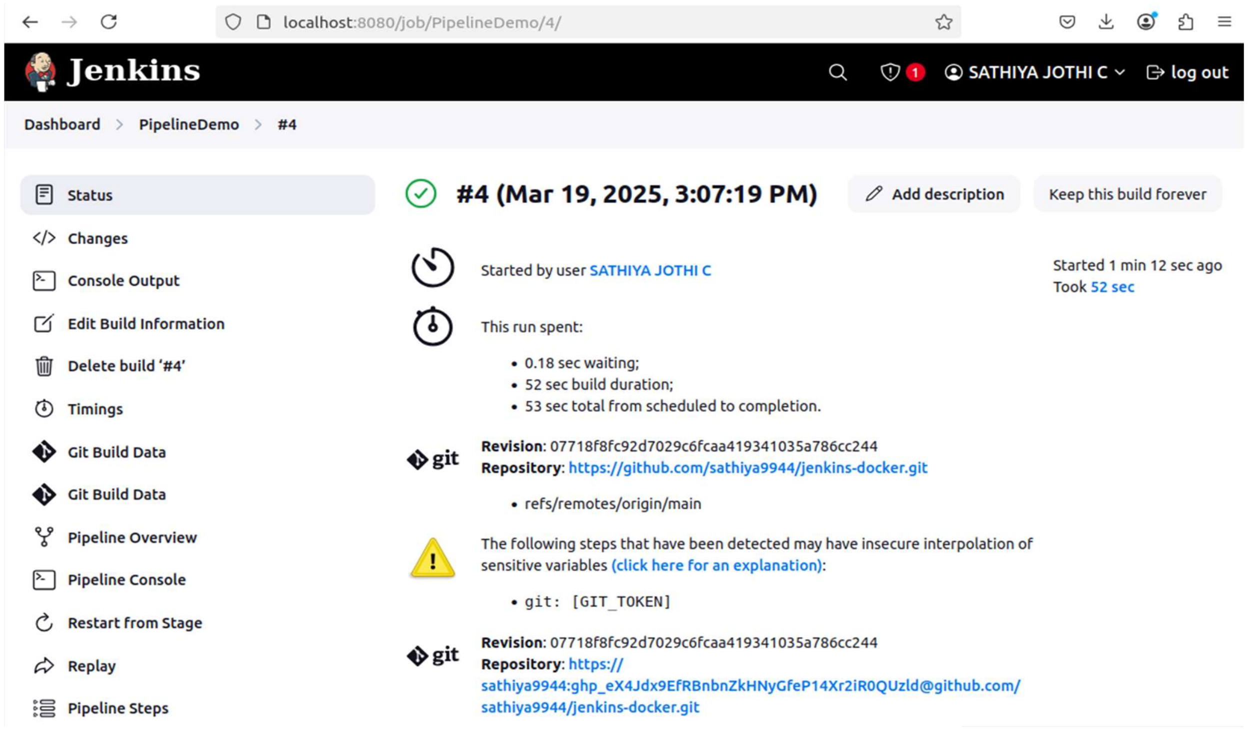
Step 3: Run the Build

Trigger the Jenkins build. If successful, the Docker image will be updated and the application will be running on port `5001`.

Step 4: Fix Naming Issues

If Jenkins cannot find the `Jenkinsfile`, rename it using:

```
mv jenkinsfile Jenkinsfile
git add .
git commit -m "Fixed Jenkinsfile naming issue"
git push origin main
```



#4 (Mar 19, 2025, 3:07:19 PM)

Started by user SATHIYA JOTHI C

Started 1 min 12 sec ago
Took 52 sec

This run spent:

- 0.18 sec waiting;
- 52 sec build duration;
- 53 sec total from scheduled to completion.

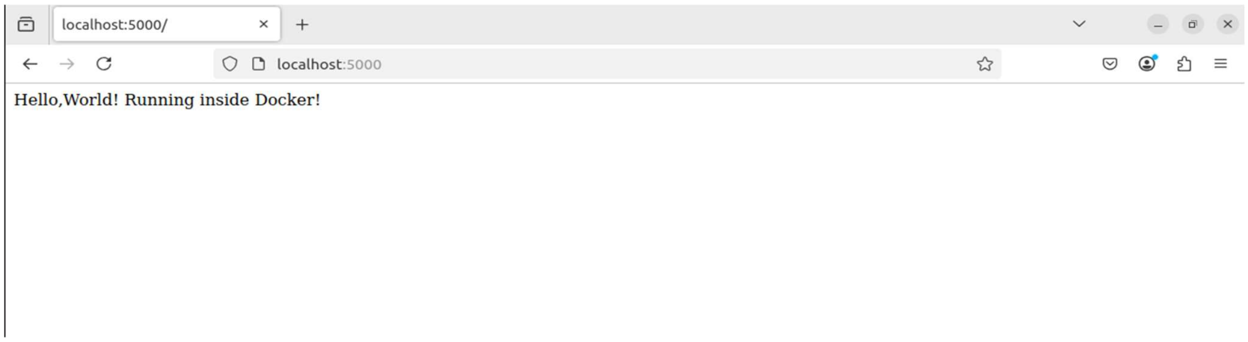
Revision: 07718f8fc92d7029c6fcaa419341035a786cc244
Repository: https://github.com/sathiya9944/jenkins-docker.git

- refs/remotes/origin/main

The following steps that have been detected may have insecure interpolation of sensitive variables (click here for an explanation):

- `git: [GIT_TOKEN]`

Revision: 07718f8fc92d7029c6fcaa419341035a786cc244
Repository: https://// sathiya9944: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@github.com/sathiya9944/jenkins-docker.git



Hello,World! Running inside Docker!