DAY - 2

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

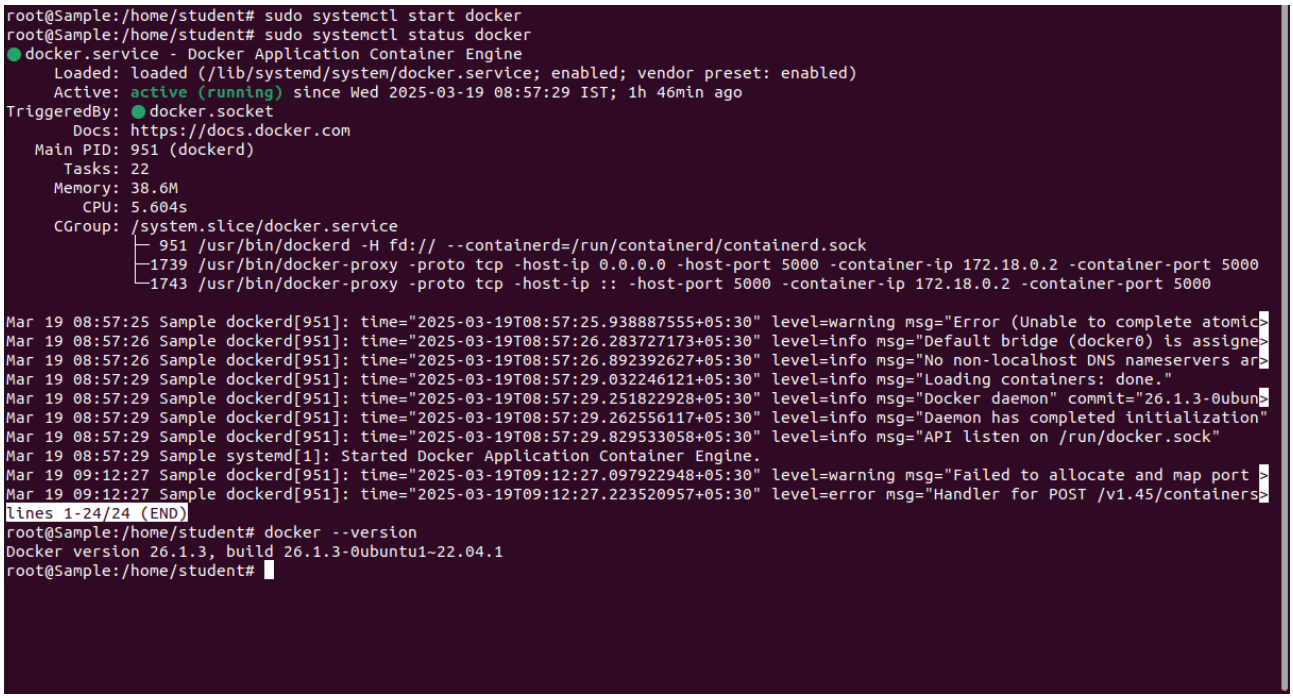
Enable Docker to start at boot and then start the Docker service:

```
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@Sample:/home/student# sudo systemctl start docker
root@Sample:/home/student# 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 Wed 2025-03-19 08:57:29 IST; 1h 46min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 951 (dockerd)
      Tasks: 22
     Memory: 38.6M
        CPU: 5.604s
     CGroup: /system.slice/docker.service
             ├─ 951 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
             ├─1739 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5000 -container-ip 172.18.0.2 -container-port 5000
             └─1743 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 5000 -container-ip 172.18.0.2 -container-port 5000

Mar 19 08:57:25 Sample dockerd[951]: time="2025-03-19T08:57:25.938887555+05:30" level=warning msg="Error (Unable to complete atomic>
Mar 19 08:57:26 Sample dockerd[951]: time="2025-03-19T08:57:26.283727173+05:30" level=info msg="Default bridge (docker0) is assigne>
Mar 19 08:57:26 Sample dockerd[951]: time="2025-03-19T08:57:26.892392627+05:30" level=info msg="No non-localhost DNS nameservers ar>
Mar 19 08:57:29 Sample dockerd[951]: time="2025-03-19T08:57:29.032246121+05:30" level=info msg="Loading containers: done."
Mar 19 08:57:29 Sample dockerd[951]: time="2025-03-19T08:57:29.251822928+05:30" level=info msg="Docker daemon" commit="26.1.3-0ubun>
Mar 19 08:57:29 Sample dockerd[951]: time="2025-03-19T08:57:29.262556117+05:30" level=info msg="Daemon has completed initialization"
Mar 19 08:57:29 Sample dockerd[951]: time="2025-03-19T08:57:29.829533058+05:30" level=info msg="API listen on /run/docker.sock"
Mar 19 08:57:29 Sample systemd[1]: Started Docker Application Container Engine.
Mar 19 09:12:27 Sample dockerd[951]: time="2025-03-19T09:12:27.097922948+05:30" level=warning msg="Failed to allocate and map port >
Mar 19 09:12:27 Sample dockerd[951]: time="2025-03-19T09:12:27.223520957+05:30" level=error msg="Handler for POST /v1.45/containers>
lines 1-24/24 (END)
root@Sample:/home/student# docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~22.04.1
root@Sample:/home/student#
```

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@Sample:/home/student# sudo chmod +x /usr/local/bin/docker-compose
root@Sample:/home/student# docker-compose --version
Docker Compose version v2.34.0
root@Sample:/home/student#
```

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`:

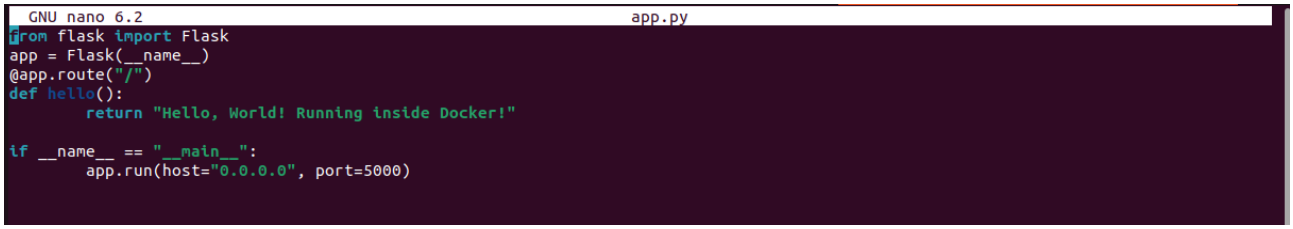
```
root@Sample:/home/student# cd ~/docker-python-app
root@Sample:~/docker-python-app# nano app.py
root@Sample:~/docker-python-app#
```

```
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)
```



Installing Dependencies

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

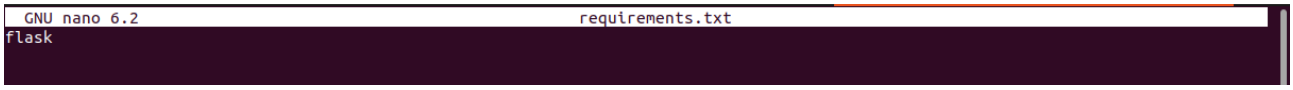
Step 1: Create a Dependencies File

```
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.

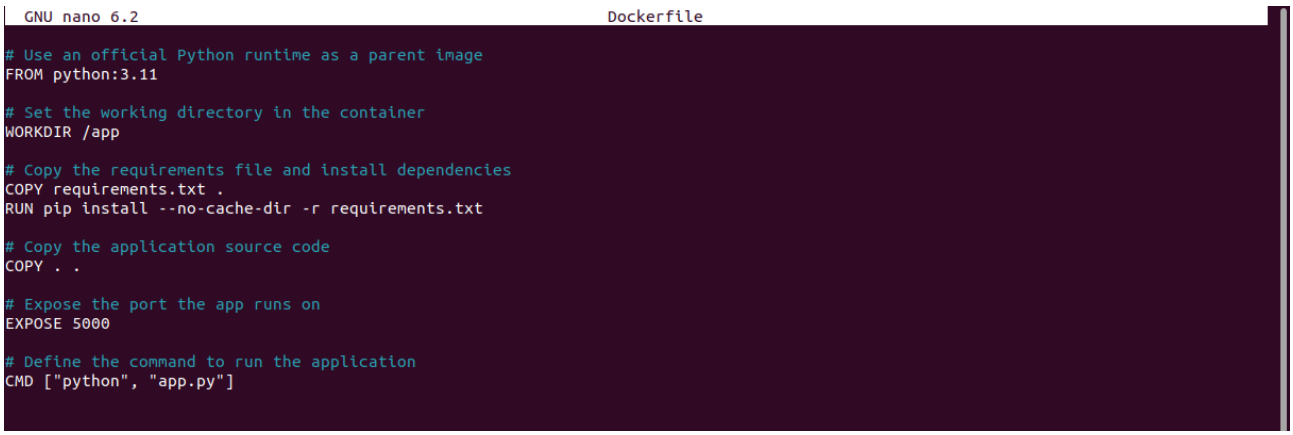
Step 1: Create a Dockerfile

```
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"]
```



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!
```

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
```

```
root@Sample:~# cd ~/docker-python-app
root@Sample:~/docker-python-app# git clone https://github.com/SujithaKC/jenkins-docker-demo.git
Cloning into 'jenkins-docker-demo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
root@Sample:~/docker-python-app# ls
app.py  docker-compose.yml  Dockerfile  jenkins-docker-demo  requirements.txt
root@Sample:~/docker-python-app# cd jenkins-docker-demo
root@Sample:~/docker-python-app/jenkins-docker-demo# cd ..
root@Sample:~/docker-python-app# mv Dockerfile requirements.txt app.py docker-compose.yml jenkins-docker-demo/
root@Sample:~/docker-python-app# ls
jenkins-docker-demo
root@Sample:~/docker-python-app# cd jenkins-docker-demo
root@Sample:~/docker-python-app/jenkins-docker-demo# ls
app.py  docker-compose.yml  Dockerfile  README.md  requirements.txt
root@Sample:~/docker-python-app/jenkins-docker-demo# git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Dockerfile
        app.py
        docker-compose.yml
        requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
root@Sample:~/docker-python-app/jenkins-docker-demo# git add --all
root@Sample:~/docker-python-app/jenkins-docker-demo# git commit -m "Initial commit for docker app"
[main d53da81] Initial commit for docker app
 Committer: root <root@Sample.myguest.virtualbox.org>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
```

```
root@Sample:~/docker-python-app/jenkins-docker-demo# git config --global user.name "SujithaKC"
root@Sample:~/docker-python-app/jenkins-docker-demo# git config --global user.email "sujithakanagarathinam@gmail.com"
root@Sample:~/docker-python-app/jenkins-docker-demo# git config --global --edit
root@Sample:~/docker-python-app/jenkins-docker-demo# git commit -m "Initial commit for docker app"
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
root@Sample:~/docker-python-app/jenkins-docker-demo# git push -u origin main
Username for 'https://github.com': SujithaKC
Password for 'https://SujithaKC@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls fo
r information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/SujithaKC/jenkins-docker-demo.git/'
root@Sample:~/docker-python-app/jenkins-docker-demo# git push https://SujithaKC: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@github.com/
SujithaKC/
jenkins-docker-demo.git
remote: Not Found
fatal: repository 'https://github.com/SujithaKC/' not found
jenkins-docker-demo.git: command not found
root@Sample:~/docker-python-app/jenkins-docker-demo# git push https://SujithaKC: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@github.com/
SujithaKC/jenkins-docker-demo.git
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 947 bytes | 947.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/SujithaKC/jenkins-docker-demo.git
   3b7d0c0..d53da81  main -> main
root@Sample:~/docker-python-app/jenkins-docker-demo#
```

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 = "DOCKER_USERNAME/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:
'github_credentials', usernameVariable: 'GIT_USER', passwordVariable:
'GIT_TOKEN')]) {
                    git url:
"https://$GIT_USER:$GIT_TOKEN@github.com/git_username/jenkins-docker-demo.git",
branch: 'main'
                }
            }
        }

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

        stage('Login to Docker Registry') {
            steps {
                withCredentials([usernamePassword(credentialsId:
'docker_credentials', 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."
        }
    }
}
```

This Jenkins Pipeline automates the process of deploying a Dockerized application. It does the following:

1. Checks out the code from a GitHub repository.
2. Builds a Docker image using the project's Dockerfile.
3. Logs in to Docker Hub with stored credentials.
4. Pushes the image to Docker Hub.
5. Stops and removes any existing container to prevent conflicts.
6. Runs a new container using the updated Docker image.
7. Displays success or failure messages after execution.

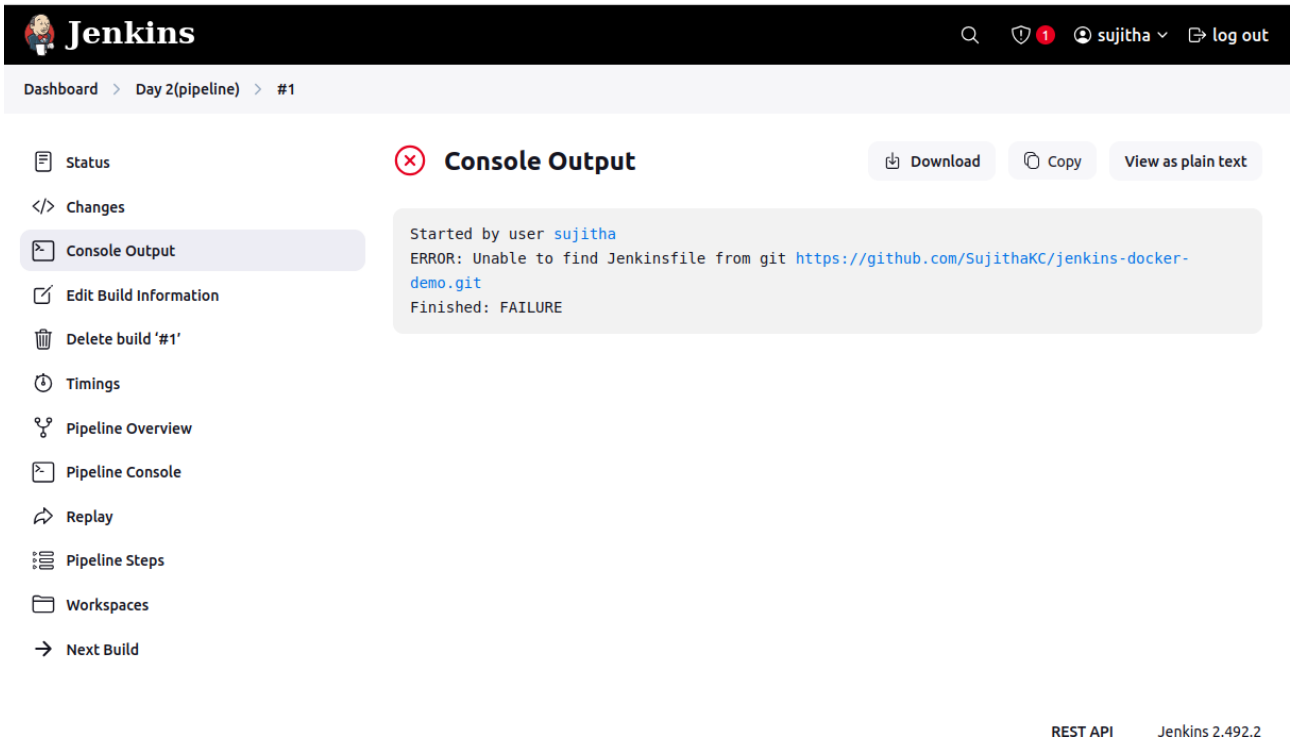
Running Jenkins Build

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: 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
```

Step 4: Run the Build

Hello, World! Running inside Docker!

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