

Terna Engineering College

Computer Engineering Department
Program: Sem VI

Course: Cloud Computing Lab (CSL605)

PART A
(PART A: TO BE REFERRED BY STUDENTS)

Experiment No.9

Title: To study and Implement Containerization using Docker

A.1 Objective: To know the basic differences between Virtual machine and Container. It involves demonstration of creating, finding, and building, installing, and running Linux/Windows application containers inside local machine or cloud platform.

A.2 Prerequisite:

Knowledge of Networking, Distributed Computing and knowledge of Software architectures.

A.3 Objective:

Objectives this experiment is to provide students an overview AWS, its Features and Services.

A.4 Outcome: (LO3)

After successful completion of this experiment student will be able to run Linux/ windows application containers inside local machine.

A.5 Theory :

Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be developed or tested or in production.



Containerization:

Containerization is OS-based virtualization that creates multiple virtual units in the userspace, known as Containers. Containers share the same host kernel but are isolated from each other through private namespaces and resource control mechanisms at the OS level. Container-based Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. Hypervisors use a lot of hardware which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating system (e.g -Linux, Windows) runs on top of this virtualized hardware in each virtual machine instance. But in contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine and one or more processes can be run within each container. In containers you don't have to pre-allocate any RAM, it is allocated dynamically during the creation of containers while in VMs you need to first pre-allocate the memory and then create the virtual machine. Containerization has better resource utilization compared to VMs and a short boot-up process. It is the next evolution in virtualization.

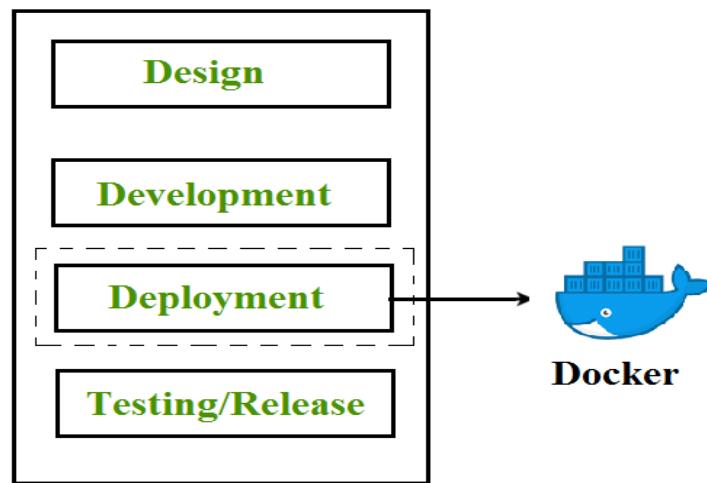
Containers can run virtually anywhere, greatly easy development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal, on a developer's machine or in data centers on-premises; and of course, in the public cloud. Containers virtualize CPU, memory, storage, and network resources at the OS level, providing developers with a sandboxed view of the OS logically isolated from other applications. Docker is the most popular open-source container format available and is supported on Google Cloud Platform and by Google Kubernetes Engine.

After you understand the concept of Docker, now let's get into the implementation. There are several steps that we will do:

1. Install Docker
2. Create a file called Dockerfile

3. Build the image

4. Run the image



Useful link:

<https://docs.docker.com/>

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the ERP or emailed to the concerned lab in charge faculties at the end of the practical in case there is no ERP access available)

Roll No. B48	Name: Aryan Unhale
Class : TE B Comps	Batch : B3
Date of Experiment: 3/4/25	Date of Submission: 5/4/25
Grade :	

Step 1: Install Docker on Windows

1. Go to the official Docker download page:
 <https://docs.docker.com/desktop/install/windows-install>
2. **Download Docker Desktop for Windows.**
3. **Run the installer** and follow the on-screen instructions:
 - o Ensure **WSL 2** is enabled (Docker will prompt you to install it if needed).
 - o You may need to restart your system.
4. After installation, **launch Docker Desktop**. You should see the whale icon in the system tray.

The screenshot shows a web browser window for the Docker documentation site (docs.docker.com/desktop/setup/install/windows-install/). The main content is titled "Install Docker Desktop on Windows". It includes a sidebar with navigation links for Docker Build, Docker Compose, Testcontainers, Docker Desktop (Setup, Install, Mac, Windows), Enterprise deployment, Linux, VM or VDI environments, Sign in, Allowlist, and Explore Docker Desktop. The "Windows" link under Docker Desktop is highlighted.

The main area displays the "Install Docker Desktop on Windows" page with sections for "Docker Desktop terms", "System requirements", and "Configuration". Under "Configuration", there are two checked checkboxes: "Use WSL 2 instead of Hyper-V (recommended)" and "Add shortcut to desktop".

A sidebar on the right shows "Recent download history" with entries for "Docker Desktop Installer.exe" (503 MB, Done) and two PDF files named "Invoice_2105207441 (1).pdf" and "Invoice_2105207441.pdf" (both 93.1 KB, 19 hours ago).

At the bottom of the browser window, there is a cookie consent banner with buttons for "Cookies Settings", "Reject All", "Accept All", and "Give feedback".

Docker Desktop 4.39.0

Installation succeeded

You must restart Windows to complete installation.

[Close and restart](#)



Finish setting up Docker Desktop
version 4.39.0 (184744)

Complete the installation of Docker Desktop.

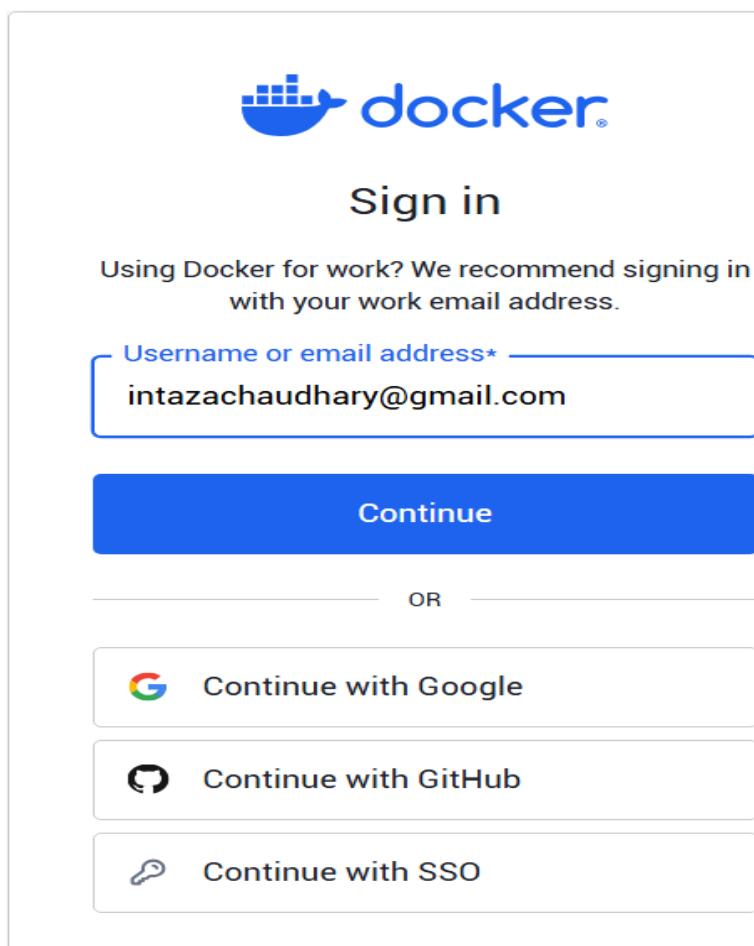
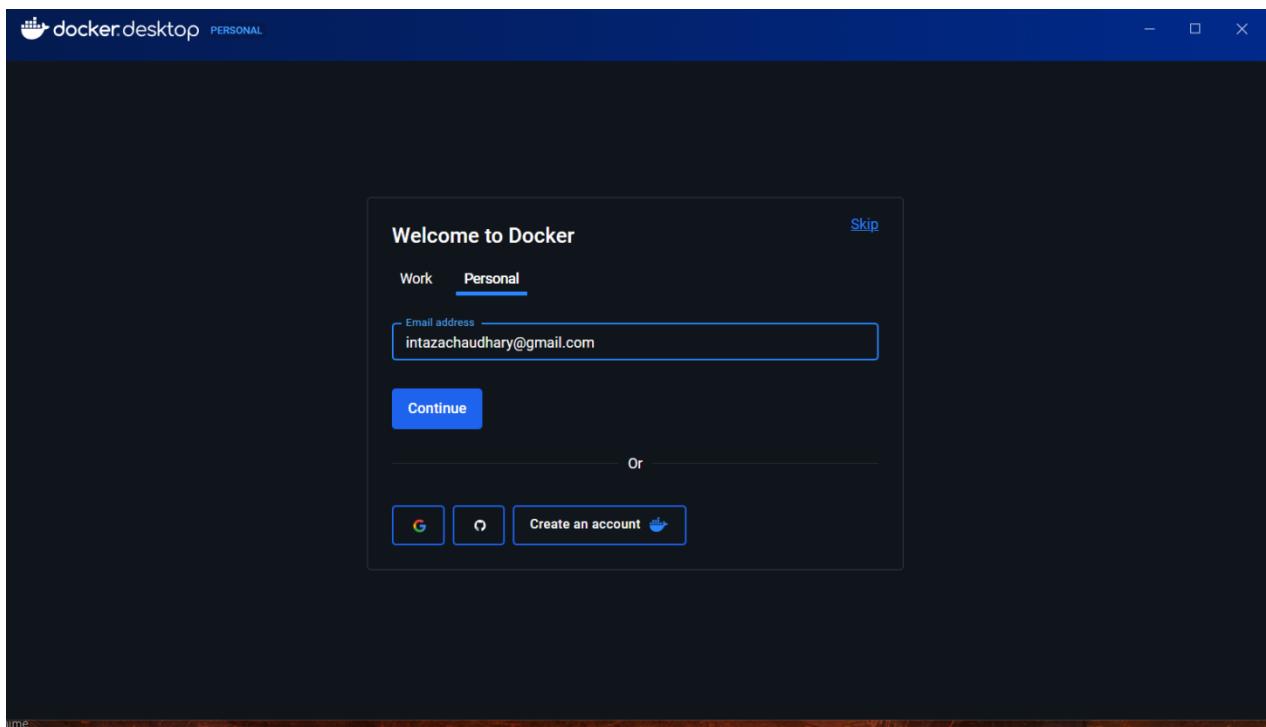
- Use recommended settings (requires administrator password)

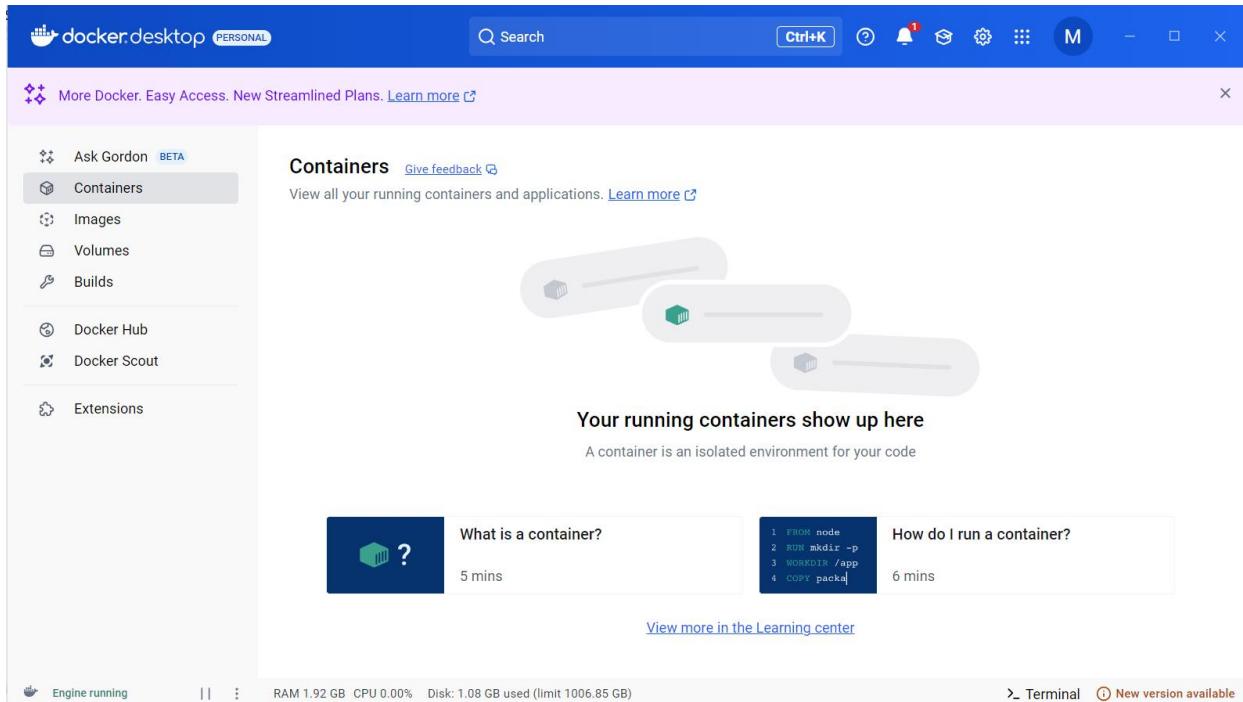
Docker Desktop automatically sets the necessary configurations that work for most developers.

- Use advanced settings

You manually set your preferred configurations.

[Finish](#)





Step 2: To verify installation, open **Command Prompt (CMD)** or **PowerShell** and type:

```
docker --version
```

Create a new folder for your project:

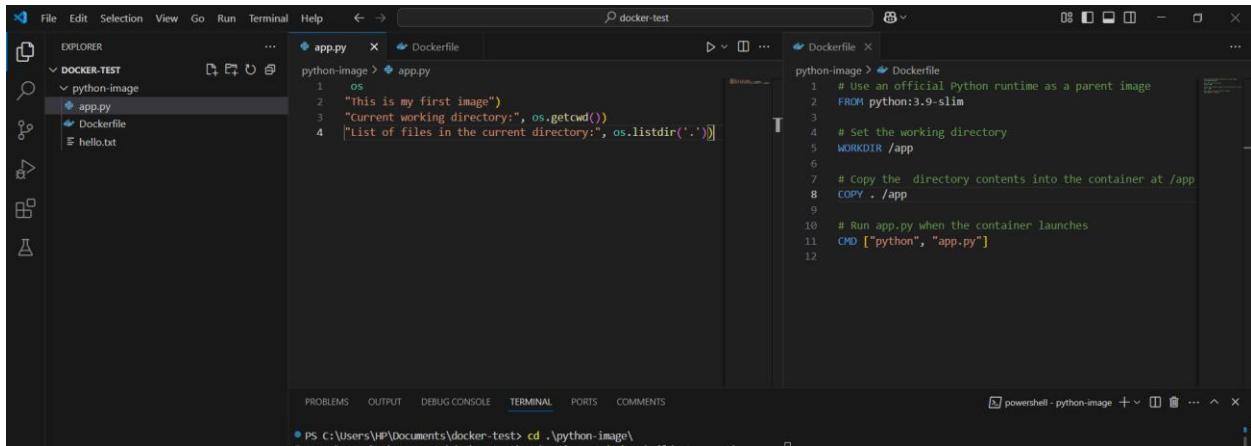
```
mkdir docker-test  
cd docker-test
```

A screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The window displays the following text:

```
Microsoft Windows [Version 10.0.19045.5608]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\HP\Documents>docker --version  
Docker version 28.0.1, build 068a01e  
  
C:\Users\HP\Documents>mkdir docker-test  
  
C:\Users\HP\Documents>cd docker-test  
  
C:\Users\HP\Documents\docker-test>code .  
  
C:\Users\HP\Documents\docker-test>
```

The window has standard Windows window controls at the top right.

Step 3: Open any code editor (e.g., VS Code) and create a file named **Dockerfile** (no extension).



Step 4: Add the following sample content

App.py

```
import os
print("This is my first image")
print("Current working directory:", os.getcwd())
print("List of files in the current directory:", os.listdir('.'))
```

Dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the directory contents into the container at /app
COPY . /app

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Hello.txt

hi

Step 5;Build the Docker Image

The screenshot shows a VS Code interface with multiple tabs open. The left sidebar has 'EXPLORER' and 'DOCKER-TEST' sections. The main area has tabs for 'app.py', 'Dockerfile', 'hello.txt', and 'powershell - python-image'. The 'TERMINAL' tab displays a build log for a Dockerfile:

```
PS C:\Users\VIP\Documents\docker-test> cd .\python-image  
PS C:\Users\VIP\Documents\docker-test> python -m venv .<br/>python -m docker build -t my-python-app-exp-9 .  
[+] Building 16.9s (9/9) FINISHED  
=> [internal] load build context from Dockerfile  
=> transferring Dockerfile: 30B  
=> [internal] load metadata for docker.io/library/python:3.9-slim  
=> [auth] library/python: pull token for registry-1.docker.io  
=> [internal] load .dockerignore  
=> transferring context: 2B  
=> [1/1] FROM docker.io/library/python:3.9-slim@sha256:e52ca5f579cc58fed41efcb95aaed5dcfc7a156cba76acf4b42Fc19dd00  
=> resolve Dockerfile: 30B  
=> sha256:9a95d174747c9e4a85d4bf5822065468265f5725d8ad4fb30f1e69216f2498 / 2498  
=> sha256:dabce0584f421af08256cc9d4d6161312687007642c23da4259346 14.93MB / 14.93MB  
=> sha256:cce49b84d9d75777ee48546b2d57231a9e7c6746f4aa93175231a21 3.51MB / 3.51MB  
=> sha256:ee09acdb7905a1989d9fc795fda5a26d664bb27b5c688c2b742b25fad 28.20MB / 28.20MB  
=> extracting sha256:ee09acdb7905a1989d9fc795fda5a26d664bb27b5c688c2b742b25fad  
=> extracting sha256:01bed0584f421af08256cc9d4d6161312687007642c23da4259346  
=> extracting sha256:9a95d174747c9e4a85d4bf582205468265f57a58daef4fb3071e60216f  
=> [internal] load build context  
=> transferring context: 539B  
=> [2/3] WORKDIR . /app  
=> [3/3] COPY . /app  
=> exporting to image  
=> exporting layers  
=> exporting manifest sha256:516e42ab55e8856faa42a0eb1766880c79aa8cb6b00c0ed1b1f51b6955582  
=> exporting config sha256:8148a5d73c73115de655b46bda6499859a1e482412e5c10114ab55bc2271383f  
=> exporting attestation manifest sha256:56ce657c91d674c0785e11217b6f725bf2ebd4a25e6fb1a28fa70bbcadc0e0  
=> exporting manifest list sha256:4f559faae475a8fa2354d04d9493e569c1948ae447fa10372ff01ee5cae2  
=> naming to docker.io/library/my-python-app-exp-9:latest  
=> unpacking to docker.io/library/my-python-app-exp-9:latest
```

The terminal also shows a command to switch to the 'powershell - python-image' tab.

Step 6: Run the Docker Image

The screenshot shows the Visual Studio Code interface with a Docker project open. The Explorer sidebar on the left lists files: Dockerfile, app.py, and hello.txt. The current file is Dockerfile, which contains Python code to build a Docker image. The terminal tab at the bottom shows the command `docker run` being executed, resulting in the output: "This is my first image".

```
python-image > Dockerfile
python-image > app.py
1  os
2  "This is my first image"
3  "Current working directory:", os.getcwd()
4  "List of files in the current directory:", os.listdir('')\n
python-image > Dockerfile
1  # Use an official Python runtime as a parent image
2  FROM python:3.9-slim
3
4  # Set the working directory
powershell... powershell
```

```
PS C:\Users\VHP\Documents\docker-test> docker run --name firstpycontainer my-python-app-exp-9
This is my first image
Current working directory: /app
List of files in the current directory: ['hello.txt', 'Dockerfile', 'app.py']
PS C:\Users\VHP\Documents\docker-test>
```

Containers Give feedback

No containers are running.

Name	Container ID	Image	Port(s)	CPU (%)	Last start	Actions
firstpycontainer	43548822594a	my-python-		N/A	1 second	▶ ⋮ trash

Showing 1 item

Step 7: Bonus Commands

- **List all images:** docker images
- **List running containers:** docker ps
- **List all containers (including stopped):** docker ps -a
- **Stop a running container (use container ID):** docker stop <container-id>

```

PS C:\Users\HP\Documents\docker-test> docker run --name firstpycontainer my-python-app-exp-9
This is my first image
Current working directory: /app
List of files in the current directory: ['hello.txt', 'Dockerfile', 'app.py']

PS C:\Users\HP\Documents\docker-test> docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
my-python-app-exp-9  latest   4f555faae475  3 minutes ago  189MB

PS C:\Users\HP\Documents\docker-test> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED          STATUS          PORTS     NAMES
43548822594a        "python app.py"    "python app.py"    About a minute ago   Exited (0) 53 seconds ago   firstpycontainer

A "List of files in the current directory..."  os.listdir('\'')
A "# Set the working directory"  os.chdir('\'')

PS C:\Users\HP\Documents\docker-test> docker rm firstpycontainer
firstpycontainer
PS C:\Users\HP\Documents\docker-test>

```

The screenshot shows the Docker Desktop application window. The left sidebar has 'Images' selected. The main area displays a table of local images:

Name	Tag	Image ID	Created	Size	Actions
my-python-app-exp-9	latest	4f555faae475	8 minutes ago	189.19 MB	Delete View Edit Copy

Below the table, it says "Selected 1 of 1".

The screenshot shows the Docker Desktop application window. The left sidebar has 'Builds' selected. The main area displays a table of build history:

Name	ID	Builder	Duration	Created	Author
python-image	mznys8	default	16.7s	9 minutes ago	N/A

At the bottom, there is a message about Docker Build Cloud.

```
PS C:\Users\HP\Documents\docker-test> docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE

```

The terminal window shows the command `docker images` being run, and the resulting table of images.

B.1 Question of Curiosity:

Q.1 What is containerization?

Ans: Containerization is a method of packaging, distributing, and running applications within isolated environments called containers. These containers encapsulate everything needed to run an application, including code, runtime, libraries, and dependencies. Unlike traditional virtual machines, containers share the host operating system's kernel, making them lightweight and more efficient. Containerization facilitates portability, scalability, and consistency across different computing environments, enabling developers to build, deploy, and manage applications more efficiently across various platforms and infrastructure.

Q.2 Difference between virtualization and containerization.

Aspect	Virtualization (VM)	Containerization (Docker)
Isolation	Full OS-level isolation with its own kernel	Process-level isolation using the host OS kernel
Resource Usage	Heavy; each VM runs its own full OS	Lightweight; containers share the host OS
Startup Time	Slow (in minutes)	Fast (in seconds)
Performance	Slower due to full OS overhead	Faster due to minimal abstraction
Size	Large (GBs, due to OS image)	Small (MBs, since only app and dependencies are included)
Portability	Less portable; depends on hypervisor and OS	Highly portable; runs the same way on any OS with Docker installed
Boot Process	Full OS boot required	Just starts the app process
Isolation Scope	Hardware-level using hypervisor	OS-level using namespaces and cgroups
Use Cases	Best for running multiple different OSes	Best for microservices, lightweight app deployment

Aspect	Virtualization (VM)	Containerization (Docker)
Example Tools	VMware, VirtualBox, Hyper-V	Docker, Podman, containerd

Q.3 What is docker image?

Ans: A Docker image is a lightweight, standalone, and executable package that contains all the necessary components to run a software application, including code, libraries, dependencies, and runtime environment. It serves as a blueprint for creating Docker containers. Docker images are built from a Docker file, which specifies the configuration and instructions for assembling the image layer by layer. These images can be easily shared, distributed, and deployed across different environments, ensuring consistency and reproducibility in application deployment processes.

Q.4 What is docker container?

Ans: A Docker container is a runtime instance of a Docker image. It is a lightweight, standalone, and executable environment that encapsulates an application along with its dependencies. Containers run isolated from each other and from the underlying host system, leveraging kernel-level features like namespaces and control groups for resource isolation and management. Docker containers provide consistency in application deployment across different environments, enabling developers to package, deploy, and manage applications efficiently and reliably, irrespective of the underlying infrastructure.

B.3 Conclusion:

After performing this experiment, I learned the core concepts of containerization using Docker. I understood the difference between virtualization and containerization and successfully created and ran a Docker container on my local machine. This will help me deploy apps in a more efficient and portable way.