

## Docker Weekend Task – 2

1. Write a brief explanation of what Docker volumes are and why they are used in containerized environments.

**State different types of volumes in Docker and also make a note on difference between them.**

Docker Volumes are storage for long-term data created and consumed by Docker containers. Volumes differ from storage in containers because they are controlled by Docker and persist even after the container is removed. Volumes are typically used to share data between containers, or to store database files, logs, uploads, etc. Volumes persist in a Docker-specific location and provide improved performance, portability, and flexibility compared to bind mounts, which are more difficult to use.

### Why Use Docker Volumes

#### 1. Data Persistence

Volumes ensure that data remains intact even if a container is stopped, deleted, or recreated.

For example, in a container running a database like MySQL or PostgreSQL, using a volume ensures that the actual data stored in the database is not lost when the container is updated or restarted.

#### 2. Data Sharing Between Containers

Volumes allow multiple containers to access the same data simultaneously. This is especially useful in microservices architecture or when you have a separate container for logging, caching, or analytics that needs access to the same files or datasets.

Example: A container uploads user files to a volume, and another container processes or scans those files from the same volume.

#### 3. Clean Separation of Concerns

With volumes, you can move the application code and its data into separate entities, keeping your container structure cleaner and more modular. Your app is stateless, and the data is managed separately.

#### 4. Backup & Restore

Since volumes are managed by Docker, it's easier to back up or migrate them. You can archive a volume's content and transfer it across environments (e.g., dev to prod) or host machines.

## 5. Performance

Volumes are optimized for performance, especially compared to bind mounts. Docker can manage how volumes are stored on disk to ensure fast read/write speeds and reduce I/O overhead.

Volumes also avoid some file syncing issues seen with bind mounts on Windows/macOS.

## Types of Docker Volumes

Type	Description
<b>Named Volumes</b>	Created and managed by Docker. Stored in /var/lib/docker/volumes/. Useful for long-term persistence and sharing across containers.
<b>Anonymous Volumes</b>	Similar to named volumes but without a name. Created when a container runs and removed when the container is removed (unless retained).
<b>Bind Mounts</b>	Mounts a specific path from the host system into the container. Allows more control but less portability.
<b>tmpfs Mounts</b>	Stores data in the host system's memory only (RAM). Ideal for sensitive or temporary data (non-persistent).

## Key Differences Between Volume Types

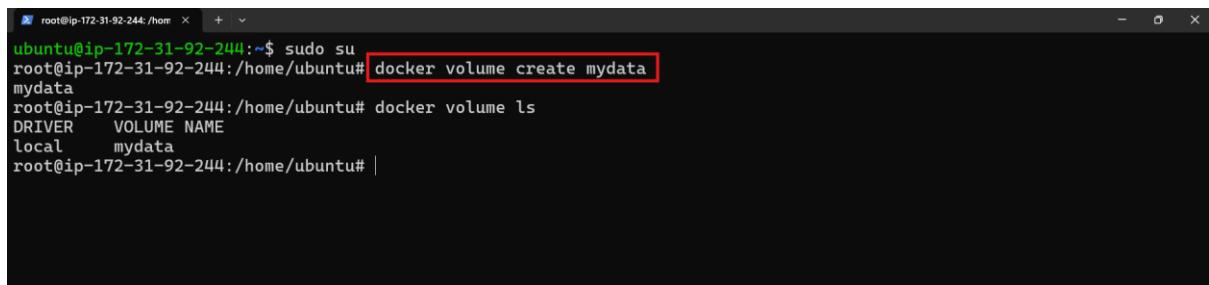
Feature	Named Volume	Anonymous Volume	Bind Mount	tmpfs Mount
<b>Persistence</b>	Yes	Yes (short-term)	Yes	No
<b>Location</b>	Managed by Docker	Managed by Docker	Host-defined	Memory
<b>Sharing</b>	Yes	Possible but hard	Yes	No
<b>Ease of Use</b>	High	Medium	Low	Medium
<b>Security</b>	High	Medium	Low	High
<b>Use Case Example</b>	Databases, volumes across containers	Ephemeral containers	Mounting config files	Sensitive credentials or cache

- **Named Volume:** Persistent and Docker-managed, ideal for sharing data like databases across containers.
- **Anonymous Volume:** Temporary Docker-managed storage, used when persistence isn't a priority.
- **Bind Mount:** Maps host directories into containers, useful for development or config files, but less secure.
- **tmpfs Mount:** Stores data in memory only, perfect for sensitive or temporary data that must not persist.

## 2. Demonstrate the use of Named Volume.

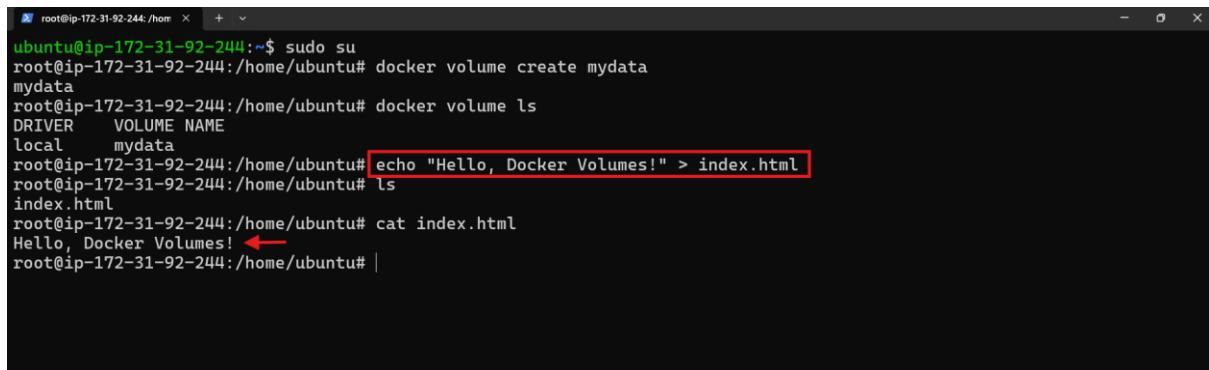
- Create a Docker Named volume named mydata.
- Attach volume to a Nginx Container
- Create an HTML file named index.html with some content (e.g., "Hello, Docker Volumes!") on your host machine. Copy this file into the mydata.
- Verify that the index.html file is accessible from within the container by starting a simple HTTP request.

Step 1: Create a Named Volume



```
ubuntu@ip-172-31-92-244:~$ sudo su
root@ip-172-31-92-244:/home/ubuntu# docker volume create mydata
mydata
root@ip-172-31-92-244:/home/ubuntu# docker volume ls
DRIVER      VOLUME NAME
local      mydata
root@ip-172-31-92-244:/home/ubuntu#
```

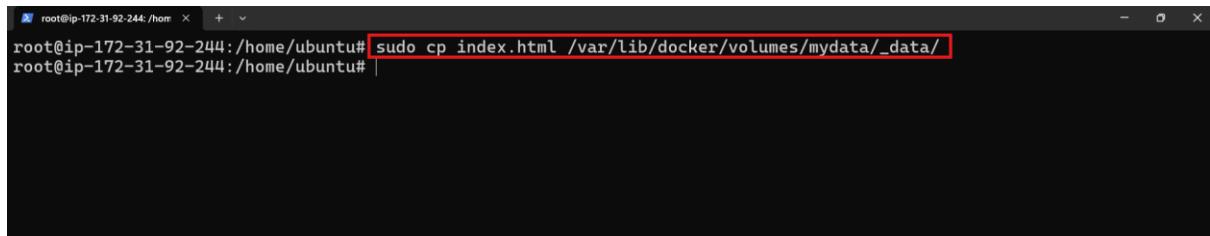
Step 2: Create index.html File on Host



```
ubuntu@ip-172-31-92-244:~$ sudo su
root@ip-172-31-92-244:/home/ubuntu# docker volume create mydata
mydata
root@ip-172-31-92-244:/home/ubuntu# docker volume ls
DRIVER      VOLUME NAME
local      mydata
root@ip-172-31-92-244:/home/ubuntu# echo "Hello, Docker Volumes!" > index.html
root@ip-172-31-92-244:/home/ubuntu# ls
index.html
root@ip-172-31-92-244:/home/ubuntu# cat index.html
Hello, Docker Volumes! ←
root@ip-172-31-92-244:/home/ubuntu#
```

Step 3: Copy index.html into the Volume

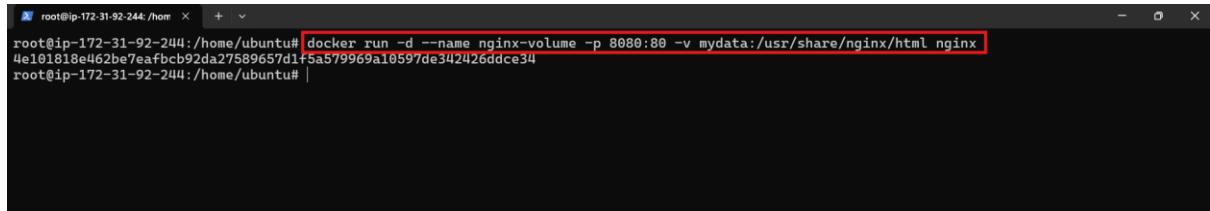
Docker named volumes are stored under `/var/lib/docker/volumes/<volume-name>/_data`.



```
root@ip-172-31-92-244:/home/ubuntu# sudo cp index.html /var/lib/docker/volumes/mydata/_data/
root@ip-172-31-92-244:/home/ubuntu#
```

## Step 4: Run Nginx Container with the Volume

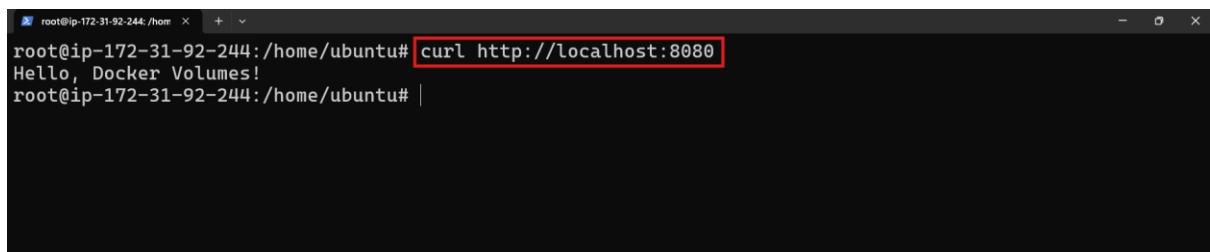
Mount the volume to NGINX's default web directory `/usr/share/nginx/html`



```
root@ip-172-31-92-244:/home/ubuntu# docker run -d --name nginx-volume -p 8080:80 -v mydata:/usr/share/nginx/html nginx
4e101818e462be7eafcb92da27589657d1f5a579969a10597de342426ddce34
root@ip-172-31-92-244:/home/ubuntu#
```

## Step 5: Verify the Setup

- ◆ Option 1: Using curl (Command Line)



```
root@ip-172-31-92-244:/home/ubuntu# curl http://localhost:8080
Hello, Docker Volumes!
root@ip-172-31-92-244:/home/ubuntu#
```

- ◆ Option 2: Open in Web Browser

Must allow port 8080 in the EC2 security group's inbound rules if want to access <http://<EC2-public-IP>:8080> from your browser (on local machine).



## Step 6: Clean Up

To stop and remove the container and volume

```

root@ip-172-31-92-244:/home/ubuntu$ sudo su
root@ip-172-31-92-244:/home/ubuntu# docker rm -f nginx-volume
nginx-volume
root@ip-172-31-92-244:/home/ubuntu# docker volume rm mydata
mydata
root@ip-172-31-92-244:/home/ubuntu# rm index.html
root@ip-172-31-92-244:/home/ubuntu#

```

### 3. Write a brief explanation of what Docker networks.

**Write the difference between host network and bridge network.**

Docker networks enable communication between containers, and between containers and the host machine or external systems. They provide isolated, secure communication channels so containers can interact with each other or services outside the container environment.

When Docker runs a container, it connects it to a network — which can be default or custom, and can have different configurations depending on the use case.

#### Types of Docker Networks

- **Bridge (default)** – Used by default for standalone containers.
- **Host** – Container shares the host's networking stack.
- **None** – No network access (isolated container).
- **Overlay** – Used in Docker Swarm for multi-host communication.
- **Macvlan** – Assigns a MAC address to a container, making it appear as a physical device.

#### Difference Between Host and Bridge Network

Feature	Bridge Network	Host Network
<b>Isolation</b>	Containers have their own network namespace	Shares the host's network namespace
<b>IP Address</b>	Container gets a private IP (e.g., 172.x.x.x)	Container uses the host's IP address
<b>Port Mapping</b>	Required (e.g., -p 8080:80)	Not required (container uses host ports directly)
<b>Use Case</b>	Default for container-to-container communication	When performance is critical (e.g., low latency)

<b>Security</b>	More secure due to isolation	Less secure; container has access to host network
<b>Performance</b>	Slight overhead due to network translation	Better performance, no network translation
<b>Docker Compose Support</b>	Yes	Limited (must be manually configured)

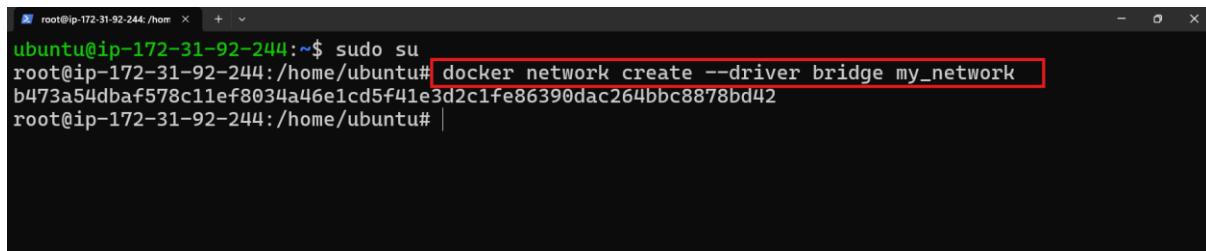
### Example

- **Bridge:** Microservices app where containers need to talk via service names.
- **Host:** High-performance apps (like game servers or monitoring tools) that need raw access to network interfaces.

## 4. Demonstrate the use of Custom Network

- Create a custom bridge network named **my\_network**.
- Start two containers, one using the **nginx** image and another using the **httpd** image.
- Attach both containers to the **my\_network** network.
- Test Network Connectivity: Ensure that the **nginx** container can communicate with the **httpd** container over the custom network. You can do this by sending an HTTP request from one container to another using tools like **curl**.

### Step 1: Create a Custom Bridge Network



```
root@ip-172-31-92-244:~$ sudo su
ubuntu@ip-172-31-92-244:~/home/ubuntu# docker network create --driver bridge my_network
b473a54dbaf578c11ef8034a46e1cd5f41e3d2c1fe86390dac264bbc8878bd42
root@ip-172-31-92-244:~/home/ubuntu#
```

### Step 2: Start the Containers on the Custom Network

```
root@ip-172-31-92-244:/home# docker run -d --name my-nginx nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
3da95a905ed5: Pull complete
6c8e51cf0087: Pull complete
9bbbd7ee45b7: Pull complete
48670a58a68f: Pull complete
ce7132063a56: Pull complete
23e05839d684: Pull complete
ee95256df030: Pull complete
Digest: sha256:93230cd54060f497430c7a120e2347894846a81b6a5dd2110f7362c5423b4abc
Status: Downloaded newer image for nginx:latest
90b53729d8bfce93b612b3649f3260bf5812aeb6878fc9b434e869c2cb22da43 ←
root@ip-172-31-92-244:/home/ubuntu# docker run -d --name my-httpd httpd
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3da95a905ed5: Already exists
1a50d37c990c: Pull complete
4f4fb700ef54: Pull complete
365277d54dac: Pull complete
d812016dda12: Pull complete
816d153af128: Pull complete
Digest: sha256:1ae8051591a5ded56e4a3d7399c423e940e8475ad0e5adb82e6e10893fe9b365
Status: Downloaded newer image for httpd:latest
37443a62227346bd5e206e4b62b363f7b047bcab9423f6238605803a3b75ced0 ←
root@ip-172-31-92-244:/home/ubuntu#
```

### Step 3: Connect Containers to the Custom Network

```
37443a62227346bd5e206e4b62b363f7b047bcab9423f6238605803a3b75ced0
root@ip-172-31-92-244:/home/ubuntu#
root@ip-172-31-92-244:/home/ubuntu# docker network connect my_network my-nginx
root@ip-172-31-92-244:/home/ubuntu# docker network connect my_network my-httpd
root@ip-172-31-92-244:/home/ubuntu#
```

### Step 4: Test Network Connectivity

```
root@ip-172-31-92-244:/home/ubuntu# docker network connect my_network my-nginx
root@ip-172-31-92-244:/home/ubuntu# docker network connect my_network my-httpd
root@ip-172-31-92-244:/home/ubuntu#
root@ip-172-31-92-244:/home/ubuntu# docker exec -it my-nginx sh
# curl http://my-httpd
<html><body><h1>It works!</h1></body></html>
#
```

### Step 5: Cleanup (Disconnect & Remove)

```

root@ip-172-31-92-244:/home/ubuntu# docker network disconnect my_network my-nginx
root@ip-172-31-92-244:/home/ubuntu# docker network disconnect my_network my-httpd
root@ip-172-31-92-244:/home/ubuntu# docker rm -f my-nginx my-httpd
my-nginx
my-httpd
root@ip-172-31-92-244:/home/ubuntu# docker network rm my_network
my_network
root@ip-172-31-92-244:/home/ubuntu#

```

## 5. Write a note on Dockerfile with usage of its attributes

A Dockerfile is a text file that contains step-by-step instructions on how to build a Docker image. It defines the base image, sets up the environment, copies application code, installs dependencies, exposes ports, and specifies the command to run the application.

It is the blueprint for creating Docker images in a repeatable, automated way.

Instruction	Description
FROM	Sets the <b>base image</b> . It must be the first instruction (except for optional ARGs). Example: FROM ubuntu:20.04
LABEL	Adds metadata. Example: LABEL version="1.0"
ENV	Sets environment variables. Example: ENV PATH="/app/bin:\$PATH"
WORKDIR	Sets the working directory for subsequent commands. Example: WORKDIR /app
COPY	Copies files from host to container. Example: COPY . /app
ADD	Like COPY, but supports remote URLs and auto-extracts archives.
RUN	Executes commands during image build. Example: RUN apt-get update && apt-get install -y nginx
CMD	Sets default command to run when the container starts. Example: CMD ["nginx", "-g", "daemon off;"]
ENTRYPOINT	Sets the main command that cannot be overridden easily.
EXPOSE	Documents the port the app uses (does not publish it).
ARG	Defines build-time variables. Example: ARG version=1.0
VOLUME	Creates a mount point for persistent data.
USER	Sets the user to run the image as.
HEALTHCHECK	Defines how Docker should check if the container is healthy.
MAINTAINER	To indicate who built or maintains the image.

## Benefits of Using Dockerfile

- Automation of image creation
- Consistency across environments
- Easier version control and debugging
- Reusability and modular builds

## 6. What is difference between CMD and ENTRYPOINT?

Dockerfile instructions CMD and ENTRYPOINT specify which command should be executed when a container starts. But they behave differently and have different functions.

Feature	CMD	ENTRYPOINT
Purpose	Sets a <b>default command or arguments</b>	Defines a <b>fixed command to run</b>
Overridable	Yes, completely overridden by docker run	No, only additional arguments are appended
Command Type	Can be used as <b>default parameters</b>	Treated as the <b>main executable</b>
Typical Use Case	Specify default behavior that can be changed	Enforce consistent behavior (e.g., always run app)
Preferred Format	Shell or exec (["arg1", "arg2"])	<b>Exec format</b> (["executable", "param"])
Flexibility	More flexible (fully replaceable)	Less flexible (command is fixed)
Ignored if ENTRYPOINT is set	Yes	No (always runs)
Example	CMD ["npm", "start"]	ENTRYPOINT ["nginx"]

## 7. What is difference between ADD and COPY?

To move files or directories from the host computer to the Docker image, use the ADD and COPY Dockerfile instructions. Their use cases and functionalities vary.

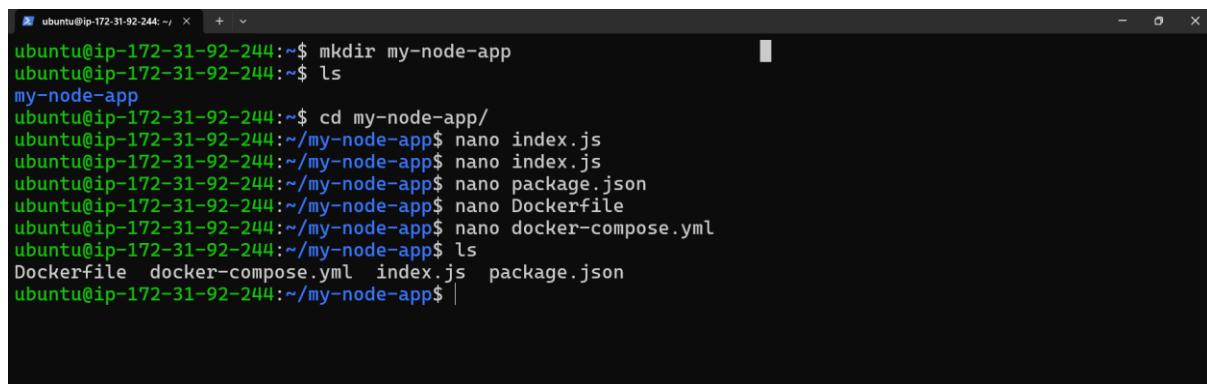
Feature	COPY	ADD
Basic Function	Copies files/directories to container image	Copies files/directories <b>with extra features</b>

<b>Archive Extraction</b>	No	Yes, extracts .tar, .tar.gz, etc. automatically
<b>Remote URL Support</b>	No	Yes, can fetch files from remote URLs
<b>Clarity &amp; Simplicity</b>	Simple and explicit	Can behave differently depending on source
<b>Preferred Use</b>	For basic file/folder copying	When archive extraction or URL download is needed
<b>Best Practice</b>	Recommended for most use cases	Use only when its special features are required
<b>Security</b>	More secure due to limited functionality	Slightly riskier (downloads and auto-extraction)

## 8. Write a Dockerfile to run Nodejs application build an image from it and create a container using that image (also include persistent volume and network in Dockerfile).

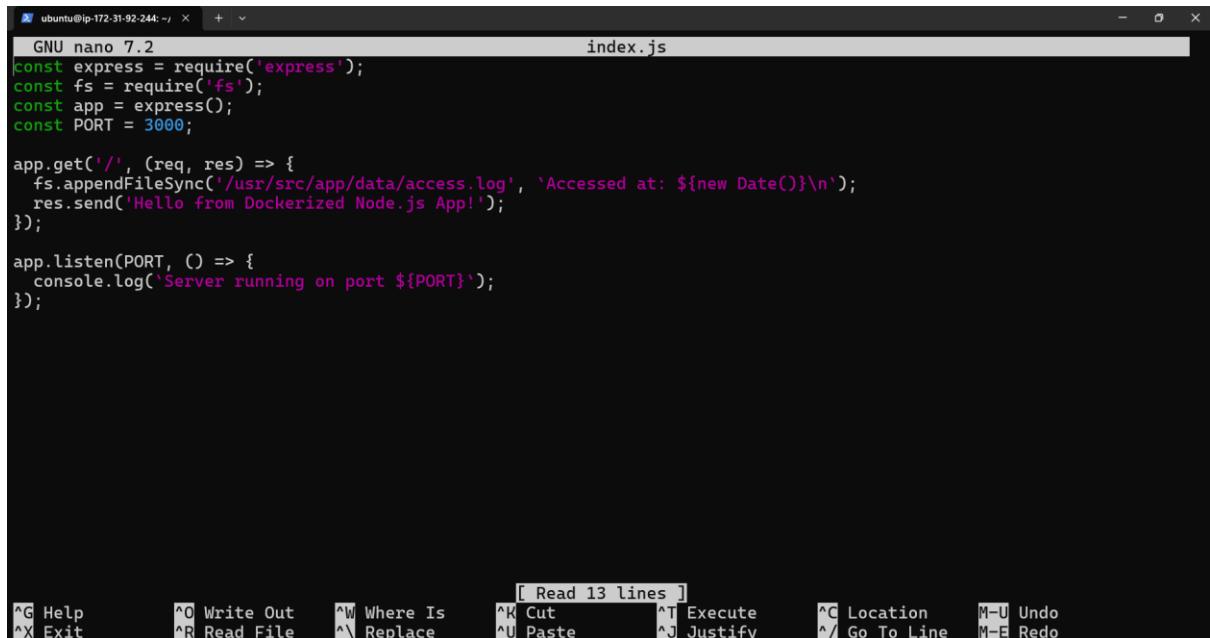
### Step 1: Create Project Structure

```
my-node-app/
├── Dockerfile
├── docker-compose.yml
└── index.js
└── package.json
```



```
ubuntu@ip-172-31-92-244:~$ mkdir my-node-app
ubuntu@ip-172-31-92-244:~$ ls
my-node-app
ubuntu@ip-172-31-92-244:~$ cd my-node-app/
ubuntu@ip-172-31-92-244:~/my-node-app$ nano index.js
ubuntu@ip-172-31-92-244:~/my-node-app$ nano index.js
ubuntu@ip-172-31-92-244:~/my-node-app$ nano package.json
ubuntu@ip-172-31-92-244:~/my-node-app$ nano Dockerfile
ubuntu@ip-172-31-92-244:~/my-node-app$ nano docker-compose.yml
ubuntu@ip-172-31-92-244:~/my-node-app$ ls
Dockerfile  docker-compose.yml  index.js  package.json
ubuntu@ip-172-31-92-244:~/my-node-app$ |
```

## Step 2: Create index.js (Node.js App)



```
ubuntu@ip-172-31-92-244:~$ nano index.js
GNU nano 7.2                                         index.js
const express = require('express');
const fs = require('fs');
const app = express();
const PORT = 3000;

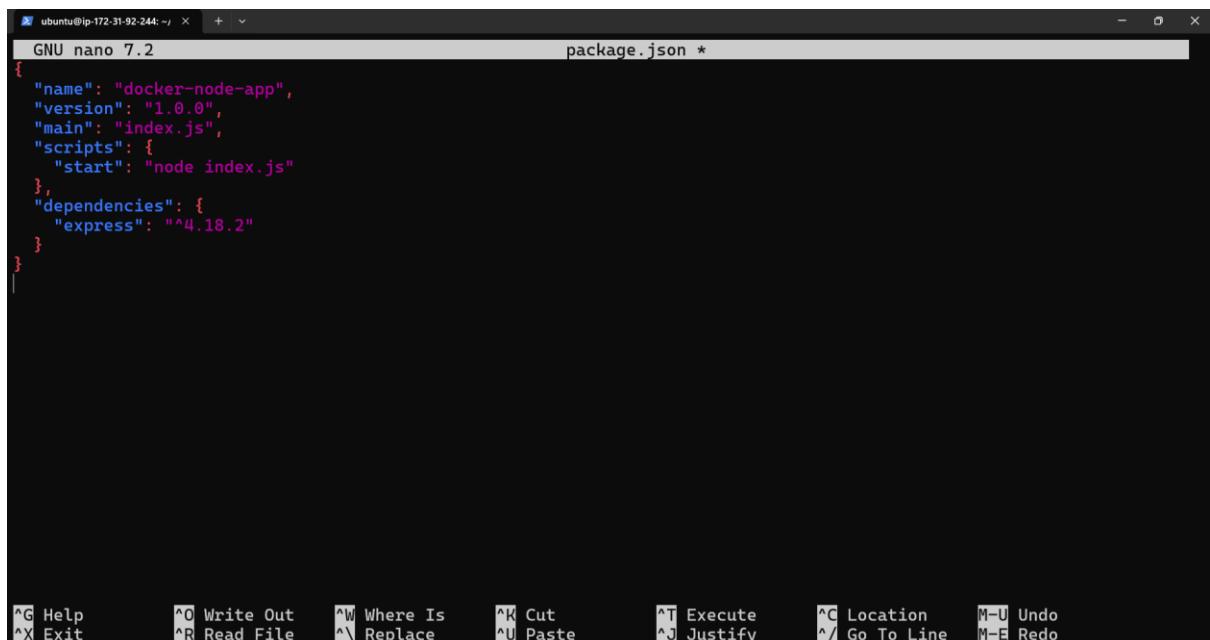
app.get('/', (req, res) => {
  fs.appendFileSync('/usr/src/app/data/access.log', `Accessed at: ${new Date()}\n`);
  res.send('Hello from Dockerized Node.js App!');
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

[ Read 13 lines ]

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

## Step 3: Create package.json



```
ubuntu@ip-172-31-92-244:~$ nano package.json *
{
  "name": "docker-node-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

[ Read 13 lines ]

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

## Step 4: Create Dockerfile

The screenshot shows a terminal window titled "ubuntu@ip-172-31-92-244: ~" running the "nano" text editor. The file being edited is named "Dockerfile". The content of the file is:

```
FROM node:18
WORKDIR /usr/src/app
COPY package*.json .
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "index.js"]
```

At the bottom of the terminal window, there is a status bar with the following text:

File Name to Write: Dockerfile | M-D DOS Format M-A Append M-B Backup File  
^G Help ^C Cancel M-M Mac Format M-P Prepend ^T Browse

## Step 5: Create docker-compose.yml

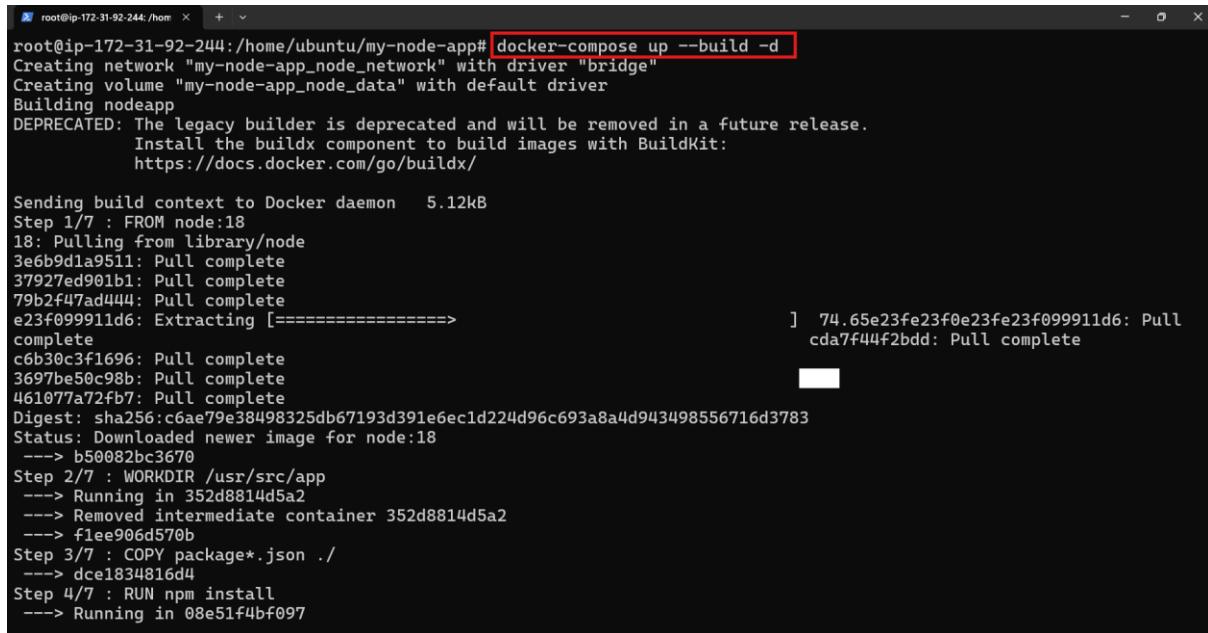
The screenshot shows a terminal window titled "ubuntu@ip-172-31-92-244: ~" running the "nano" text editor. The file being edited is named "docker-compose.yml". The content of the file is:

```
version: '3.8'
services:
  nodeapp:
    build: .
    container_name: node-container
    ports:
      - "3000:3000"
    volumes:
      - node_data:/usr/src/app/data
    networks:
      - node_network
volumes:
  node_data:
networks:
  node_network:
    driver: bridge
```

At the bottom of the terminal window, there is a status bar with the following text:

File Name to Write: docker-compose.yml | M-D DOS Format M-A Append M-B Backup File  
^G Help ^C Cancel M-M Mac Format M-P Prepend ^T Browse

## Step 6: Build and Run the Application

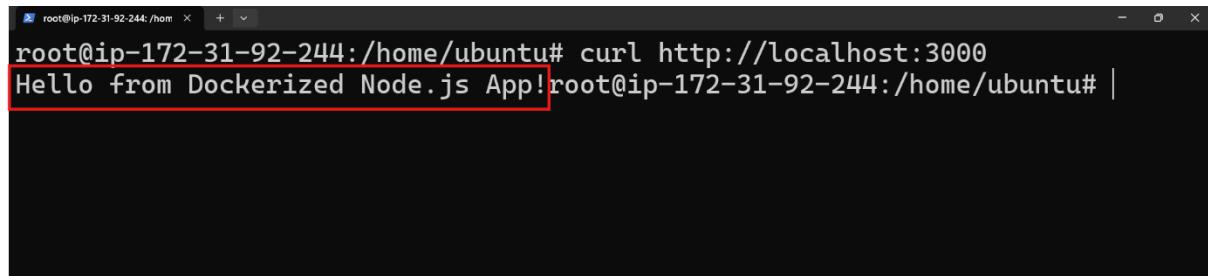


```
root@ip-172-31-92-244:/home/ubuntu/my-node-app# docker-compose up --build -d
Creating network "my-node-app_node_network" with driver "bridge"
Creating volume "my-node-app_node_data" with default driver
Building nodeapp
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
  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 node:18
18: Pulling from library/node
3e6b9d1a9511: Pull complete
37927ed9e1b1: Pull complete
79b2f47ad444: Pull complete
e23f099911d6: Extracting [=====] 74.65e23fe23f0e23fe23f099911d6: Pull complete
cda7f44f2bdd: Pull complete
c6b30c3f1696: Pull complete
3697be50c98b: Pull complete
461077a72fb7: Pull complete
Digest: sha256:c6ae79e38498325db67193d391e6ec1d224d96c693a8a4d943498556716d3783
Status: Downloaded newer image for node:18
--> b50082bc3670
Step 2/7 : WORKDIR /usr/src/app
--> Running in 352d8814d5a2
--> Removed intermediate container 352d8814d5a2
--> f1ee906d570b
Step 3/7 : COPY package*.json ./
--> dce1834816d4
Step 4/7 : RUN npm install
--> Running in 08e51f4bf097
```

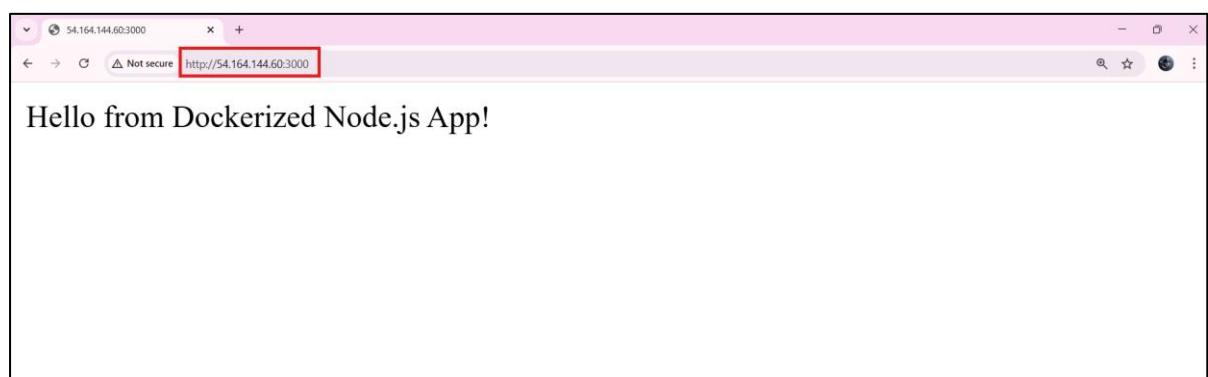
## Step 7: Test the Application

- ◆ Option 1: Using curl (Command Line)

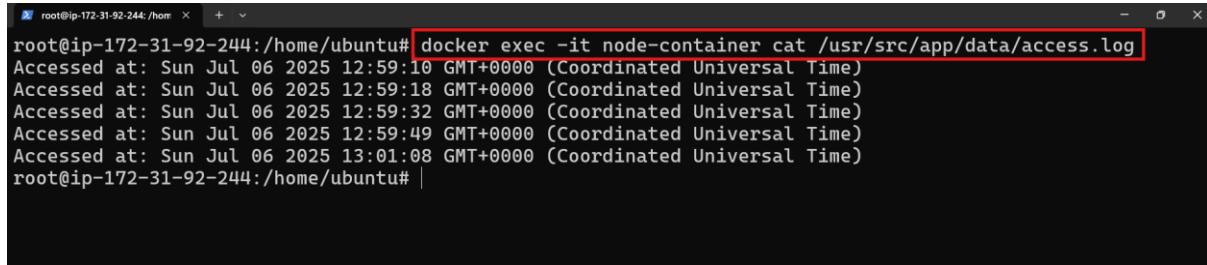


```
root@ip-172-31-92-244:/home/ubuntu# curl http://localhost:3000
Hello from Dockerized Node.js App!root@ip-172-31-92-244:/home/ubuntu# |
```

- ◆ Option 2: Open in Web Browser



## Step 8: Check Persistent Log Volume

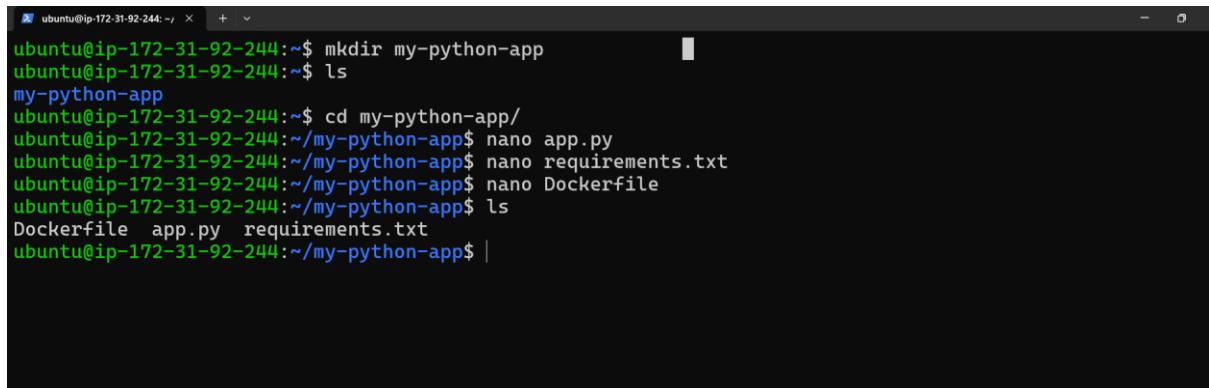


```
root@ip-172-31-92-244:/home/ubuntu# docker exec -it node-container cat /usr/src/app/data/access.log
Accessed at: Sun Jul 06 2025 12:59:10 GMT+0000 (Coordinated Universal Time)
Accessed at: Sun Jul 06 2025 12:59:18 GMT+0000 (Coordinated Universal Time)
Accessed at: Sun Jul 06 2025 12:59:32 GMT+0000 (Coordinated Universal Time)
Accessed at: Sun Jul 06 2025 12:59:49 GMT+0000 (Coordinated Universal Time)
Accessed at: Sun Jul 06 2025 13:01:08 GMT+0000 (Coordinated Universal Time)
root@ip-172-31-92-244:/home/ubuntu#
```

9. Write a Dockerfile to create a python application build image from it and push that image to private repository of Docker hub.

### Step 1: Create Your Python Project

```
my-python-app/
├── app.py
├── requirements.txt
└── Dockerfile
```



```
ubuntu@ip-172-31-92-244:~$ mkdir my-python-app
ubuntu@ip-172-31-92-244:~$ ls
my-python-app
ubuntu@ip-172-31-92-244:~$ cd my-python-app/
ubuntu@ip-172-31-92-244:~/my-python-app$ nano app.py
ubuntu@ip-172-31-92-244:~/my-python-app$ nano requirements.txt
ubuntu@ip-172-31-92-244:~/my-python-app$ nano Dockerfile
ubuntu@ip-172-31-92-244:~/my-python-app$ ls
Dockerfile app.py requirements.txt
ubuntu@ip-172-31-92-244:~/my-python-app$ |
```

## Step 2: Create Python App (app.py)

The screenshot shows a terminal window titled "ubuntu@ip-172-31-92-244: ~" with the file "app.py" open in nano editor. The code defines a simple Flask application that returns "Hello from Python Docker App!" when the root URL is accessed. It also includes a run command to start the app on host '0.0.0.0' at port 5000.

```
GNU nano 7.2                                         app.py *
# app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello from Python Docker App!"

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

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

## Step 3: Create requirements.txt

The screenshot shows a terminal window titled "ubuntu@ip-172-31-92-244: ~" with the file "requirements.txt" open in nano editor. The file contains the single line "flask".

```
GNU nano 7.2                                         requirements.txt *
flask

[ New File ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute   ^C Location  M-U Undo
^X Exit     ^R Read File  ^\ Replace    ^U Paste     ^J Justify   ^/ Go To Line M-E Redo
```

## Step 4: Create Dockerfile

```
ubuntu@ip-172-31-92-244:~$ nano Dockerfile
GNU nano 7.2                               Dockerfile *
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

The screenshot shows a terminal window with a nano text editor open. The file is named 'Dockerfile'. The content of the file is a standard Dockerfile with instructions for a Python application. The terminal has a dark theme and includes a menu bar at the top and a set of keyboard shortcuts at the bottom.

## Step 5: Build Docker Image

```
root@ip-172-31-92-244:/home/ubuntu/my-python-app# docker build -t my-python-app .
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 4.096kB
Step 1/7 : FROM python:3.10-slim
3.10-slim: Pulling from library/python
3da95a905ed5: Already exists
0ebcc011f0ec: Pull complete
92d63ec5cbeb: Pull complete
64b78282ca88: Pull complete
Digest: sha256:9dd6774a1276178f94b0cc1fb1f0edd980825d0ea7634847af9940b1b6273c13
Status: Downloaded newer image for python:3.10-slim
--> 563dffbd425
Step 2/7 : WORKDIR /app
--> Running in 24a4115cb4ed
--> Removed intermediate container 24a4115cb4ed
--> 951a37cc5e11
Step 3/7 : COPY requirements.txt .
--> bffd4eddf54
Step 4/7 : RUN pip install --no-cache-dir -r requirements.txt
--> Running in 96b36f85cb0b
Collecting flask
  Downloading flask-3.1.1-py3-none-any.whl (103 kB)
  _____ 103.3/103.3 kB 4.9 MB/s eta 0:00:00
Collecting markupsafe>=2.1.1
  Downloading MarkupSafe-3.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Collecting werkzeug>=3.1.0
```

The screenshot shows a terminal window running as root. It executes the 'docker build' command with the '-t' option to tag the resulting image as 'my-python-app'. The output shows the build process, including pulling the Python base image and copying the 'requirements.txt' file into the container. The terminal uses a light-colored background with black text.

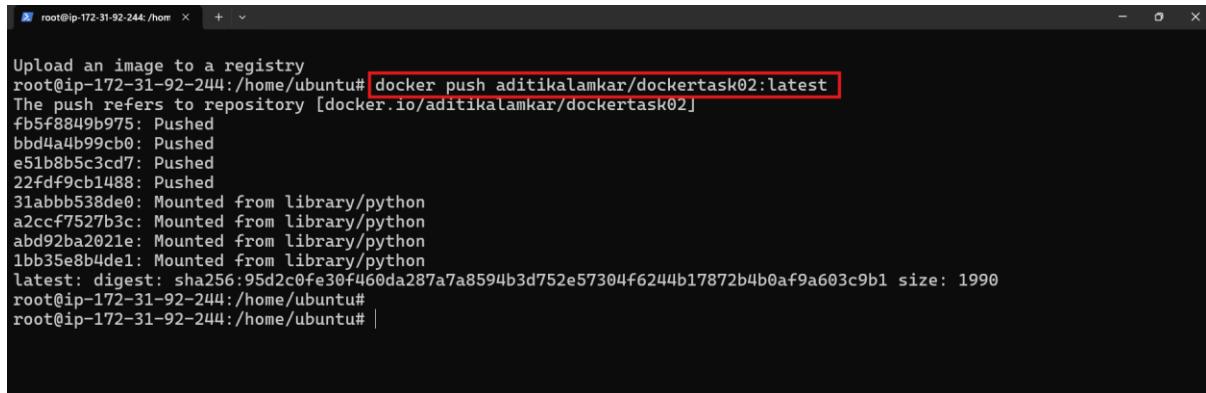
## Step 6: Log In to Docker Hub

```
ubuntu@ip-172-31-92-244:~$ sudo su
root@ip-172-31-92-244:/home/ubuntu# docker login -u aditikalamkar
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

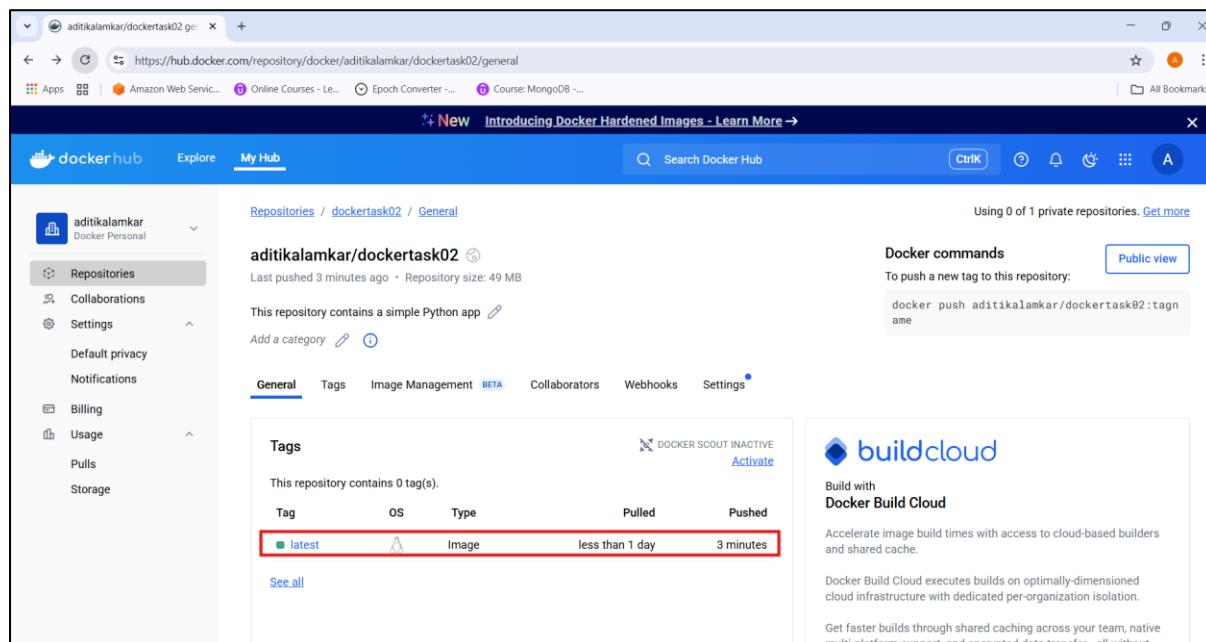
Login Succeeded
root@ip-172-31-92-244:/home/ubuntu# |
```

The screenshot shows a terminal window running as root. It runs the 'sudo su' command to become root. Then it runs 'docker login' with the username 'aditikalamkar'. A password prompt appears, followed by a warning about password storage. Finally, the message 'Login Succeeded' is displayed. The terminal has a dark theme with a light-colored status bar at the bottom.

## Step 7: Push the Image to Docker Hub

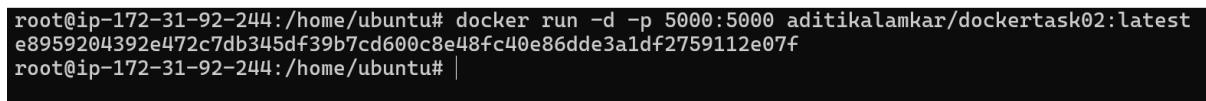


```
root@ip-172-31-92-244:/home/ubuntu# docker push aditikalamkar/dockertask02:latest
The push refers to repository [docker.io/aditikalamkar/dockertask02]
fb5f8849b975: Pushed
bbd4a4b99cb0: Pushed
e51b8b5c3cd7: Pushed
22fdf9cb1488: Pushed
31abb538de0: Mounted from library/python
a2ccf7527b3c: Mounted from library/python
abd92ba2021e: Mounted from library/python
1bb35e8b4de1: Mounted from library/python
latest: digest: sha256:95d2c0fe30f460da287a7a8594b3d752e57304f6244b17872b4b0af9a603c9b1 size: 1990
root@ip-172-31-92-244:/home/ubuntu#
```

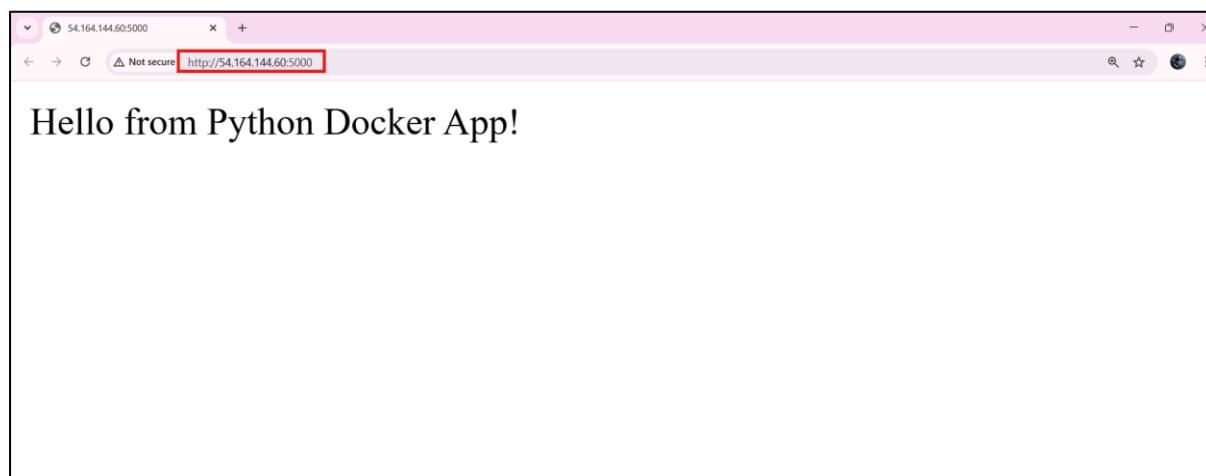


The screenshot shows the Docker Hub interface for the repository `aditikalamkar/dockertask02`. The left sidebar shows the user's profile and repository list. The main page displays the repository details, including the latest pushed tag, which is `latest`.

## Step 8: Run the Image



```
root@ip-172-31-92-244:/home/ubuntu# docker run -d -p 5000:5000 aditikalamkar/dockertask02:latest
e8959204392e472c7db345df39b7cd600c8e48fc40e86dde3a1df2759112e07f
root@ip-172-31-92-244:/home/ubuntu#
```



The screenshot shows a web browser displaying the output of the Docker container. The URL `http://54.164.144.60:5000` is shown in the address bar, and the page content reads "Hello from Python Docker App!".

**10. Prepare a documentation in README.md with proper screenshots and push that file to your Git hub repository.**

<https://github.com/aditikalamkar/FCTDevOpsLearning.git>