

## Assignment 07

**AIM:** To understand Docker architecture and container life cycle, install docker, deploy container in docker.

**LO1:** To understand the fundamentals of DevOps engineering and be fully proficient with DevOps terminologies, concepts, benefits, and deployment options to meet your business requirements.

**LO5:** To understand the concept of containerization and analyze the containerization of OS images and deployment of applications over Docker.

### **THEORY:**

**Docker** is a popular platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient environments that contain all the necessary components to run an application, including the code, runtime, libraries, and dependencies. Docker provides a way to package applications and their dependencies into a standardized format, making it easier to deploy and manage software across different environments, such as development, testing, and production.

Here are some key components and concepts related to Docker:

**Docker Containers:** Containers are instances of Docker images. An image is a lightweight, standalone, and executable package that includes everything needed to run an application, including the code, runtime, system tools, libraries, and settings. Containers are isolated from one another and the host system, ensuring consistency and reproducibility of applications across different environments.

**Docker Engine:** The Docker Engine is the core component of Docker that manages containers. It includes a server, REST API, and command-line interface (CLI) for interacting with Docker. It can run on various operating systems, including Linux and Windows.

**Dockerfile:** A Dockerfile is a text file that contains a set of instructions for building a Docker image. It specifies the base image, adds application code, sets environment variables, and configures the container. Dockerfiles are used to create customized images for specific applications.

**Docker Compose:** Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to define the services, networks, and volumes in a single YAML file, making it easier to manage complex applications with multiple interconnected containers.

**Docker Hub:** Docker Hub is a cloud-based registry service provided by Docker, Inc. It allows users to share and discover Docker images. You can find a wide range of pre-built Docker images on Docker Hub, which can be used as a base for your own containers.

**Orchestration:** Docker Swarm and Kubernetes are popular tools for orchestrating containers in production environments. They provide features like automatic scaling, load balancing, service discovery, and high availability for containerized applications.

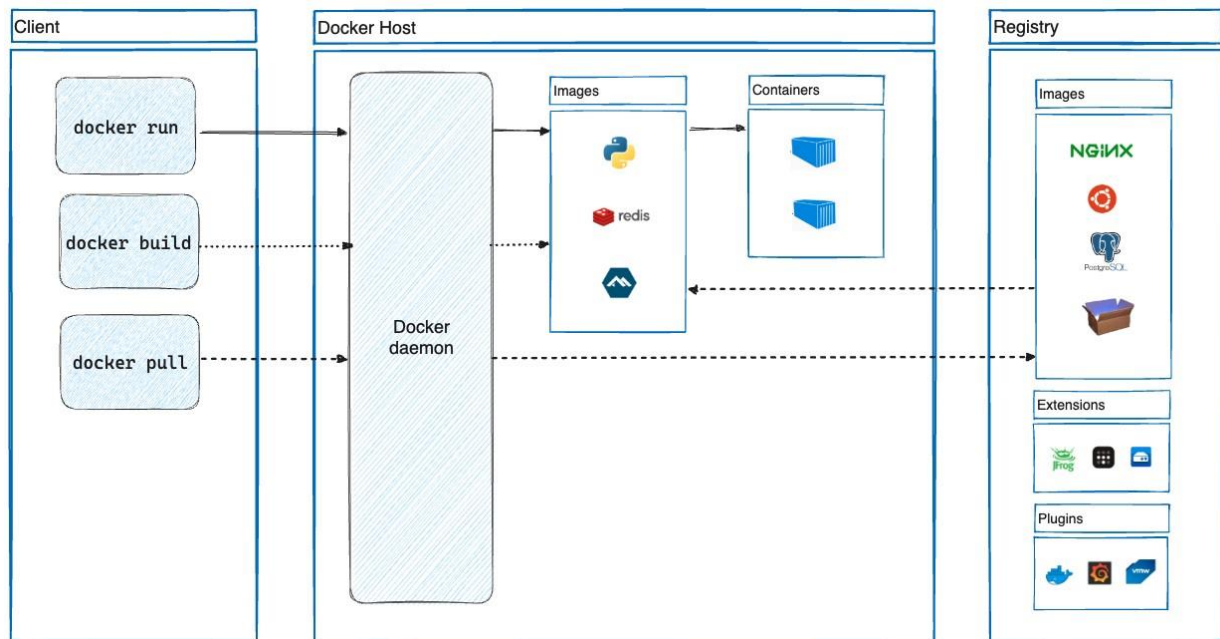
**Portability:** Docker containers are highly portable, meaning you can run the same containerized application on different platforms and cloud providers without modification. This portability simplifies development, testing, and deployment workflows.

## DOCKER ARCHITECTURE

Docker uses a client-server architecture that consists of several key components. Understanding the Docker architecture is essential for grasping how containers work and interact with each other and the host system. Here's an overview of the main components:

1. **Docker Daemon (dockerd):** The Docker daemon is a background service that runs on the host system. It is responsible for managing Docker containers, images, networks, and volumes. Docker client tools communicate with the daemon to execute commands and manage containers.
2. **Docker Client:** The Docker client is a command-line tool or graphical user interface (GUI) that allows users to interact with the Docker daemon. Users issue commands to the client, which then sends requests to the daemon for container management.
3. **Docker Registry:** Docker images are stored in registries, which are repositories for container images. Docker Hub is a popular public registry, but you can also set up private registries for your organization's custom images. When you run a Docker container, it typically pulls the required image from a registry if it's not already available on the host system.
4. **Docker Images:** Docker images are read-only templates that contain an application's code, runtime, libraries, and dependencies. Images are used as a basis for creating containers. Images can be built from Dockerfiles, which are plain text files containing instructions for building an image layer by layer.
5. **Docker Containers:** Containers are instances of Docker images that run in isolated environments on the host system. Each container has its own filesystem, processes, networking, and resource constraints. Containers are lightweight, fast to start, and can be easily moved between different Docker hosts.

NAME – AMAN SINGH  
BATCH – T23  
ROLL NO - 128



## LIFECYCLE OF A CONTAINER

**Creating a Container:** To create a Docker container, you typically start with a Docker image. You can use a pre-built image from a registry or build your own using a Dockerfile. The Docker image is essentially a snapshot of a container's filesystem and configuration.

**Running a Container:** To run a container, you use the `docker run` command, specifying the image and any additional options or configurations.

**When you run a container, Docker:** Allocates resources (CPU, memory, etc.) as per your specifications. Creates a unique instance of the container from the image. Starts the container's processes.

**Container Execution:** The container executes the processes defined in its image. These processes can be your application, services, or any command specified in the Dockerfile or provided when starting the container.

**Interacting with a Container:** You can interact with a running container by attaching to its console using the `docker exec` or `docker attach` command. This allows you to run commands inside the container or access its logs. Containers can also expose network ports, allowing communication with other containers or external systems.

**Stopping and Restarting a Container:** You can stop a running container using the `docker stop` command. This sends a termination signal to the container, allowing it to perform

NAME – AMAN SINGH  
BATCH – T23  
ROLL NO - 128

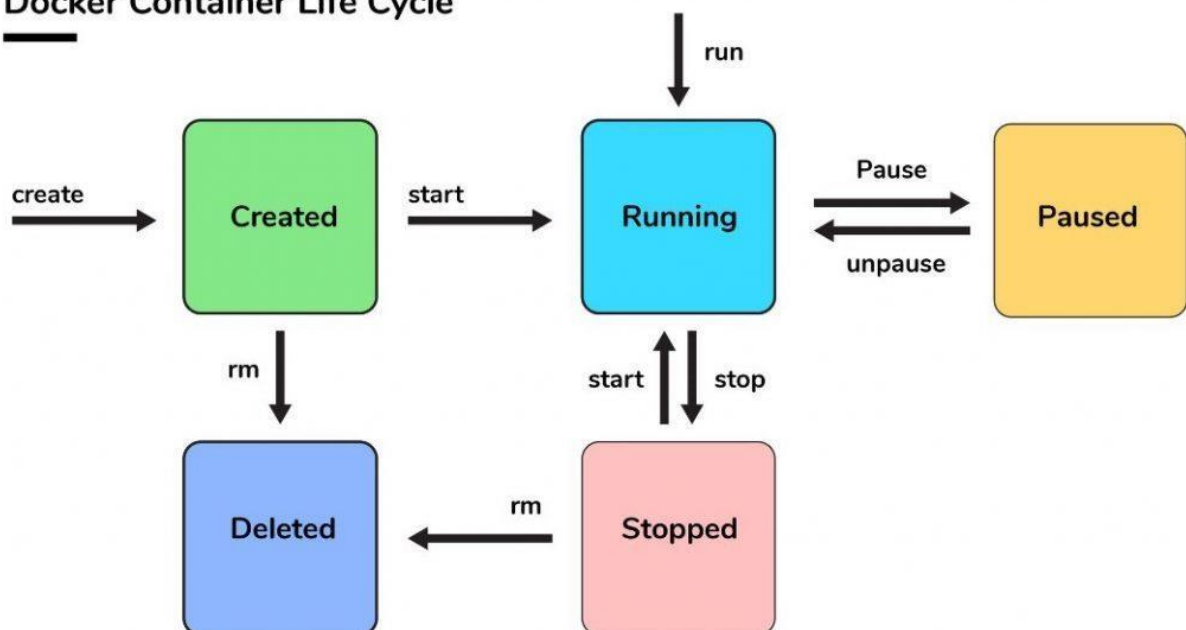
cleanup tasks before stopping. You can restart a stopped container using the docker start command.

**Container Cleanup:** When a container is stopped and removed using the docker rm command, it is deleted along with its filesystem and any changes made during its execution. However, the Docker image remains intact and can be used to create new containers.

**Monitoring and Logs:** Docker provides tools for monitoring container resource usage and collecting container logs, allowing you to troubleshoot and manage running containers effectively.

**Updating and Versioning:** If your application code or dependencies change, you can update the Docker image and recreate containers with the latest changes. This ensures consistency in different environments.

### Docker Container Life Cycle

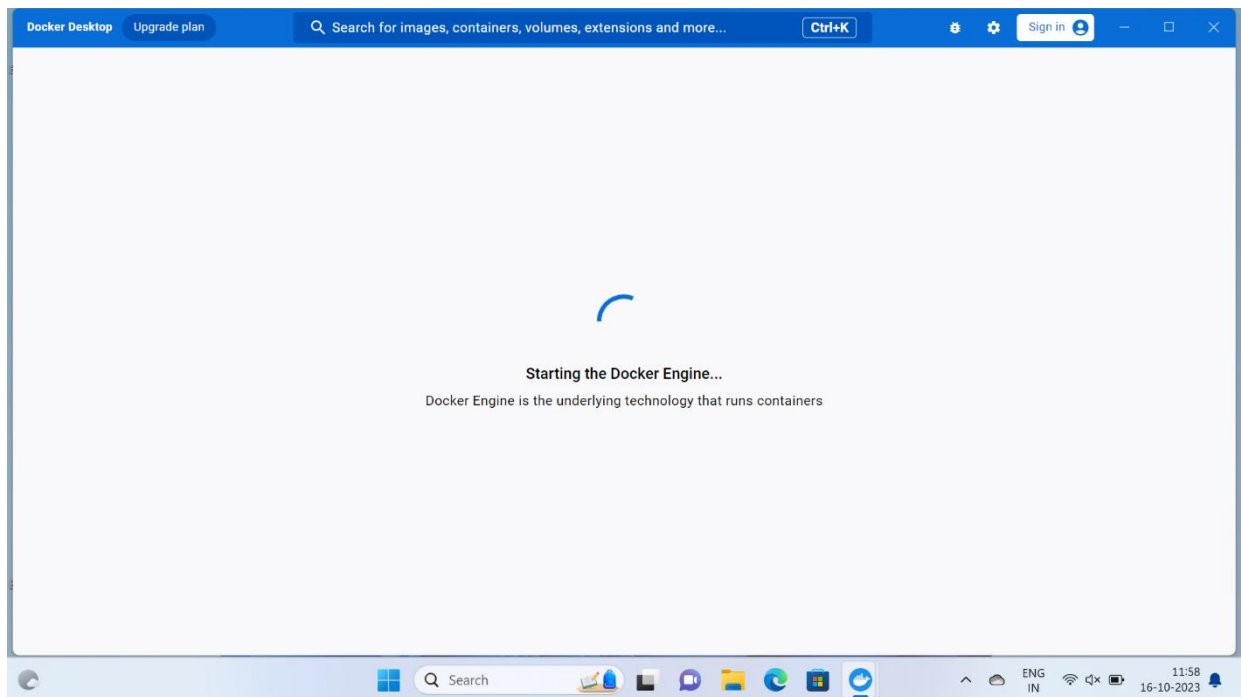
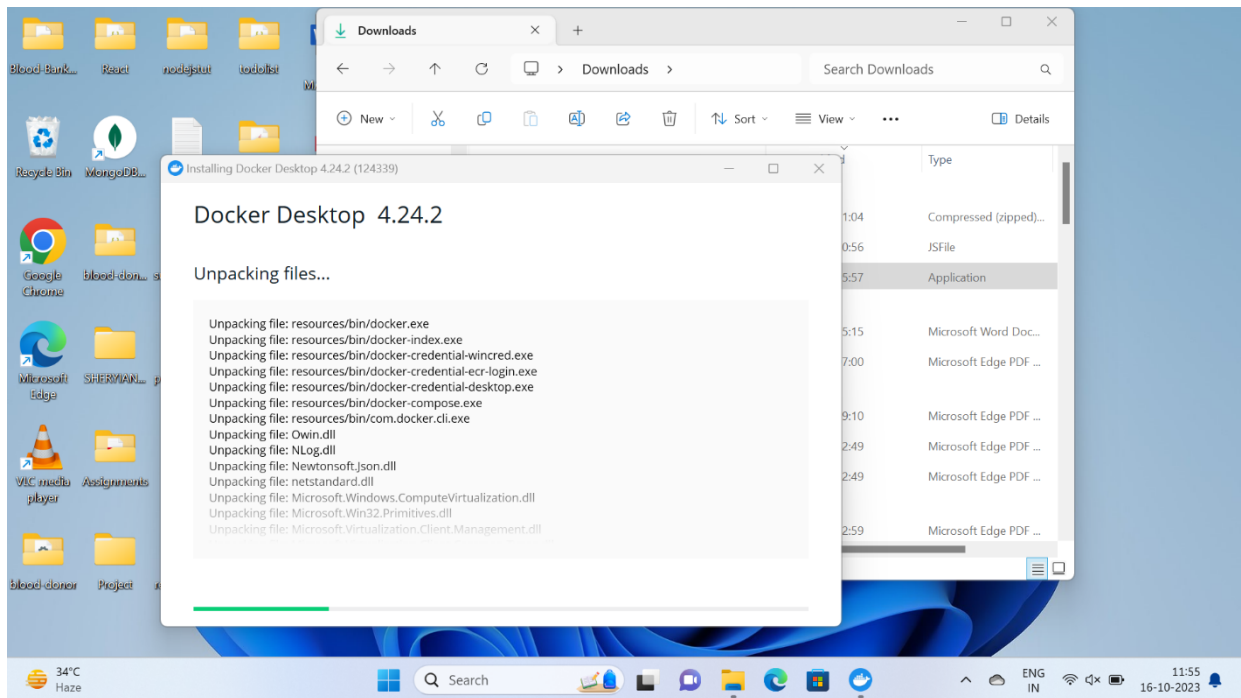


### INSTALLATION:

NAME – AMAN SINGH

BATCH – T23

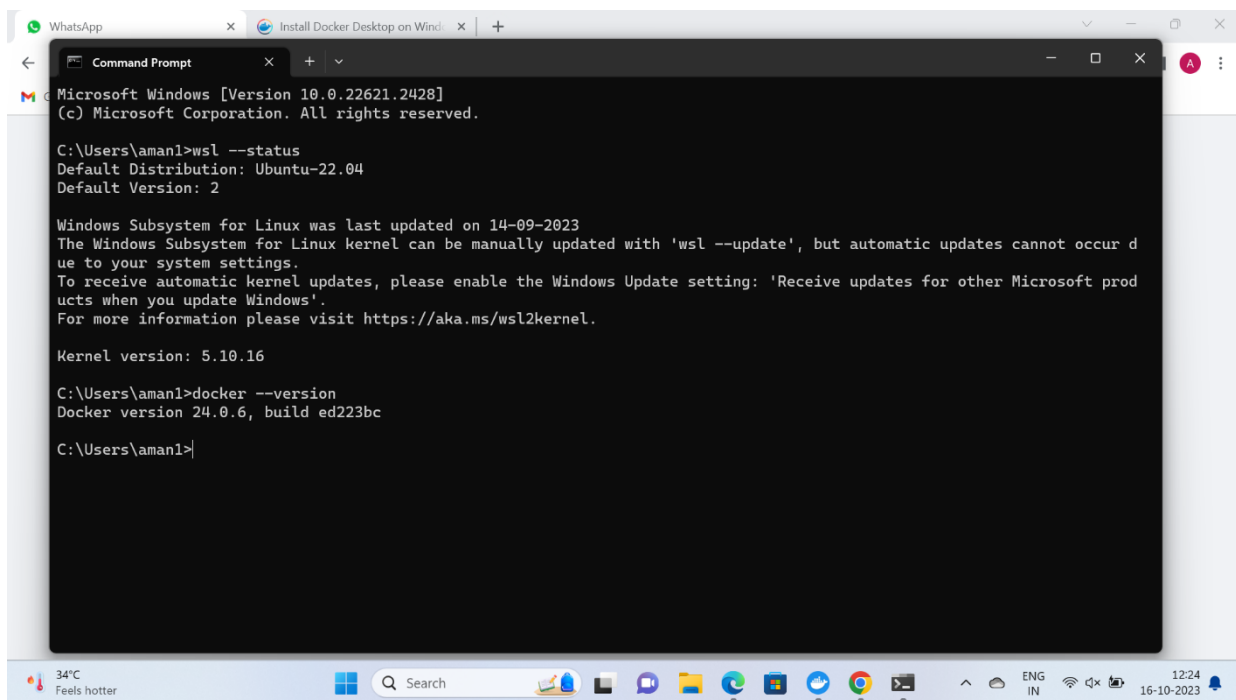
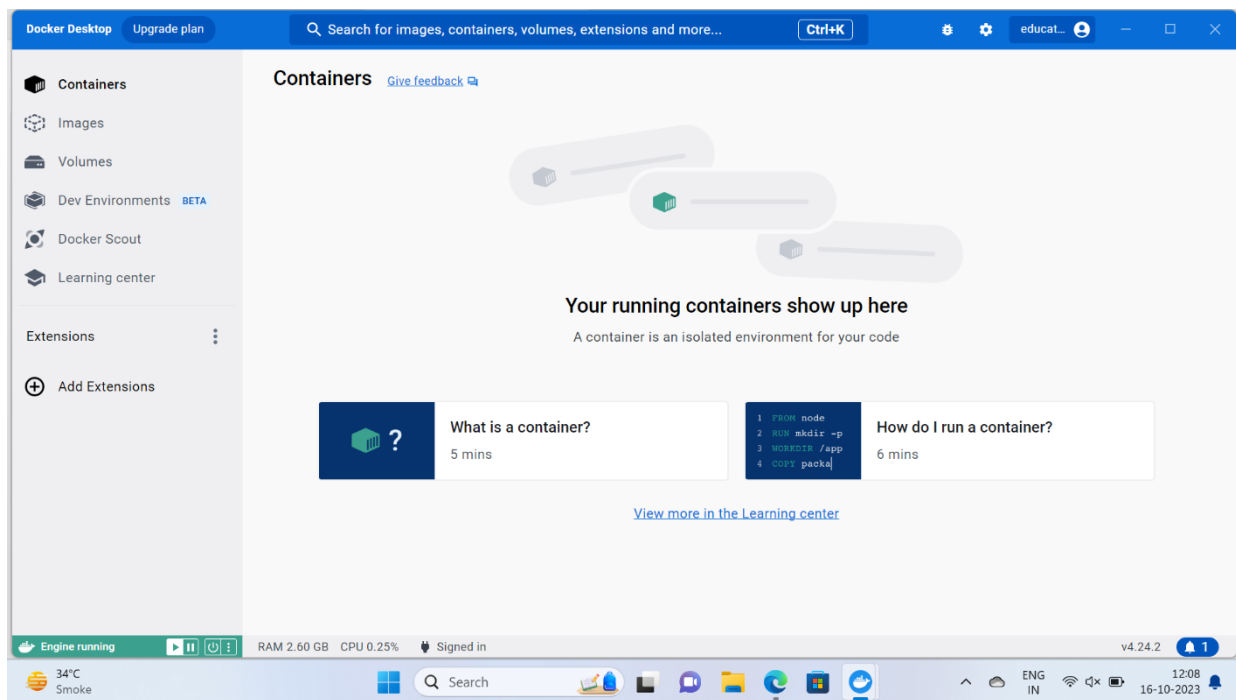
ROLL NO - 128



NAME – AMAN SINGH

BATCH – T23

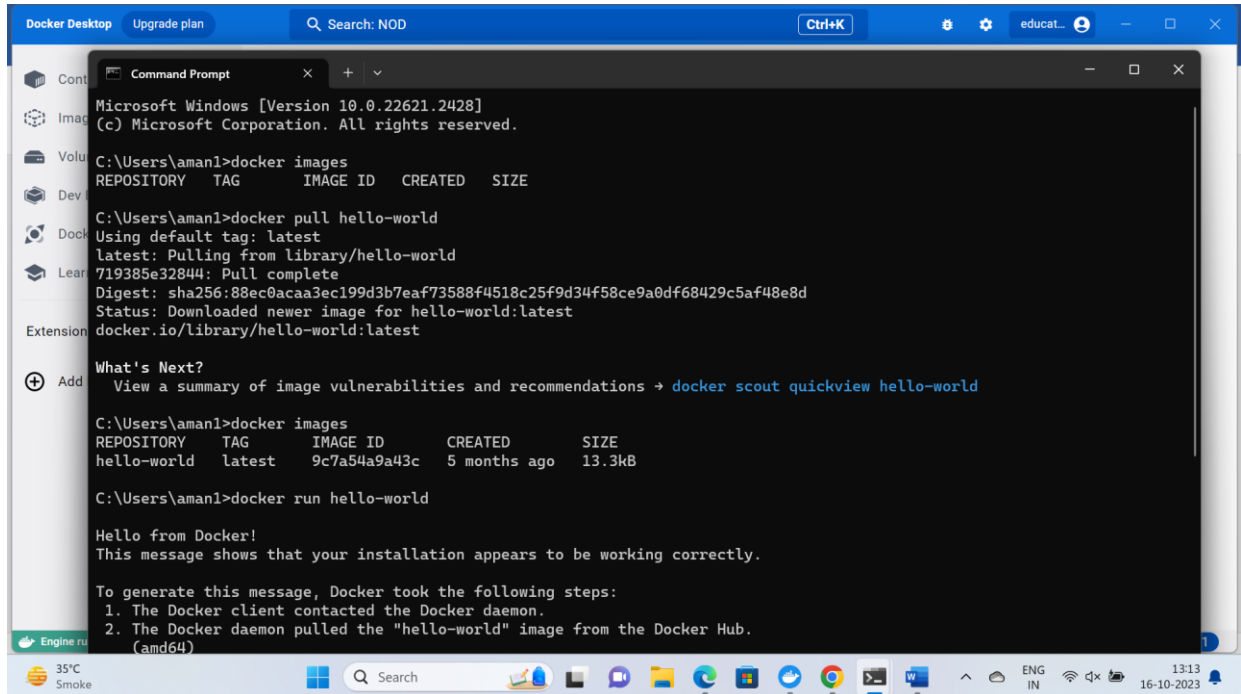
ROLL NO - 128



NAME – AMAN SINGH

BATCH – T23

ROLL NO - 128



```
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\Users\aman1>docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
C:\Users\aman1>docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:88ec0acaa3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest

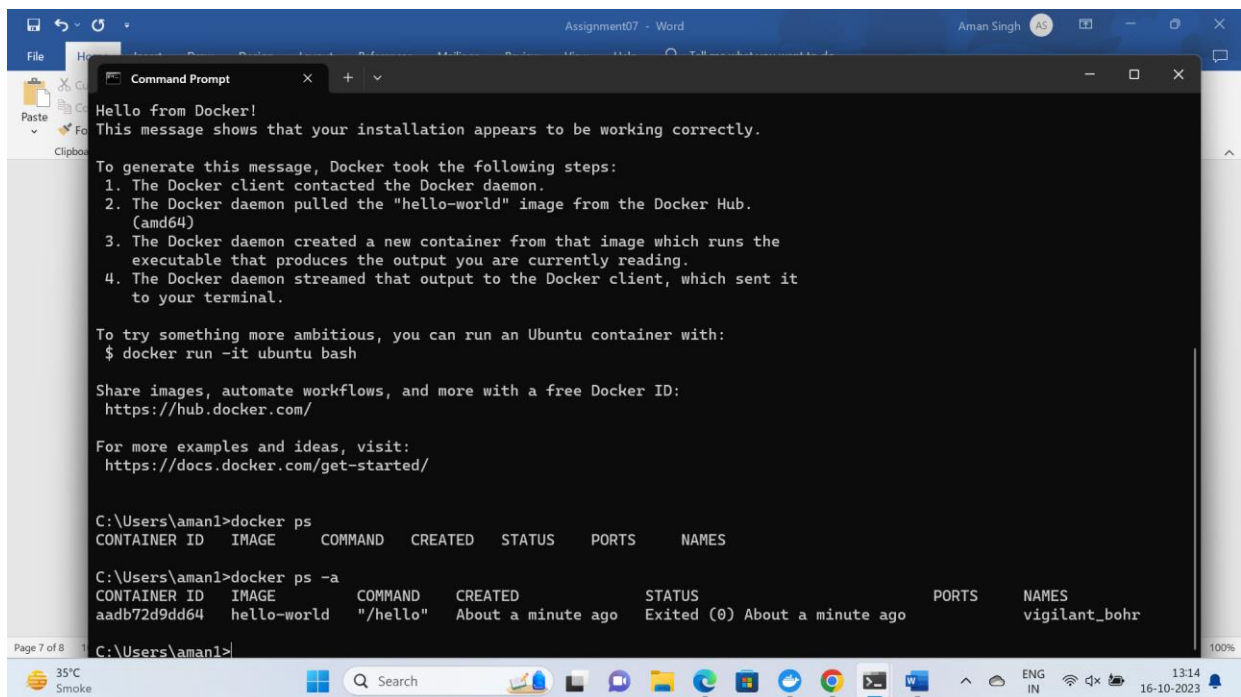
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview hello-world

C:\Users\aman1>docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hello-world latest 9c7a54a9a43c 5 months ago 13.3kB

C:\Users\aman1>docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
```



```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

C:\Users\aman1>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
C:\Users\aman1>docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
aadb72d9dd64 hello-world "/hello" About a minute ago Exited (0) About a minute ago vigilant_bohr

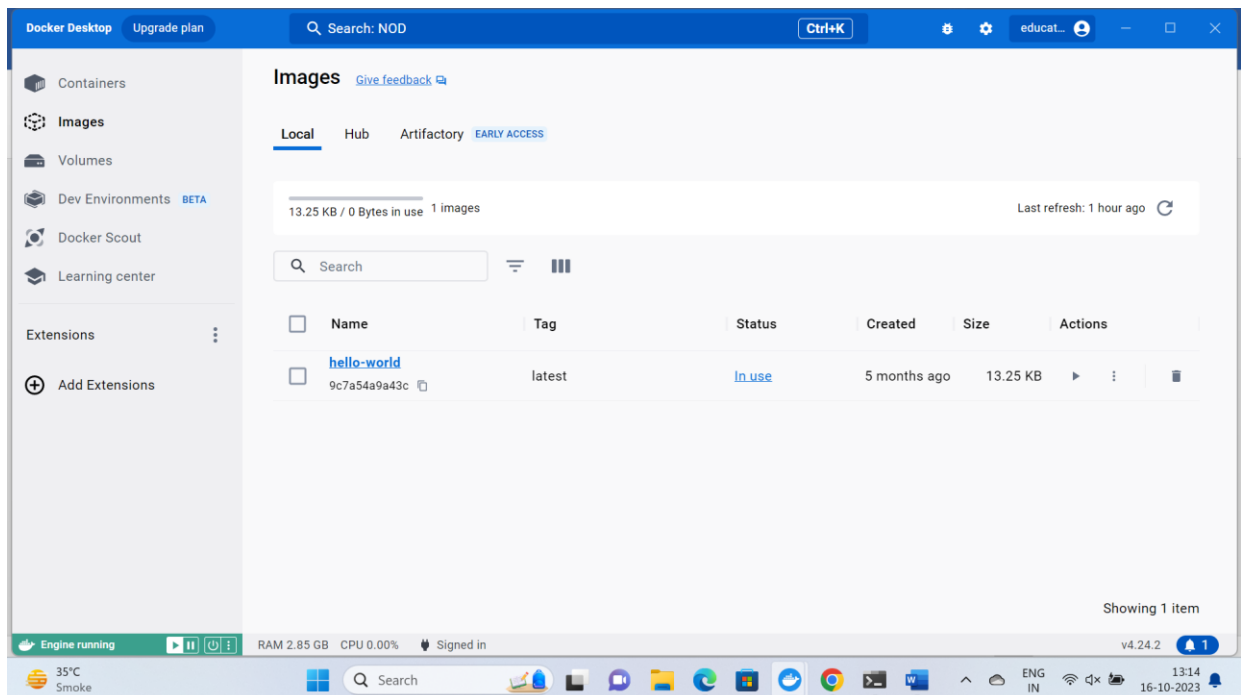
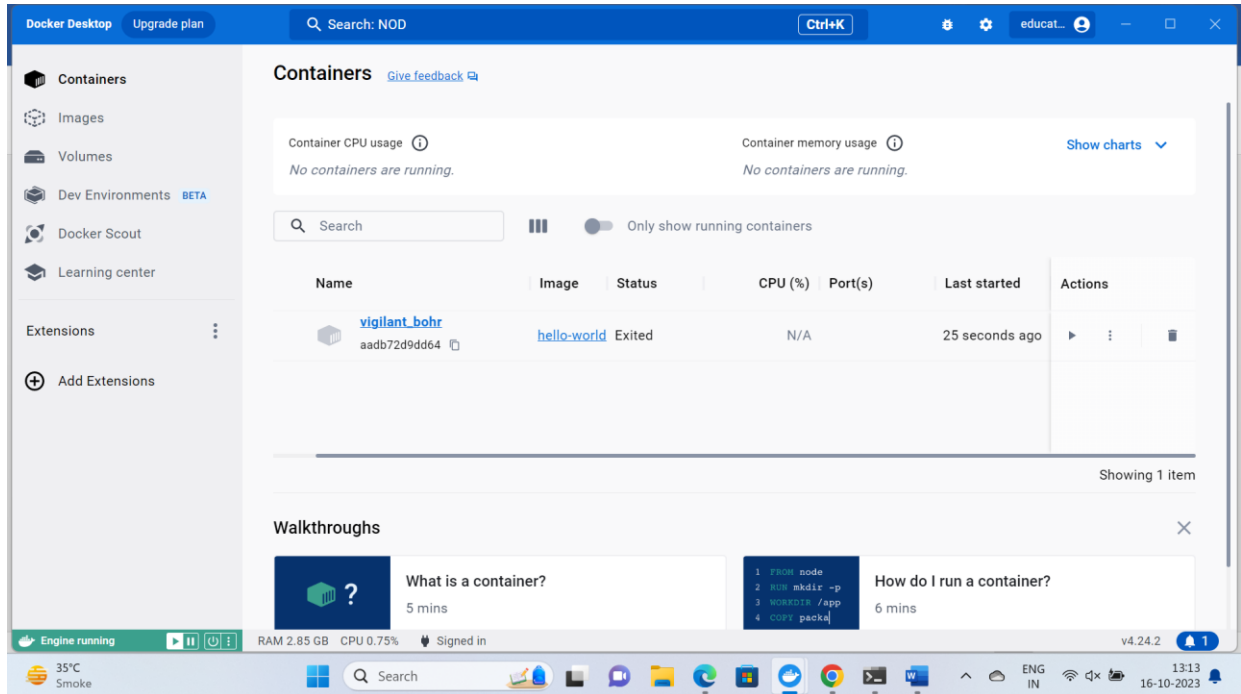
C:\Users\aman1>
```

NAME – AMAN SINGH

BATCH – T23

ROLL NO - 128

## DOCKER CONTAINER





NAME – AMAN SINGH

BATCH – T23

ROLL NO - 128

### **CONCLUSION:**

Here, we have studied about docker, its architecture and lifecycle of a docker container. Also we have successfully created and run a container in docker.