Name: Vinay Dawani
Roll no: 22
Batch: T11

Experiment: 9

Aim: To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with Containers.

Theory: Docker is an open-source platform designed to automate the deployment, scaling, and management of applications using **containerization**. Containers are lightweight, standalone packages that include everything needed to run an application—such as the code, runtime, system tools, libraries, and settings—ensuring consistent performance across various environments.

⚙️ 1. Docker Architecture

Docker follows a client-server architecture consisting of:

- **Docker Engine**: The core of Docker, responsible for building and running containers.
    - **Docker Daemon (dockerd)**: Runs in the background and manages Docker objects like images, containers, volumes, etc.
    - **Docker Client (docker)**: Command-line tool to interact with the daemon.
- **Docker Registries**: Services like **Docker Hub** or private registries that store Docker images.

📦 2. Key Docker Concepts

🧱 Containers:

- Lightweight, portable, and isolated environments.
- Share the host OS kernel, making them faster and more resource-efficient than traditional virtual machines.

🖼️ Images:

- Read-only templates used to create containers.
- Built using a **Dockerfile**, which defines the base OS, required packages, and commands.

📄 Dockerfile:

- A script containing instructions for building a Docker image.
- Includes keywords like FROM, WORKDIR, COPY, and CMD.

🛠️ 3. Docker Commands Overview

Some commonly used Docker CLI commands include:

- docker --version → Check Docker installation.
- docker pull <image> → Download an image from Docker Hub.

- docker build -t <name> . → Build a Docker image from a Dockerfile.
- docker run <image> → Start a container from an image.
- docker ps → List running containers.
- docker stop <container_id> → Stop a container.
- docker rm <container_id> → Remove a container.
- docker rmi <image> → Remove an image.

🔄 4. Container Life Cycle

1. **Create** → Define and configure the container.
2. **Start** → Run the container instance.
3. **Stop** → Halt the running container.
4. **Remove** → Delete the container after use.

🌍 5. Why Use Docker?

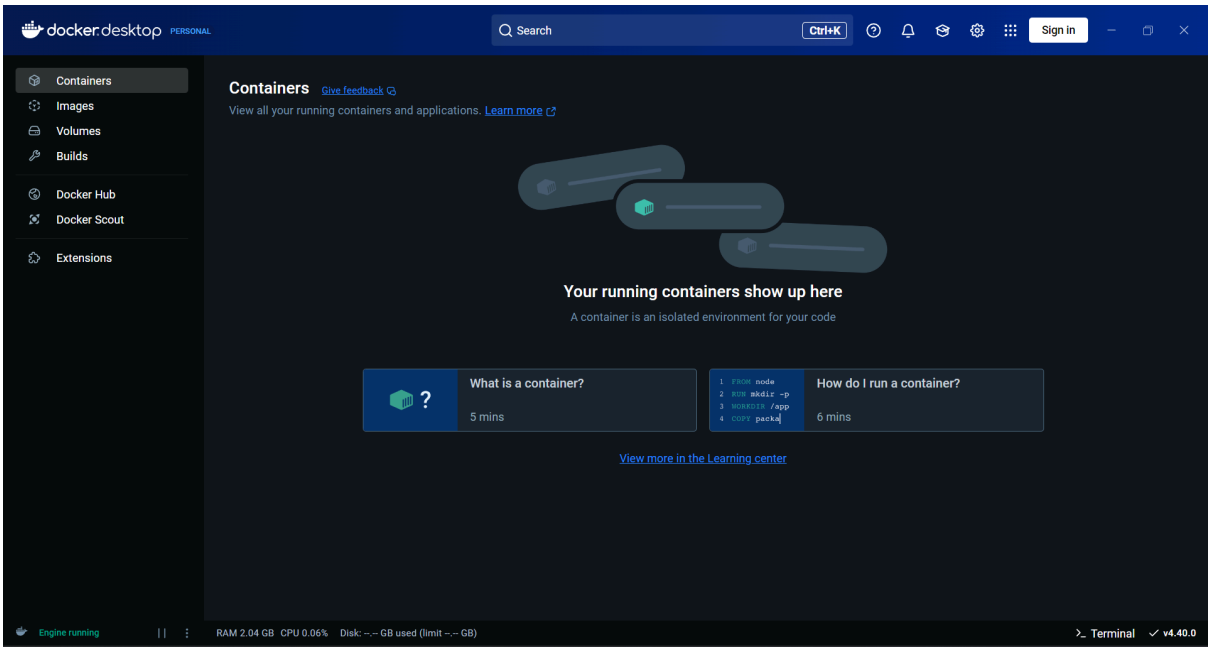
- **Portability**: Works uniformly across dev, test, and production.
- **Isolation**: Each app runs in its own container without interference.

- **Efficiency**: Minimal overhead compared to VMs.
- **Scalability**: Ideal for microservices and large-scale applications.

🧰 6. Docker Compose (Optional Advanced)

Docker Compose allows defining and running multi-container applications using a docker-compose.yml file. It's useful for managing services, volumes, and networks with a single command.

Output:



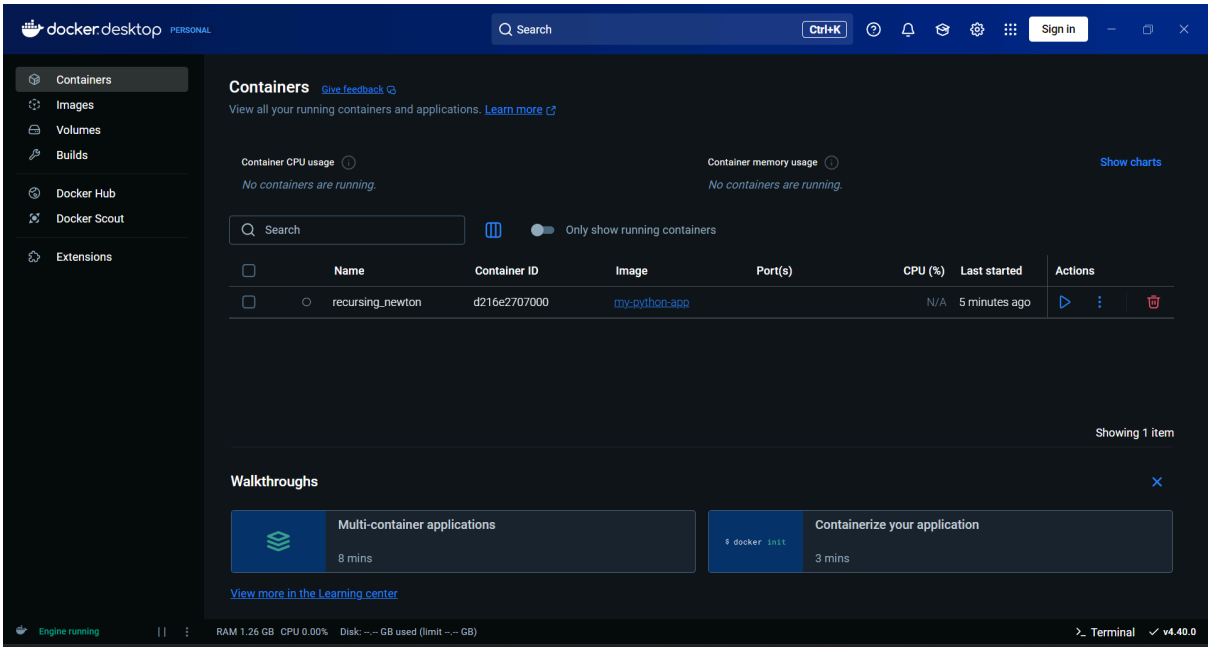
```
Command Prompt
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay>docker pull python
Using default tag: latest
latest: Pulling from library/python
739b86d2a778: Pull complete
b617a119f8a2: Pull complete
e25cca11fd29: Pull complete
1eb98adba0eb: Pull complete
07d1b5af933d: Pull complete
171e1bee1949: Pull complete
23b7d26ef1d2: Pull complete
Digest: sha256:9819e5616923079cc16af4a93d4be92c0c487c6e02fd9027220381f3e125d64a
Status: Downloaded newer image for python:latest
docker.io/library/python:latest

C:\Users\vinay>
```

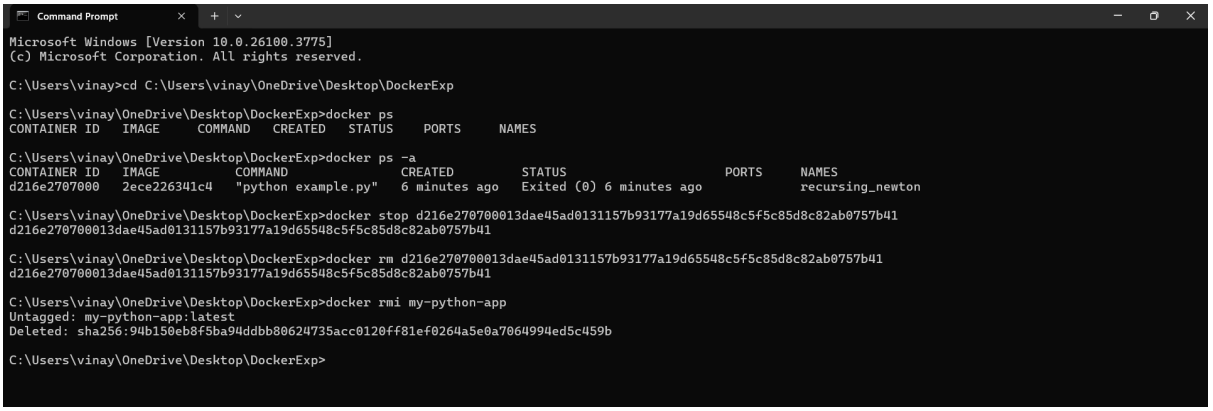
```
C:\Users\vinay\OneDrive\Desktop\DockerExp>docker build -t my-python-app .
[+] Building 11.1s (8/8) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                 0.0s
 => => transferring dockerfile: 230B                                                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:3.9                                                                        3.5s
 => [internal] load .dockerignore                                                                                                    0.0s
 => => transferring context: 2B                                                                                                      0.0s
 => [1/3] FROM docker.io/library/python:3.9@sha256:a847112640804ed2d03bb774d46bb1619bd37862fb2b7e48eebe425a168c153b                  6.6s
 => => resolve docker.io/library/python:3.9@sha256:a847112640804ed2d03bb774d46bb1619bd37862fb2b7e48eebe425a168c153b                  0.1s
 => => sha256:cc381eaa3a998eda575c9d732b81eece7d103a1c7114a4a21bca12086db769f0 249B / 249B                                           0.4s
 => => sha256:b25c426204f45b8436d684dea65578794f4f7b8f1148dd89aaa85ef1393212ea 19.85MB / 19.85MB                                     5.3s
 => => sha256:5893f2eb5418ad35eb72d79548d6a2689599e97b3ea3ab0efa6babc3c261d537 6.16MB / 6.16MB                                       2.3s
 => => extracting sha256:5893f2eb5418ad35eb72d79548d6a2689599e97b3ea3ab0efa6babc3c261d537                                             0.4s
 => => extracting sha256:b25c426204f45b8436d684dea65578794f4f7b8f1148dd89aaa85ef1393212ea                                             1.0s
 => => extracting sha256:cc381eaa3a998eda575c9d732b81eece7d103a1c7114a4a21bca12086db769f0                                             0.0s
 => [internal] load build context                                                                                                    0.1s
 => => transferring context: 308B                                                                                                    0.0s
 => [2/3] WORKDIR /app                                                                                                               0.2s
 => [3/3] COPY . /app                                                                                                                0.1s
 => exporting to image                                                                                                               0.4s
 => => exporting layers                                                                                                              0.2s
 => => exporting manifest sha256:91d06685d6b4bec8eddee8fa7e5f504c42abde864ed343a2a0d10d3283c84d56                                    0.0s
 => => exporting config sha256:6e4d802c9d1f0957a6af879a878e19cc5d97e1355e282c6d68ee4d43b40ca8d7                                      0.0s
 => => exporting attestation manifest sha256:e423884dec208d8a361b91672edae161d81a6c93af9486daeaa9c5d2dbe60d4e                        0.0s
 => => exporting manifest list sha256:2ece226341c4aa793a45fa3eb296037716b58a4b7395b571f3c7197b126d211f                               0.0s
 => => naming to docker.io/library/my-python-app:latest                                                                              0.0s
 => => unpacking to docker.io/library/my-python-app:latest                                                                           0.1s

C:\Users\vinay\OneDrive\Desktop\DockerExp>
```




```
C:\Users\vinay\OneDrive\Desktop\DockerExp>docker run my-python-app
Hello from Docker container!

C:\Users\vinay\OneDrive\Desktop\DockerExp>
```



Conclusion:

In this experiment, we understood the fundamental concepts of Docker, including its architecture, container lifecycle, and key commands. We successfully installed Docker, created a Dockerfile, built a custom image, and ran a container. This demonstrated how Docker simplifies application deployment by providing lightweight, portable, and consistent environments.