Docker Assignment_2
Name : Vishal Dasharath Bobhate

Docker Task2

Date: 04/07/25

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 – What They Are and Why We Use Them

Docker volumes are basically a way to **store data that needs to stay around even if your container stops, restarts, or gets deleted**.

Normally, anything you do inside a container (like saving files) disappears when the container goes away. Volumes solve this by providing **persistent storage**, separate from the container's own filesystem.

We use volumes because:

- They **keep important data safe** beyond the life of a container.
- They make it easy to **share data between containers.**
- They are **faster and easier to back up or move around** compared to bind mounting directories manually.

Types of Volumes in Docker (There are **three main types)**:

1. **Named Volumes**
   - These are created and managed by Docker.
   - Docker stores them in a special location (/var/lib/docker/volumes).
   - You give them a name, and Docker keeps track of them for you.
   - Best for most situations where you want persistent data.
2. **Anonymous Volumes**
   - Similar to named volumes, but you **don't give them a name**.
   - Docker assigns them a random ID.
   - They're often created automatically if you declare a VOLUME in your Dockerfile.
   - Harder to manage because you don't have an easy name to reference.
3. **Bind Mounts**
   - Here, you **specify an exact path on your host machine** to mount into the container.
   - Useful when you want your container to access or update specific files on your host (like source code during development).
   - Gives you more control but requires you to manage the path yourself.

Difference Between Them (in plain words):

- **Named Volumes:**
  Docker manages them for you. Easy to use, good for storing databases or app data.
- **Anonymous Volumes:**
  Similar to named ones but without a clear name. Can be confusing to clean up.
- **Bind Mounts:**
  You pick exactly where the data lives on your computer. More control, often used in development.

- Vishal bobahte

Example:

- **Named volume:**
- docker run -v mydata:/app/data myimage
  (Here, mydata is the volume name.)
- **Bind mount:**
- docker run -v /home/user/data:/app/data myimage
  (Here, /home/user/data is a folder on your host.)

In Short:
Docker volumes are how you **keep your data safe and persistent** in containers.
Use **named volumes** most of the time, and **bind mounts** when you need direct access to host files.

---

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.

```
 39*
 40  docker volume create mydata
 41  echo "hello docker volume" > index.html
 42  ls
 43  docker run -d --name bob_cont -p 80:80 -v mydata:/usr/share/nginx/html/ nginx
 44  docker container ls
 45  docker cp index.html bob_cont:/usr/share/nginx/html/index.html
 46  curl http://localhost
 47  history
root@ip-172-16-20-112 ec2-user]#
root@ip-172-16-20-112 ec2-user]#
```

Docker Assignment_2
Name : Vishal Dasharath Bobhate

3. Write a brief explanation of what Docker networks.
Write the difference between host network and bridge network.

What are Docker Networks?

Docker networks are how containers communicate with each other and with the outside world.
Think of them as **virtual networks Docker creates inside your machine** so containers can:

- Talk to each other securely
- Access the internet (if needed)
- Be isolated from other containers

By default, Docker creates a few networks automatically. When you run a container, you can attach it to one or more of these networks.

Difference Between Host Network and Bridge Network

Here's a simple comparison:

Feature	Bridge Network	Host Network
How it works	Docker creates a **private virtual network**. Each container gets its **own IP address**. Docker manages routing between containers and the host.	The container **shares the host machine's network stack** directly. No separate container IP address.
Isolation	More isolated—containers can talk to each other through the bridge but are separate from the host.	Less isolated—container ports are the **same as the host's ports**.
Use Case	Most common—good for apps that don't need special network performance. Default network when you run containers.	Useful for high-performance apps or tools (like monitoring agents) that need to listen on the host network directly.
Port Mapping	You **must** map container ports to host ports (-p).	No need to map ports—container ports are the host ports.

In short:

- **Bridge Network:** default, more isolated, each container has its own IP.
- **Host Network:** less isolated, container shares host's network.

- Vishal bobahte

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`.

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

Absolutely—here's a **clear, human-style note about Dockerfile**, with all the important attributes explained simply:

What is a Dockerfile?

A **Dockerfile** is a plain text file that contains **step-by-step instructions to build a Docker image**.

Think of it as a **recipe** that tells Docker:

- Which base image to start from
- What files to copy
- What dependencies to install
- How to configure the environment
- What command to run when a container starts

When you build a Dockerfile, Docker runs each instruction **in order**, creating layers in your image.

FROM

- **Usage:** Sets the **base image** your image will build upon.
- **Example:**
- FROM python:3.11-slim
- You *must* have a FROM line in every Dockerfile.

WORKDIR

- **Usage:** Sets the **working directory** inside the image for all subsequent commands.
- **Example:**
- WORKDIR /app

COPY

- **Usage:** Copies files or directories from your **host machine** into the image.
- **Example:**
- COPY . .
  (Copies everything from current folder on host to the working directory in the image.)

ADD

- Like COPY, but also supports:
  - Extracting tar archives automatically.
  - Downloading files via URLs.
- Use COPY for simple copying; ADD for extra features.
- Example:
- ADD https://example.com/file.tar.gz /data/

RUN

- **Usage:** Executes a command **while building the image**.
- Usually used to install packages or do setup work.
- Example:
- RUN apt-get update && apt-get install -y curl

CMD

- **Usage:** Sets the **default command** that runs when you start a container.
- Can be overridden when you docker run.

- Example:
- CMD ["python", "app.py"]

EXPOSE

- **Usage:** Tells Docker which **port your app will use**.
- *Note:* It doesn't actually publish the port—you still need -p when you run the container.
- Example:
- EXPOSE 5000

ENV

- **Usage:** Sets **environment variables** inside the container.
- Example:
- ENV APP_ENV=production

VOLUME

- **Usage:** Declares a mount point for **persistent or shared data**.
- Example:
- VOLUME ["/data"]

ENTRYPOINT

- **Usage:** Sets the **main command** for the container that is harder to override.
- Often used in combination with CMD.
- Example:
- ENTRYPOINT ["python"]
- CMD ["app.py"]
  This would run:
  python app.py

LABEL

- **Usage:** Adds metadata to your image (e.g., author, description).
- Example:
- LABEL maintainer="yourname@example.com"

Docker Assignment_2
Name : Vishal Dasharath Bobhate

Example Dockerfile (All in One)

```
FROM node:20
WORKDIR /usr/src/app
COPY package*.json .
RUN npm install
COPY . .
ENV NODE_ENV=production
EXPOSE 3000
VOLUME ["/data"]
CMD ["node", "app.js"]
```

In Short

A **Dockerfile**:

- Is a blueprint to create images.
- Uses instructions like FROM, COPY, RUN, and CMD.
- Helps automate builds so you can deploy apps consistently.

---

6. What is difference between CMD and ENTRYPOINT?

Both CMD and ENTRYPOINT define **what command runs when your Docker container starts**, but they behave a bit differently.

Feature	**CMD**	**ENTRYPOINT**
Purpose	Default command or args	Always runs as main command
Can be overridden?	☑ Yes (fully replaced)	☑ Yes (acts as arguments only)
Typical use case	Flexible command	Force a specific executable
Works alone?	☑ Yes	☑ Yes
Used with CMD often?	No need, unless for default args	Often used **with CMD** for args

CMD (Default Command)

- CMD provides **default arguments** or a default command.
- It **can be overridden** when you run the container using docker run.

ENTRYPOINT (Fixed Start Command)

- ENTRYPOINT is the **main command that always runs**.
- Anything you pass in docker run becomes **arguments to ENTRYPOINT**.
- Used when you want to **enforce a specific command**, like python, bash, or your own script.

7. What is difference between ADD and COPY?

Both ADD and COPY are used to **copy files and directories into your Docker image**, but there are **important differences**:

COPY

Just **copies files or folders from your host to the image**, nothing more.

- When you just want to move files.
- This is the **most common and recommended** for clarity.

ADD

- Copies files and folders **like COPY**, **PLUS:**
  - Can **extract tar archives automatically** (.tar, .tar.gz, etc.).
  - Can **download files from a URL**.

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).

```
  GNU nano 8.3
FROM node:latest
WORKDIR /usr/src/app
COPY package.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "app.js"]
VOLUME /data/my_volume
```

```
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 240B
 => [internal] load metadata for docker.io/library/node:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/node:latest@sha256:256a2e7037e745228f7630d578e6c1d327ab4c0a8e401c
 => [internal] load build context
 => => transferring context: 1.52kB
 => CACHED [2/5] WORKDIR /usr/src/app
 => CACHED [3/5] COPY package.json ./
 => CACHED [4/5] RUN npm install
 => CACHED [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:21c7ef730c03410bc24481ea7d49b01c9cdf7031d6fbbd69090748048c1d9f8d
 => => naming to docker.io/library/node_js
[root@ip-172-16-20-112 ec2-user]# ls
Dockerfile  app.js  index.html  package.json
[root@ip-172-16-20-112 ec2-user]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
node_js      latest    21c7ef730c03   58 seconds ago   1.14GB
nginx        latest    9592f5595f2b   13 days ago      192MB
httpd        latest    e6af1d8a44e5   5 months ago     148MB
busybox      latest    6d3e4188a38a   9 months ago     4.28MB
[root@ip-172-16-20-112 ec2-user]# docker run -d --name node_cont node_js
d932d9b9b9e487178f0afec87bc18dc1d3a75fc67a78108a9c0033e9ef399cca
[root@ip-172-16-20-112 ec2-user]# docker exec -it node_cont /bin/bash
root@d932d9b9b9e4:/usr/src/app# curl http://localhost
curl: (7) Failed to connect to localhost port 80 after 0 ms: Couldn't connect to server
root@d932d9b9b9e4:/usr/src/app# curl http://localhost:3000
Hello from Node.js app with volume and network!
root@d932d9b9b9e4:/usr/src/app#
root@d932d9b9b9e4:/usr/src/app#
root@d932d9b9b9e4:/usr/src/app#
```

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

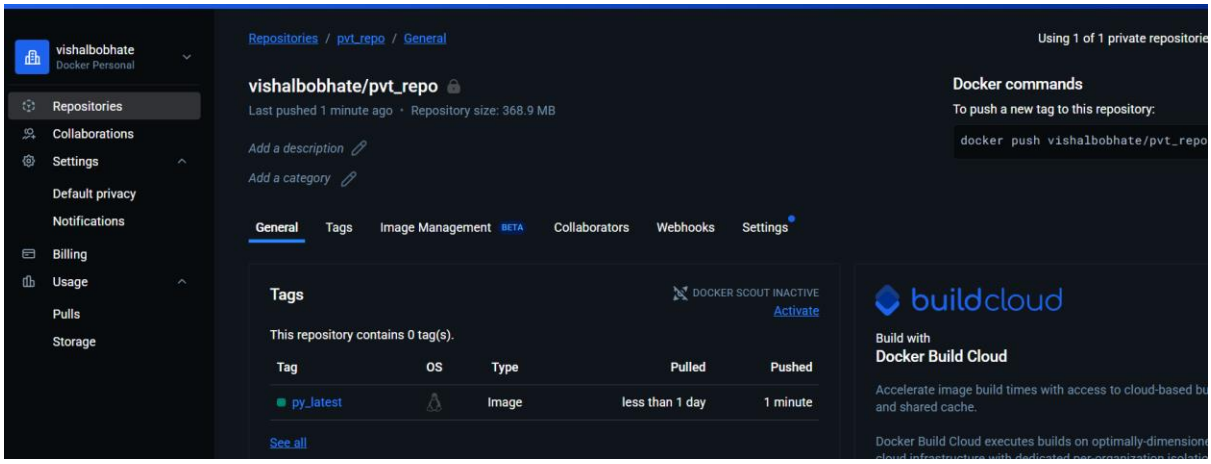
```
  GNU nano 8.3
FROM python:latest
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python","app.py"]
```

Docker Assignment_2
Name : Vishal Dasharath Bobhate

```
[root@ip-172-16-20-112 pythonApp]# nano Dockerfile
[root@ip-172-16-20-112 pythonApp]# docker build -t pythonimg .
[+] Building 3.9s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 250B
 => [internal] load metadata for docker.io/library/python:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:latest@sha256:a6af772cf98267c48c145928cbeb35bd8e89b610acd70f93
 => [internal] load build context
 => => transferring context: 630B
 => CACHED [2/5] WORKDIR /app
 => CACHED [3/5] COPY requirements.txt .
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => writing image sha256:c9b15d8115757344980edd36560bfdba0f57ee29ed06f7c749b4384fff8b6490
 => => naming to docker.io/library/pythonimg
[root@ip-172-16-20-112 pythonApp]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
pythonimg    latest    c9b15d811575   13 seconds ago   1.03GB
node_js      latest    21c7ef730c03   19 minutes ago   1.14GB
nginx        latest    9592f5595f2b   13 days ago      192MB
httpd        latest    e6af1d8a44e5   5 months ago     148MB
busybox      latest    6d3e4188a38a   9 months ago     4.28MB
[root@ip-172-16-20-112 pythonApp]# docker run -d --name ptrhon_cont pythonimg
d88237bb20f9552f18848eadc0537ae1460333d53a0d5bd11d8b5268b01f2e36
[root@ip-172-16-20-112 pythonApp]# docker exec -it ptrhon_cont /bin/bash
root@d88237bb20f9:/app# curl http:localhost:5000
curl: (3) URL using bad/illegal format or missing URL
root@d88237bb20f9:/app# curl http://localhost:5000
Hello from Dockerized Python app!root@d88237bb20f9:/app#
root@d88237bb20f9:/app#
root@d88237bb20f9:/app#
```

```
sername:
rror: Non-null Username Required
root@ip-172-16-20-112 pythonApp]#
root@ip-172-16-20-112 pythonApp]# docker login -u vishalbobhate
assword:
ARNING! Your password will be stored unencrypted in /root/.docker/config.json.
onfigure a credential helper to remove this warning. See
ttps://docs.docker.com/engine/reference/commandline/login/#credentials-store

ogin Succeeded
root@ip-172-16-20-112 pythonApp]# docker push vishalbobhate/pvt_repo:tagname
he push refers to repository [docker.io/vishalbobhate/pvt_repo]
n image does not exist locally with the tag: vishalbobhate/pvt_repo
root@ip-172-16-20-112 pythonApp]# docker tag pythonimg vishalbobhate/pvt_repo:py_latest
root@ip-172-16-20-112 pythonApp]# docker push vishalbobhate/pvt_repo:py_latest
he push refers to repository [docker.io/vishalbobhate/pvt_repo]
c38ad00f304: Pushed
c5d50f62f4d: Pushed
bb5854a68e4: Pushed
6e921208834: Pushed
90349e3ad0a: Mounted from library/python
d482936d216: Mounted from library/python
85902786fbc: Mounted from library/python
a3d3e410343: Mounted from library/python
bc405ffff1c: Mounted from library/python
878d283e64a: Mounted from library/python
85eb556134e: Mounted from library/python
y_latest: digest: sha256:01c9d7f3875572742ba00d199a3328df29250f778cc7f094b5c32c7d5a3c8dd3 size: 2627
root@ip-172-16-20-112 pythonApp]#
```



- Vishal bobahte