St. Francis Institute of Technology, Mumbai-400 103

**Department Of Information Technology**

A.Y. 2024-2025

Class: TE-ITA/B, Semester: V

Subject: **DevOps Lab**

**Experiment – 9: a. To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers.**

**b. To learn Dockerfile instructions, build an image for a sample web application using Dockerfile**

1. **Aim: a.** To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers. b. To learn Dockerfile instructions, build an image for a sample web application using Dockerfile.
2. **Objectives:** Aim of this experiment is that, the students will learn:

* Introduction to Docker Architecture
* To use Docker to Build, ship and manage applications using containerization
* To understand concept of containerization
* To analyze the Containerization of OS images and deployment of applications over Docker

1. **Outcomes:** After study of this experiment, the students will learn following:

* Introduction to Docker Architecture
* Container Life Cycle
* Understanding images and containers
* Publishing image on Docker Hub.
* Create and implement docker images using Dockerfile.
* Container Lifecycle and working with containers.
* To Build, deploy and manage web or software application on Docker Engine.

1. **Prerequisite:** None
2. **Requirements:** Docker Desktop,JDK, Personal Computer, Windows operating system, Internet Connection, Microsoft Word.
3. **Pre-Experiment Exercise:**

**Brief Theory:** Refer shared material

1. **Laboratory Exercise**
   * + 1. **Procedure:**

**a. Answer the following:**

* What are docker containers and docker images?
* Explain docker architecture with diagram.
* What is a Dockerfile?
* Explain Dockerfile commands with syntax and example.

**b**. **Execute following (Refer the shared material) and attach screenshots:**

* + - Create Docker Hub account
    - 12
    - 11
    - 10
    - 9
    - 8
    - 7
    - 6
    - 5
    - 4
    - 3
    - 2
    - 1
    - 1
    - 2
    - 3
    - 4
    - 5
    - 6
    - 7
    - 8
    - 9
    - 10
    - 11
* Download and install Docker Desktop
* Execute docker commands to manage images and interact with containers
  + - Create a Dockerfile
* Create an html file
* Build and run the web application on Docker Engine

1. **Post-Experiments Exercise**
2. **Extended Theory:**

Nil

1. **Questions:**

* Write all Docker commands with syntax and example
* Explain differences between VMs and docker containers
* What is a Docker cheat sheet?
* Why do we require volumes for Docker?

1. **Conclusion:**

* Write what was performed in the experiment.
* Write the significance of the topic studied in the experiment.

1. **References:**

<https://www.youtube.com/watch?v=zJ6WbK9zFpI>

<https://www.simplilearn.com/tutorials/docker-tutorial>

<https://www.edureka.co/blog/docker-explained/>

<https://www.youtube.com/watch?v=zJ6WbK9zFpI>

<https://www.simplilearn.com/tutorials/docker-tutorial>

<https://www.edureka.co/blog/docker-explained/>

https://www.youtube.com/watch?v=3c-iBn73dDE

**7. Laboratory Exercise**

A. Procedure:

1. **What are docker containers and docker images?**

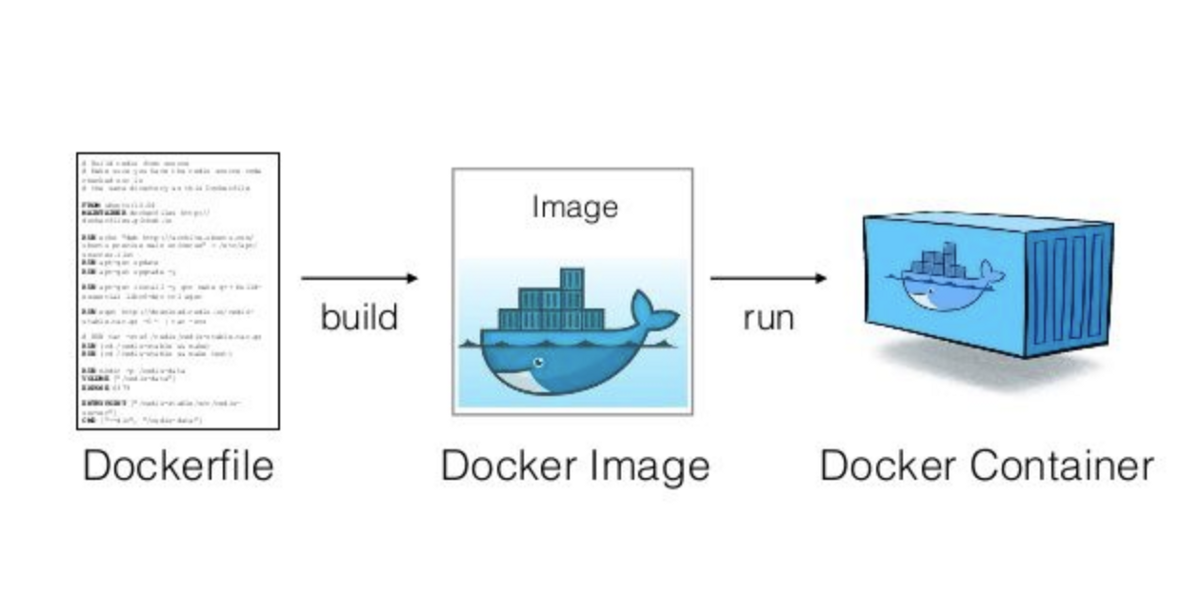
**ANS:**

**Docker Containers**: Docker containers are lightweight, standalone, and executable software packages that include everything needed to run a piece of software, including the code, runtime, libraries, and dependencies. Containers are isolated from each other and the host system, which allows for consistent environments and easier deployment across different systems.

**Docker Images**: A Docker image is a read-only template used to create containers. It contains the application code along with the dependencies and configurations necessary to run the application. Images can be versioned and stored in repositories (like Docker Hub), allowing for easy sharing and deployment.

A Docker image is a blueprint of code that is executed in a Docker container. To use Docker, you add layers of core functionalities to a Docker image that are then used to create a running container. In other words, a Docker container is a running instance of a Docker image. You can create many containers from the same image, each with its own unique data and state.

Using containers standardizes and simplifies **development, operations, and testing**. To use containers effectively, make sure developers, operations engineers, and testers create consistent environments.



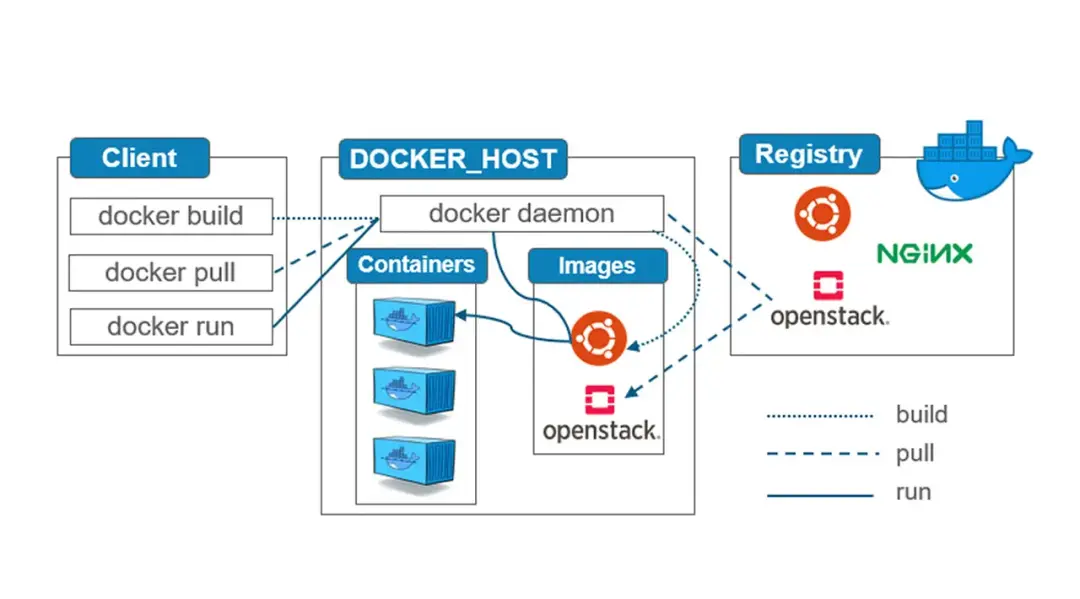
1. **Explain docker architecture with diagram.**

**ANS:**

**Docker Architecture**: Docker follows a client-server architecture. Here’s a brief overview of its components:

1. **Docker Client**: This is the primary interface through which users interact with Docker. It sends commands to the Docker daemon.
2. **Docker Daemon**: The Docker daemon (dockerd) runs on the host machine and is responsible for managing Docker containers, images, networks, and volumes. It listens for API requests from the Docker client.
3. **Docker Registry**: This is a storage and distribution system for Docker images. Docker Hub is the default public registry where users can store and share their images.
4. **Containers**: These are instances of Docker images that run applications.
5. **Images**: These are the blueprints for containers. They are created from a Dockerfile and can be stored in a registry.

Docker's architecture promotes a modular and efficient approach to software development and deployment. By separating concerns through components like the client, daemon, images, containers, and registries, Docker provides a powerful platform for containerization that enhances scalability, consistency, and resource utilization.



1. **What is a Dockerfile?**

**ANS**:

A **Dockerfile** is a text file that contains a series of instructions for building a Docker image. It automates the process of creating images by specifying the necessary environment, dependencies, and configurations required to run an application. This enables developers to package their applications in a standardized way, ensuring that the environment remains consistent across different systems and deployments.

#### Key Features of a Dockerfile

1. **Declarative Syntax**: Dockerfiles use a simple, declarative syntax to specify the steps for building an image. This makes it easy to read and understand.
2. **Layered Structure**: Each command in a Dockerfile creates a new layer in the Docker image. Layers are cached, which speeds up the build process when changes are made, as unchanged layers are reused.
3. **Reproducibility**: By using a Dockerfile, developers can ensure that the same image is built every time, regardless of the environment. This eliminates inconsistencies and "it works on my machine" issues.
4. **Version Control**: Dockerfiles can be versioned in source control systems, allowing teams to track changes and collaborate effectively.

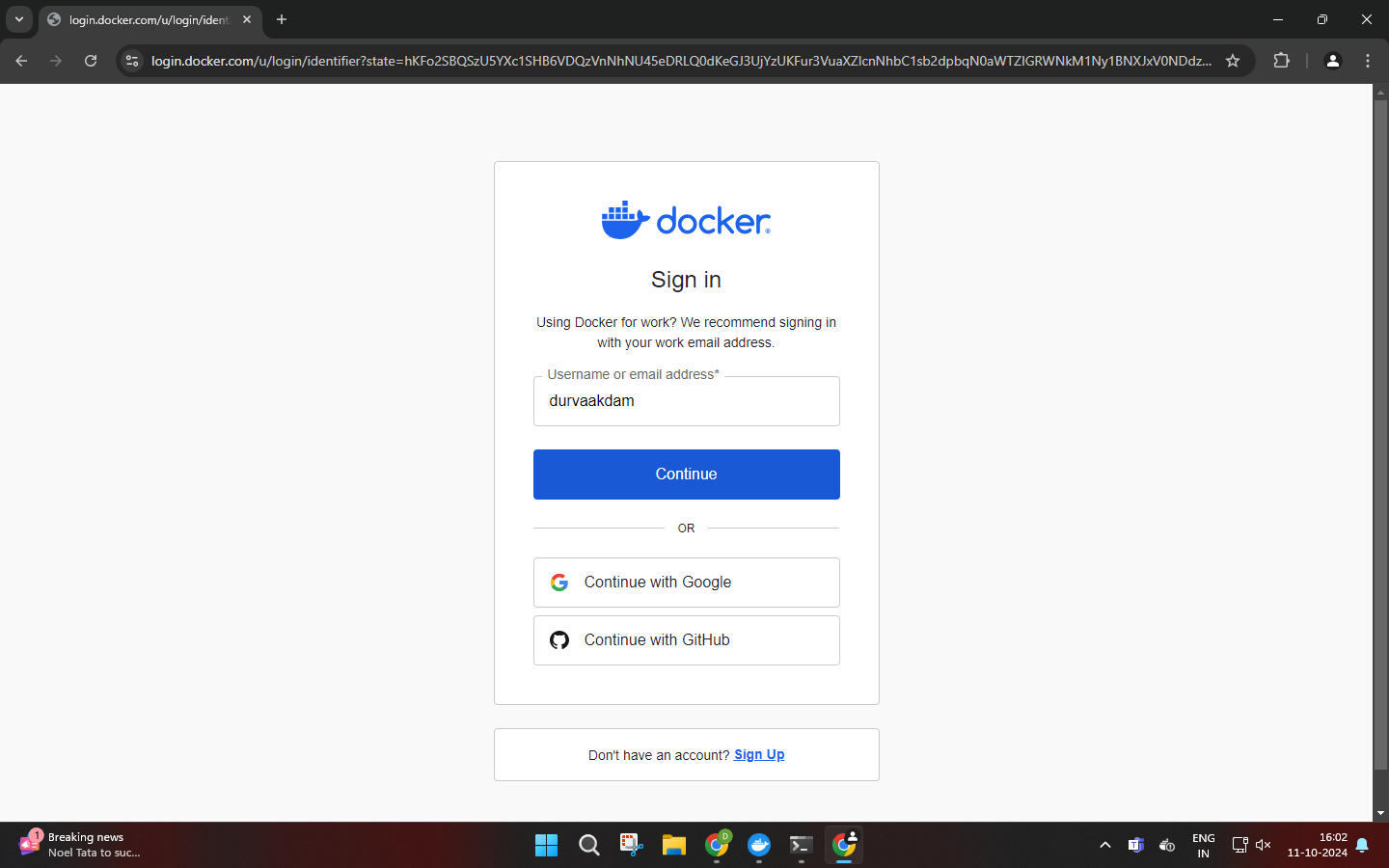
Dockerfiles are key to the containerization process because they ensure that applications are lightweight, portable, and reproducible. By specifying dependencies, configurations, and build steps, Dockerfiles eliminate discrepancies between development, testing, and production environments. Dockerfiles allow teams to automate the process of creating, testing, and deploying containers, leading to increased efficiency.

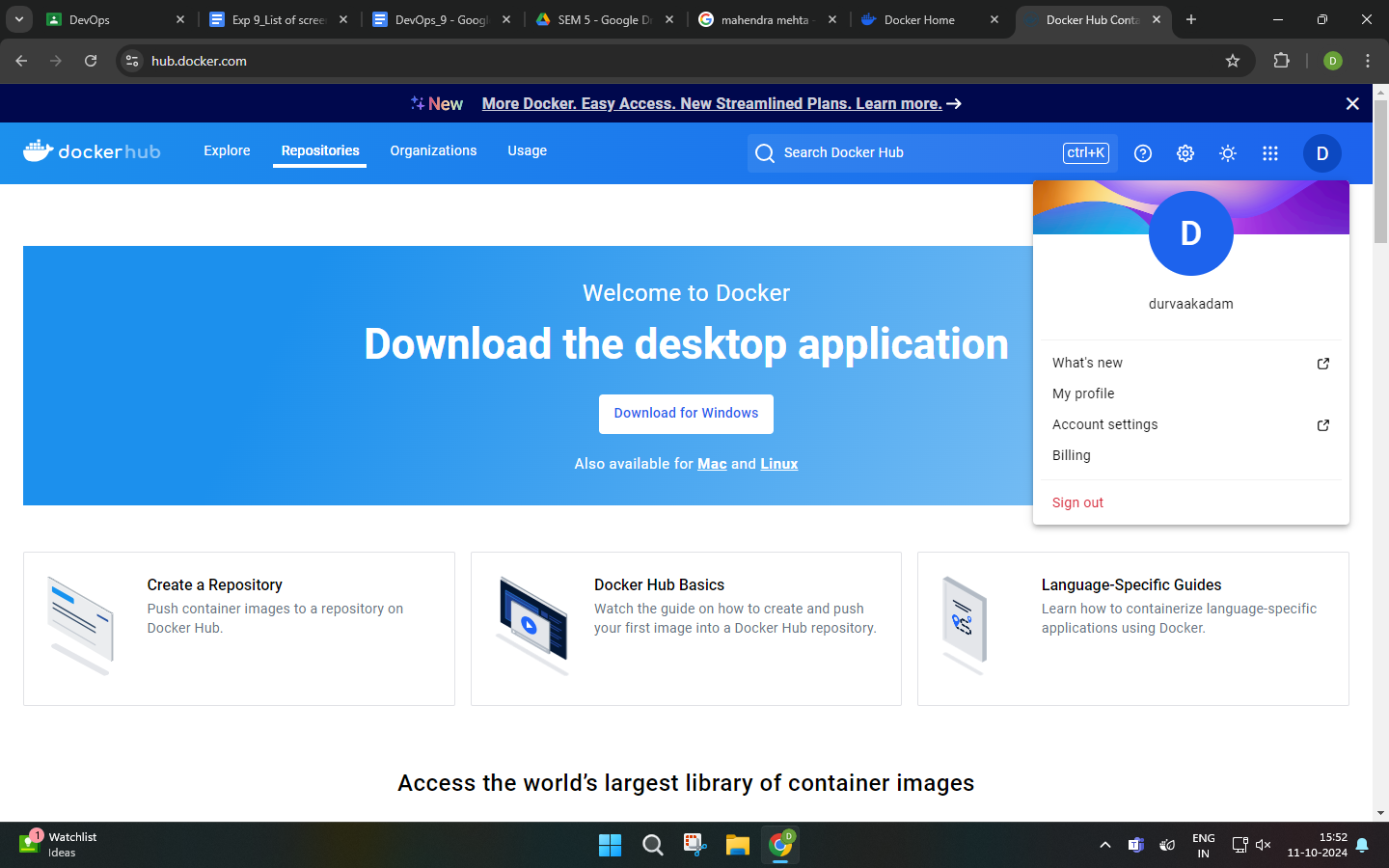
1. **Explain Dockerfile commands with syntax and example.**

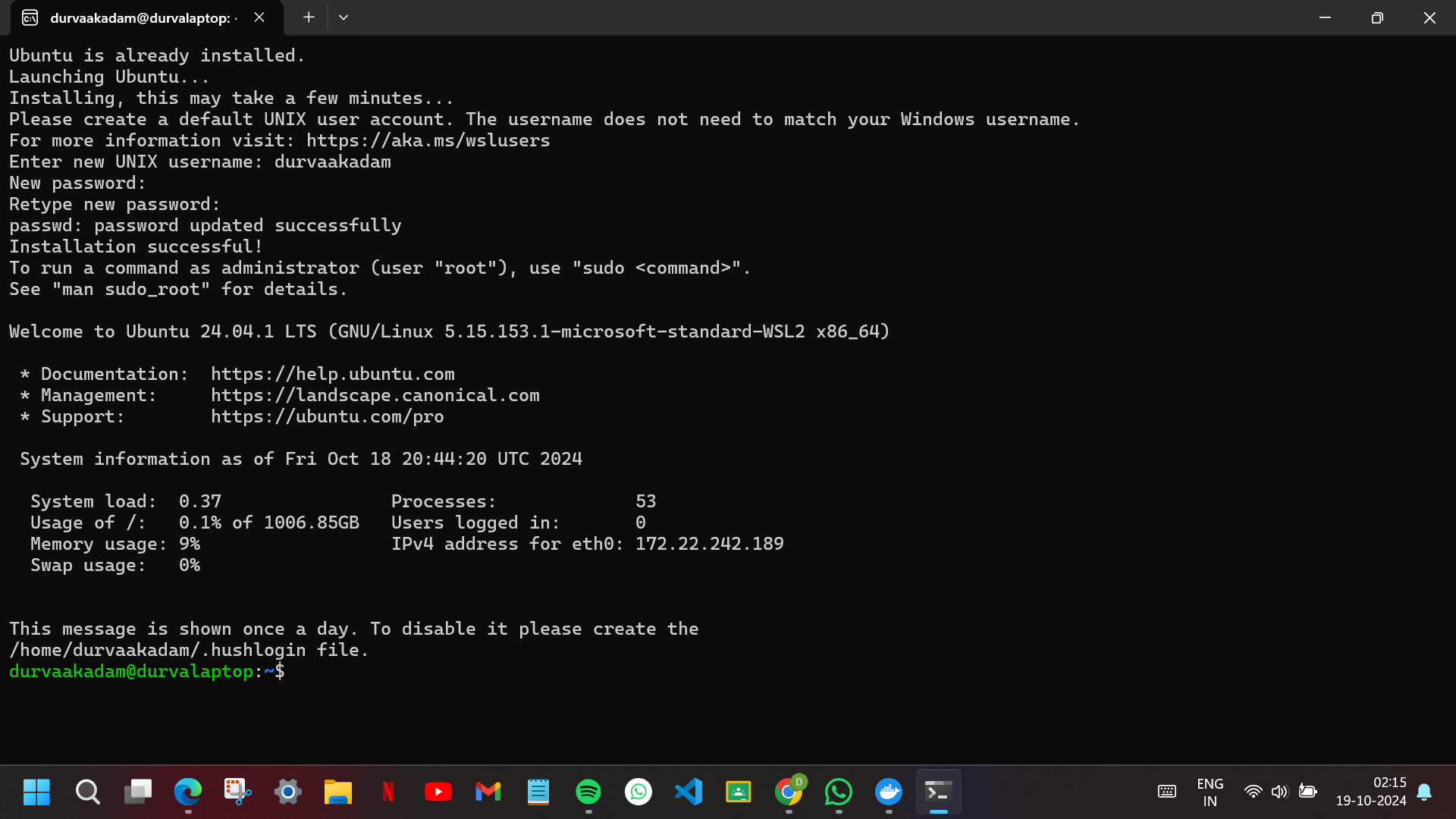
**ANS:** Commonly used Dockerfile commands with their syntax and examples are:

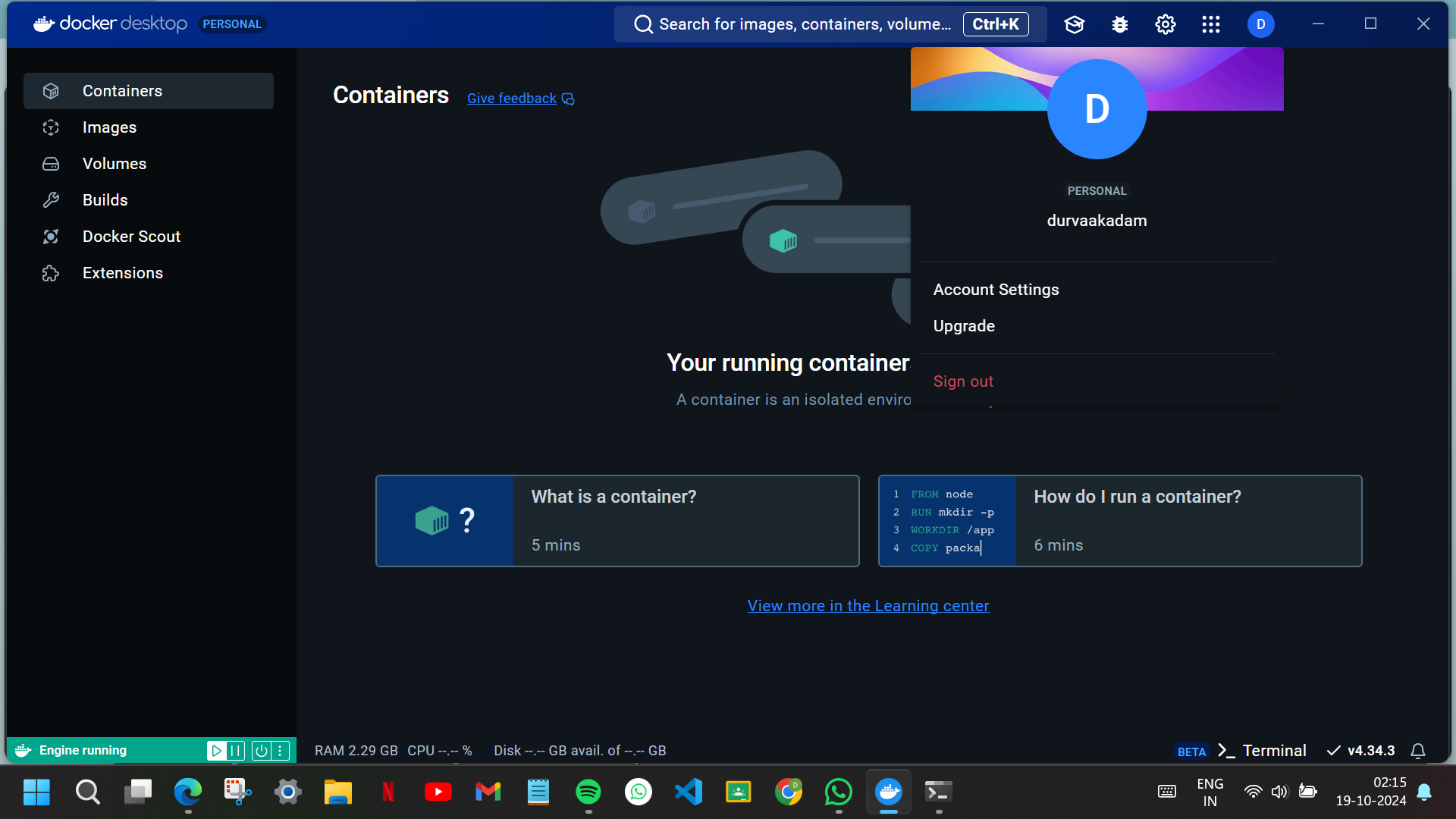
1. **FROM**: Defines the base image for the build process.  
   Syntax: FROM <image>:<tag>  
   Example: FROM ubuntu:20.04
2. **RUN**: Executes commands during the image build process.  
   Syntax: RUN <command>  
   Example: RUN apt-get update && apt-get install -y python3
3. **COPY**: Copies files or directories from the host to the container.  
   Syntax: COPY <source> <destination>  
   Example: COPY . /app
4. **WORKDIR**: Sets the working directory for subsequent Dockerfile instructions.  
   Syntax: WORKDIR <path>  
   Example: WORKDIR /app
5. **CMD**: Specifies the default command to run when the container starts.  
   Syntax: CMD ["executable", "param1"]  
   Example: CMD ["python3", "app.py"]
6. **EXPOSE**: Informs Docker that the container listens on a specific network port.  
   Syntax: EXPOSE <port>  
   Example: EXPOSE 80
7. **ENV**: Sets an environment variable inside the container.  
   Syntax: ENV <key>=<value>  
   Example: ENV APP\_ENV=production
8. **ENTRYPOINT**: Specifies the main command to run inside the container.  
   Syntax: ENTRYPOINT ["executable", "param1"]  
   Example: ENTRYPOINT ["python3", "app.py"]
9. **ADD**: Copies files and can extract compressed archives or download files from URLs. Syntax: ADD <source> <destination>  
   Example: ADD app.tar.gz /app
10. **USER**: Specifies the user under which the container will run.  
    Syntax: USER <username>  
    Example: USER appuser

INSTALLATION OF DOCKER:

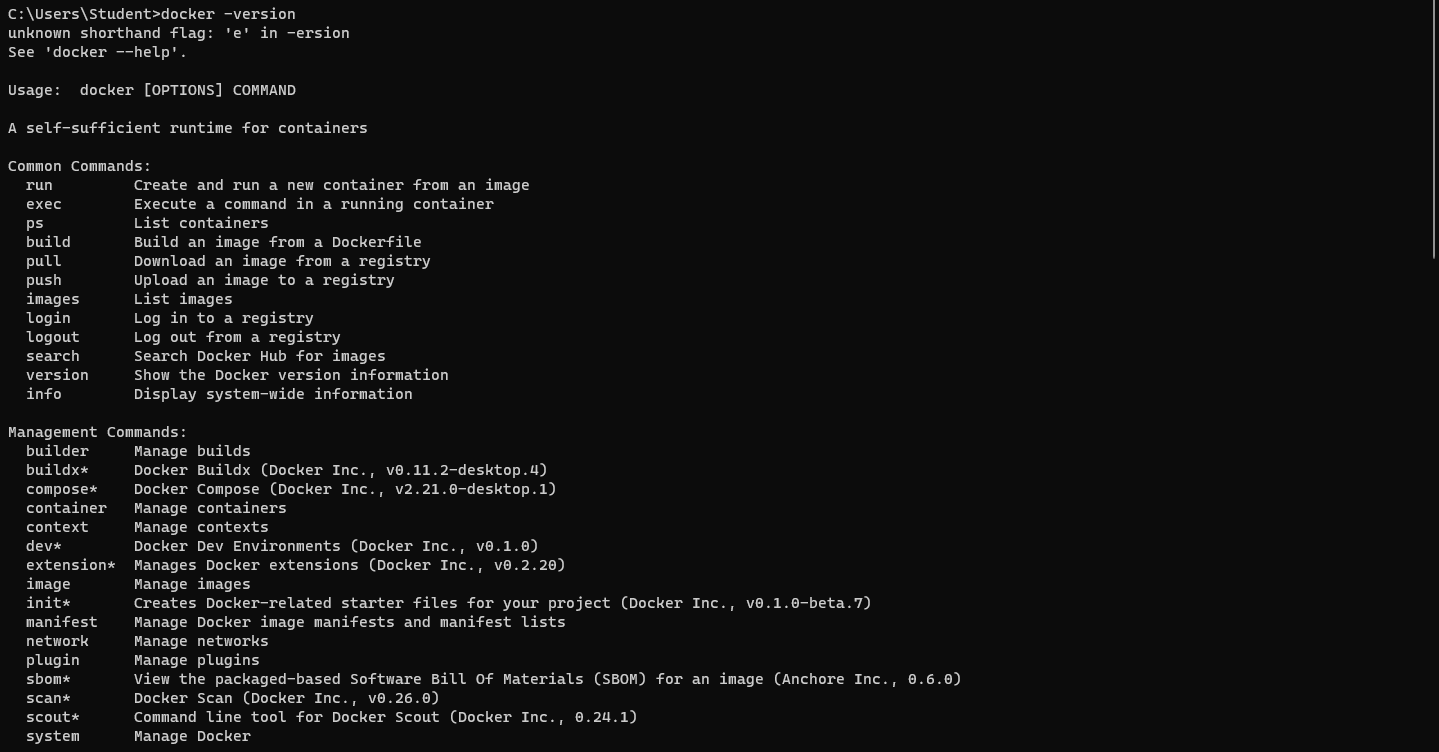


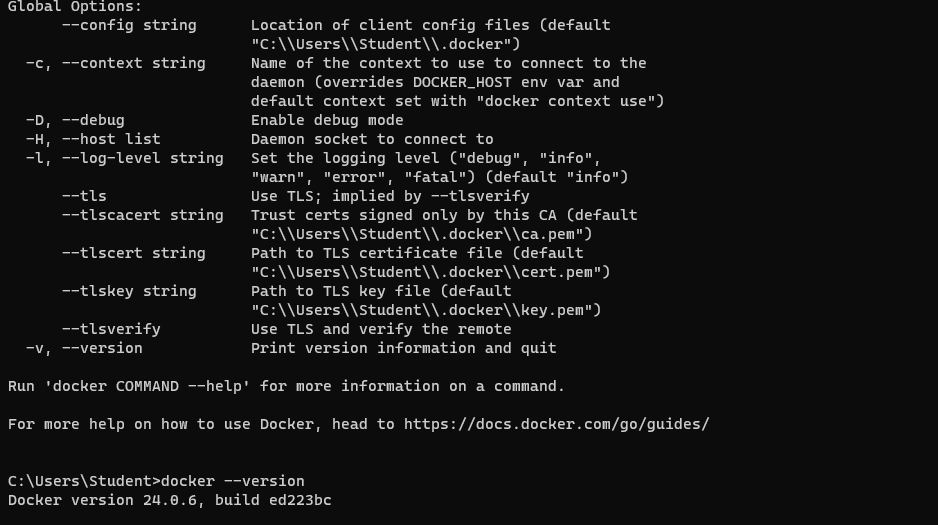




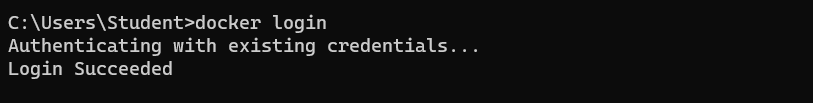


1. Docker Version

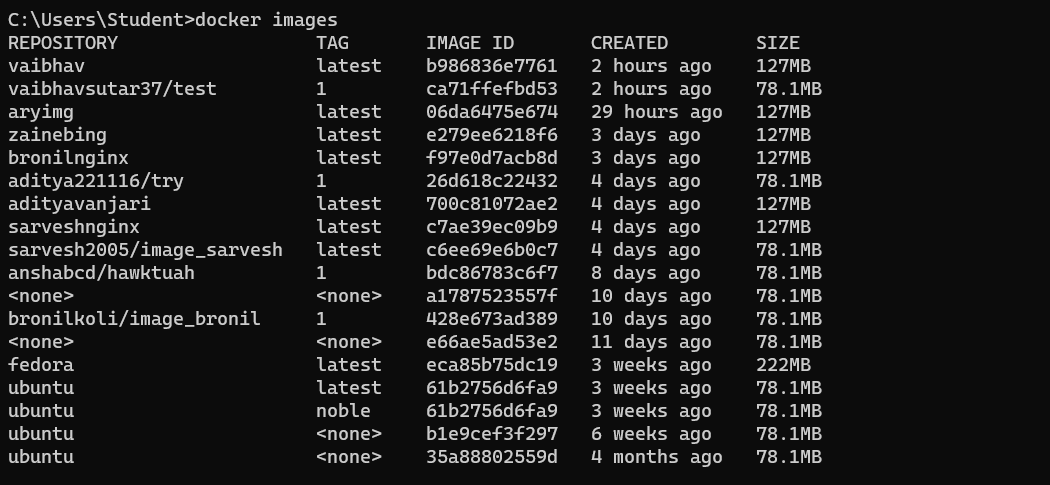




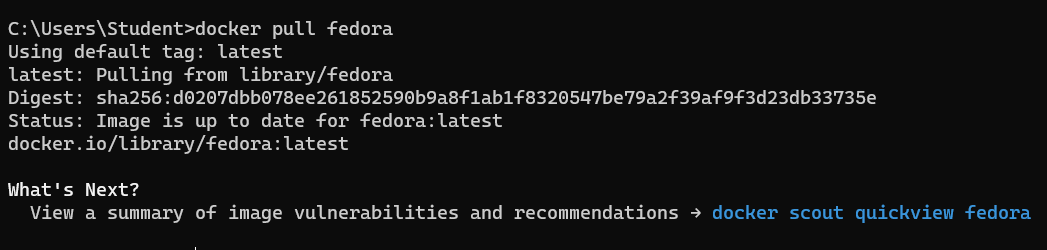
2. Docker login



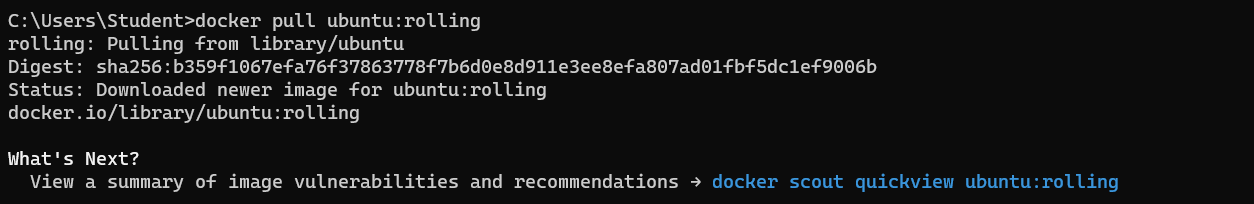
3. Docker images



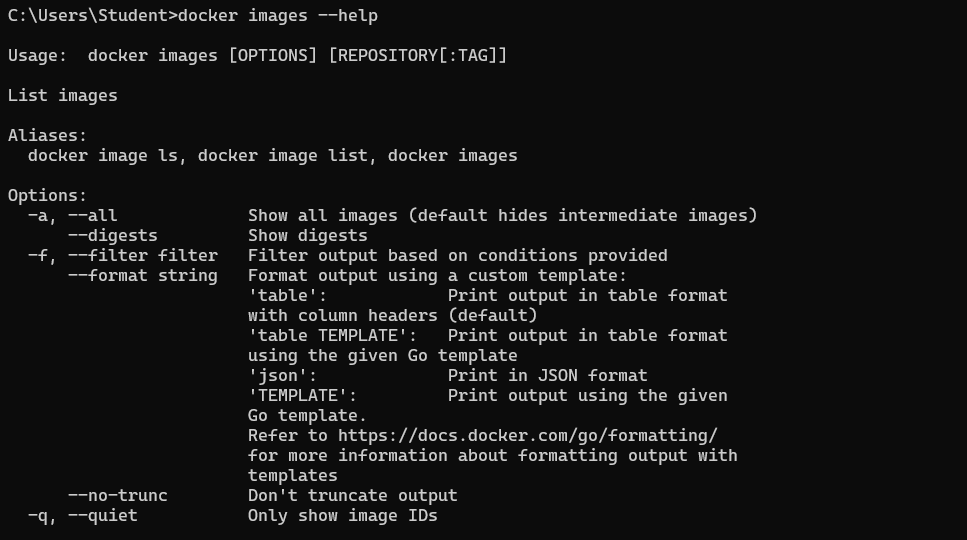
4. Docker pull image

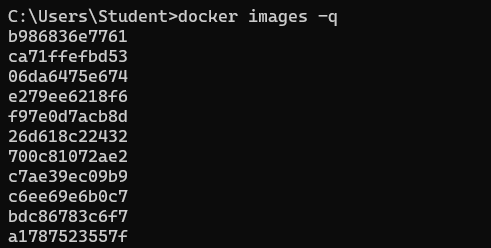
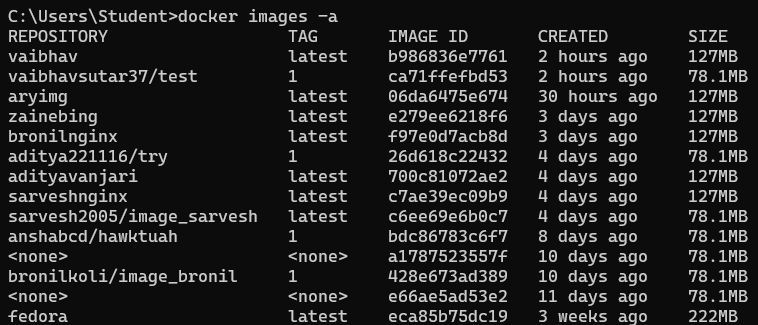


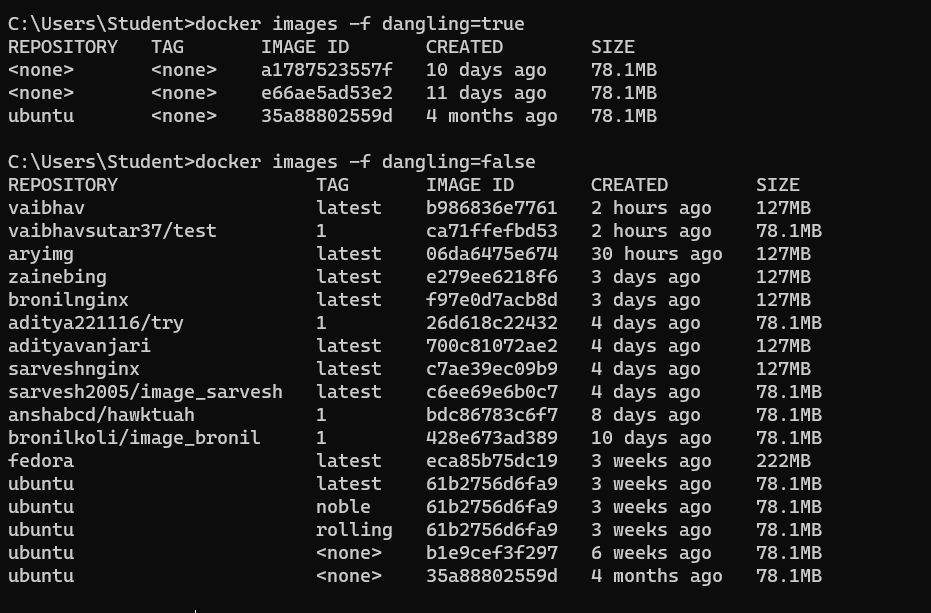
5. Docker pull image-tag



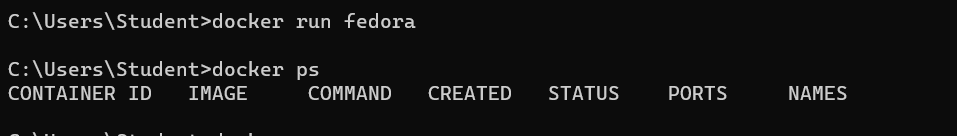
6. Docker images help

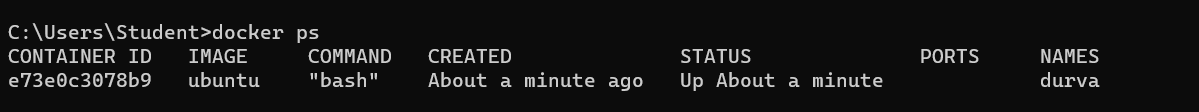




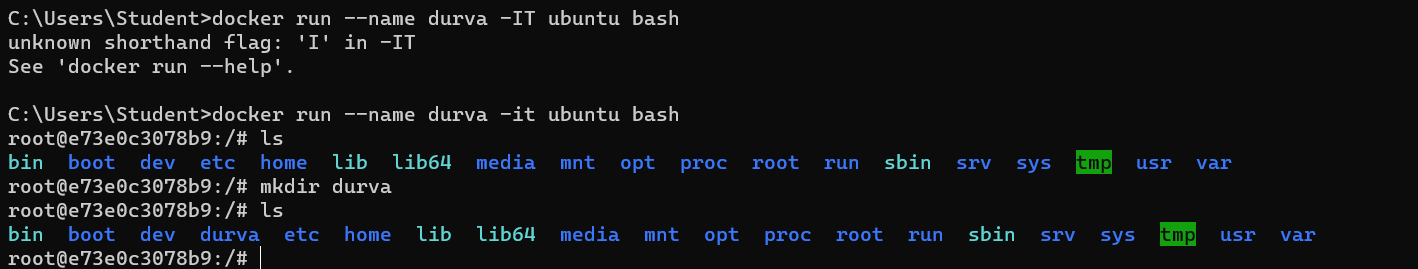


7. Docker run commands

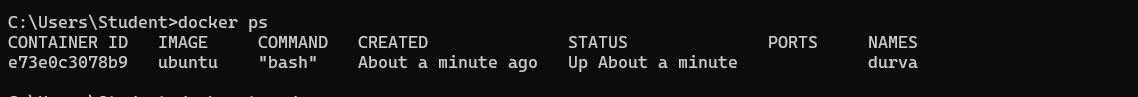




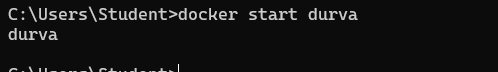




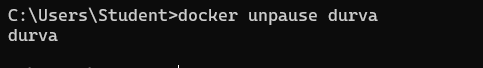
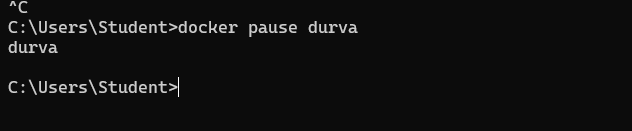
8. Docker ps



9. Docker start container



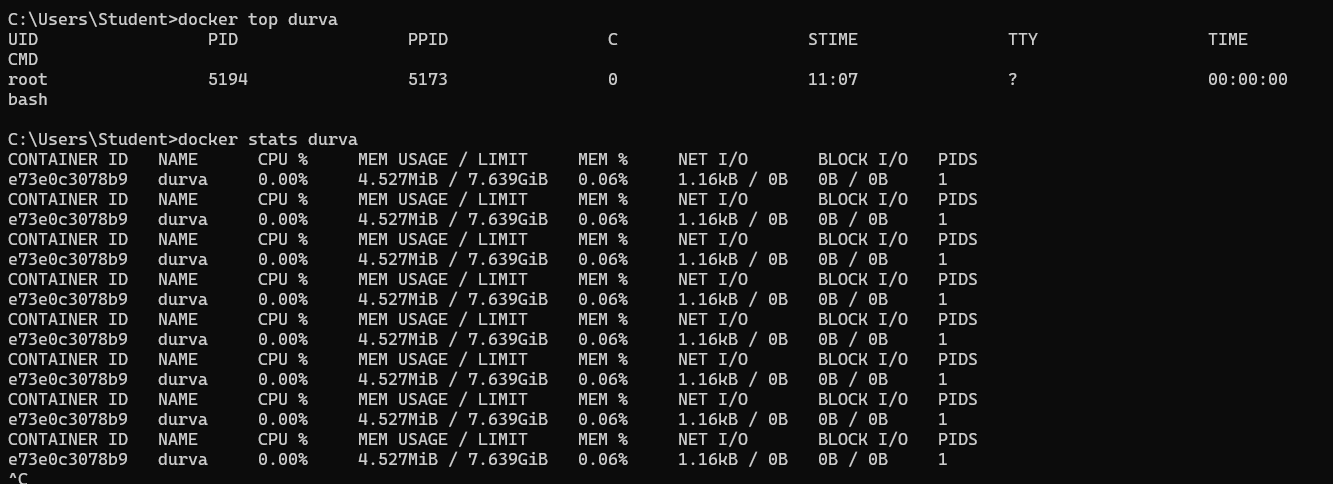
10. Docker pause container



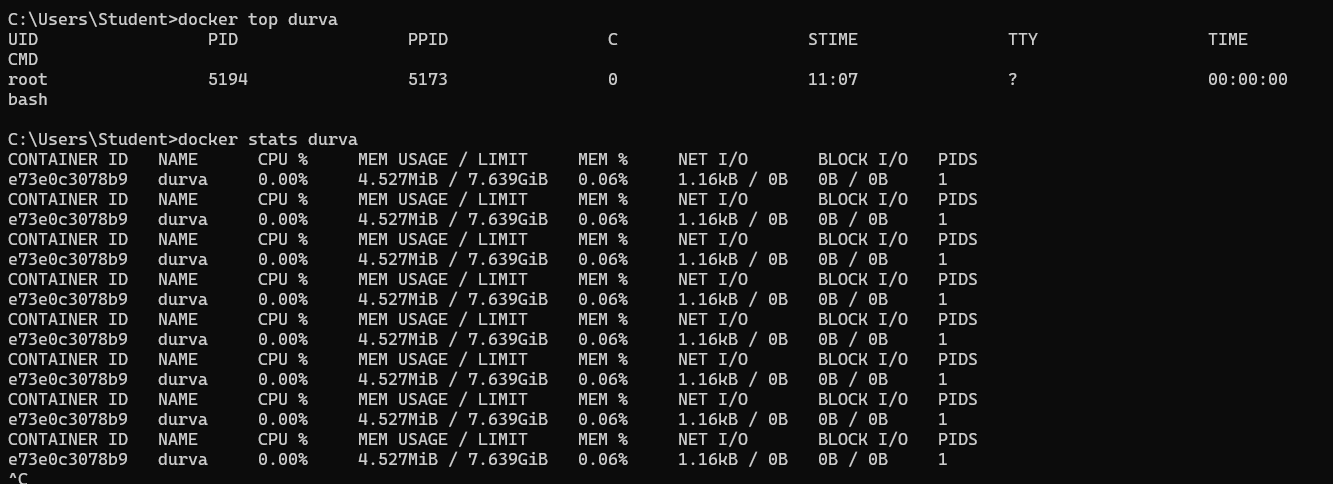
11. Docker stop container



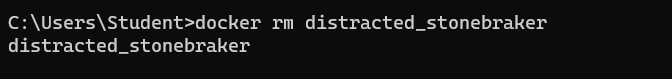
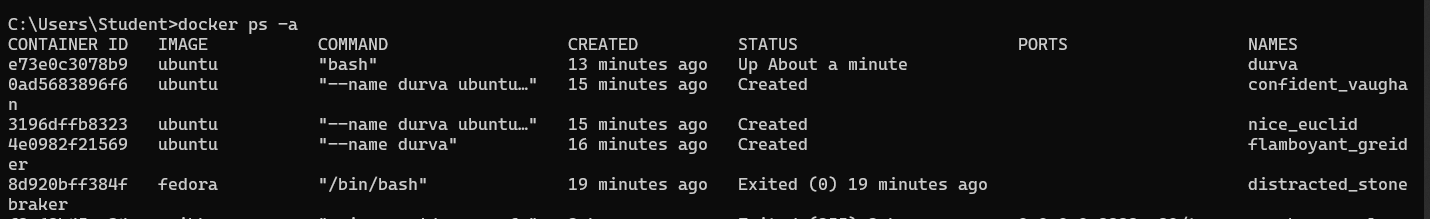
12. Docker top container

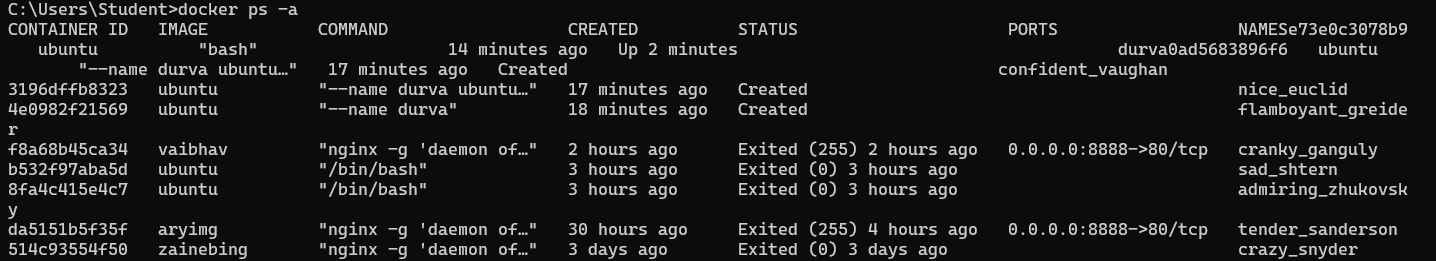


13. Docker stats container

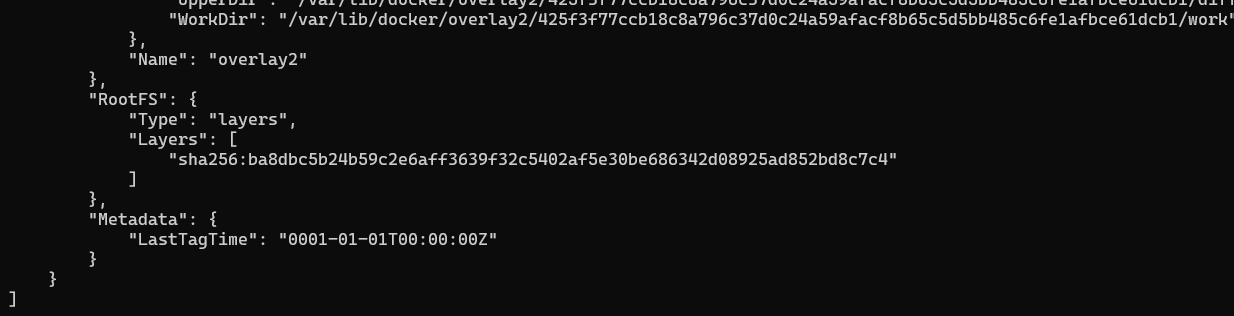
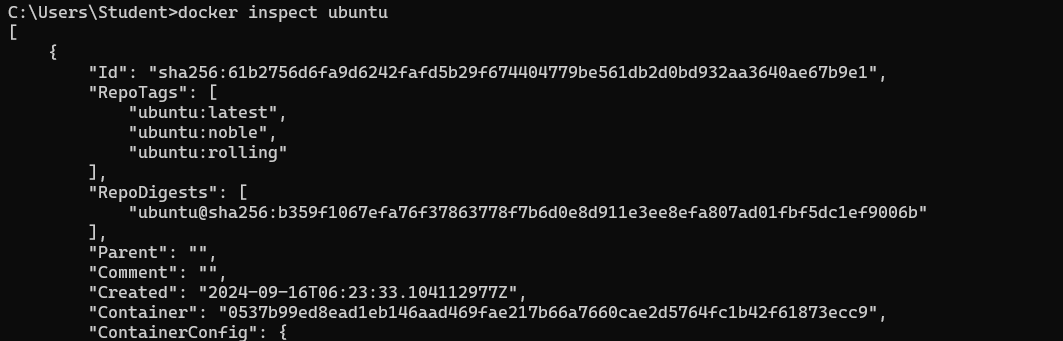


14. Docker rm container

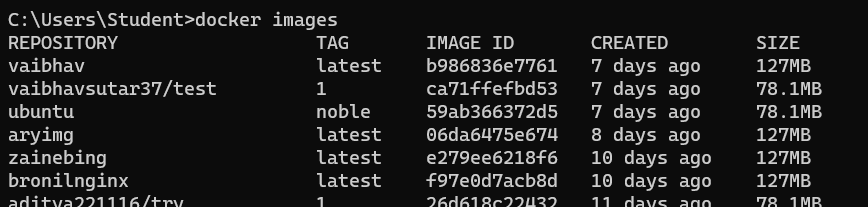
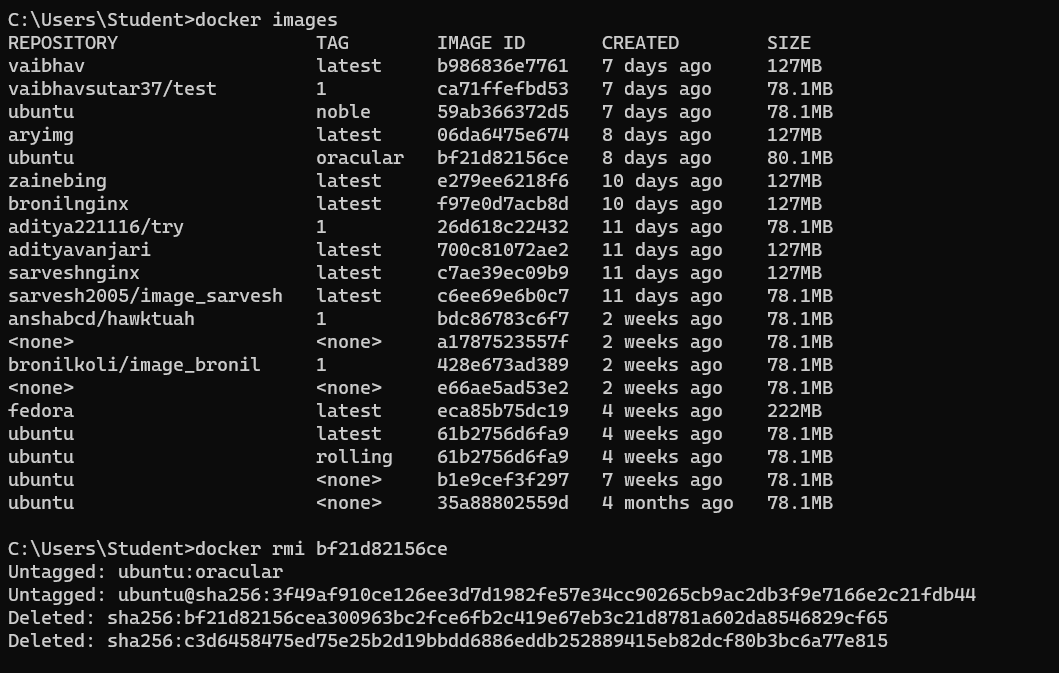
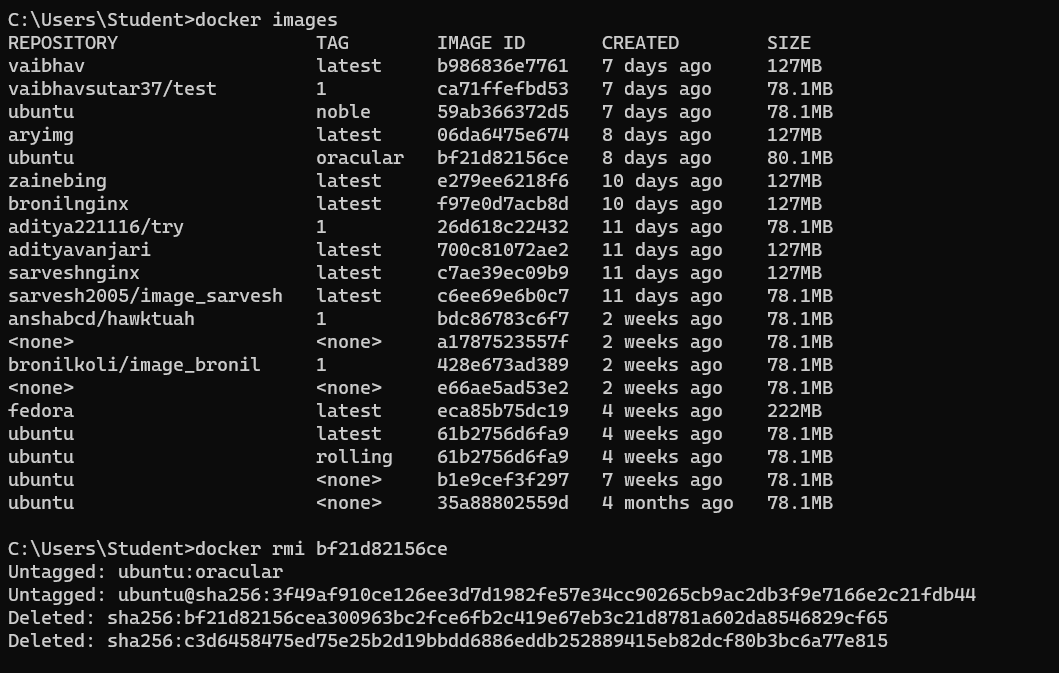


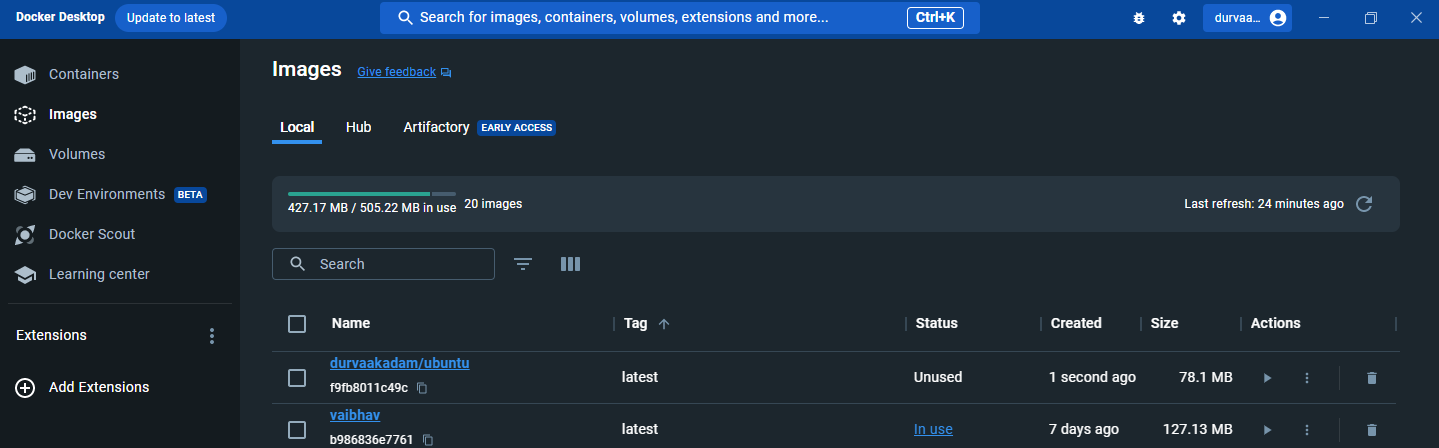
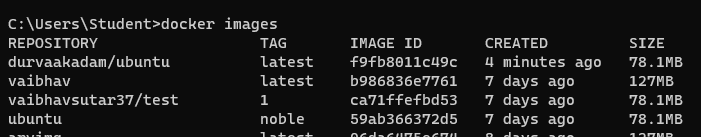
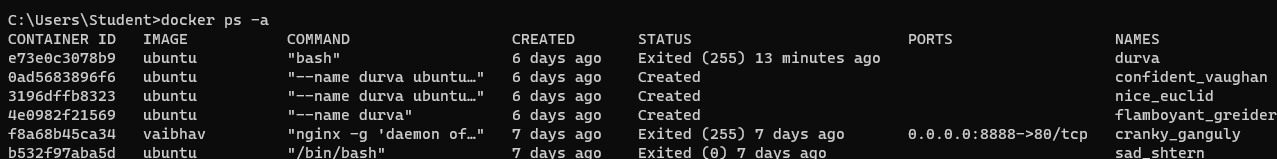


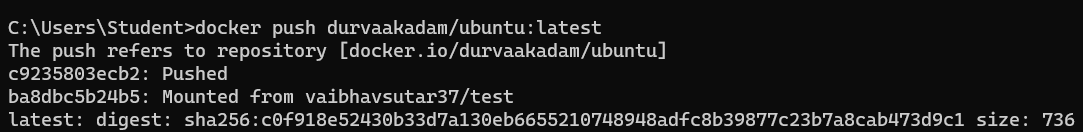
15. Docker inspect

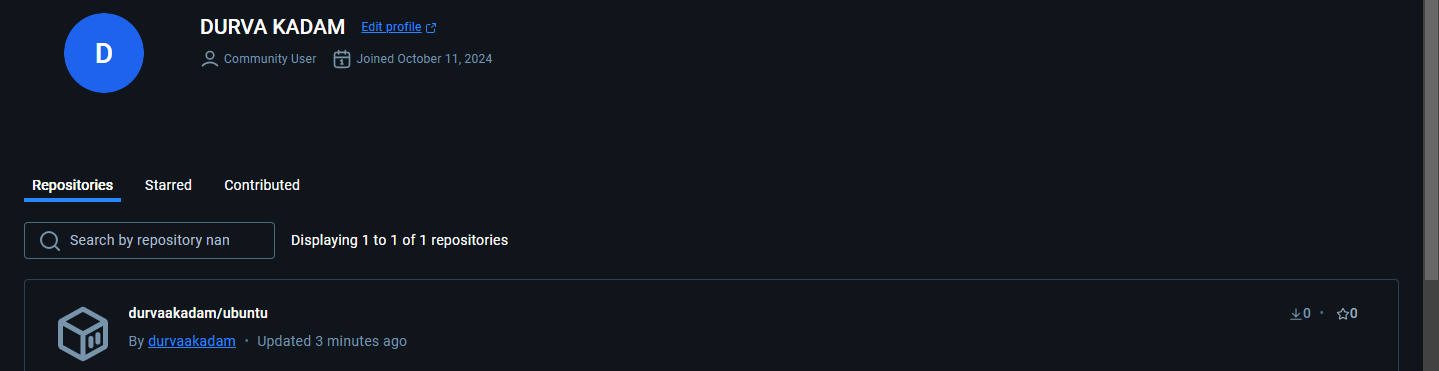


16. Docker rmi

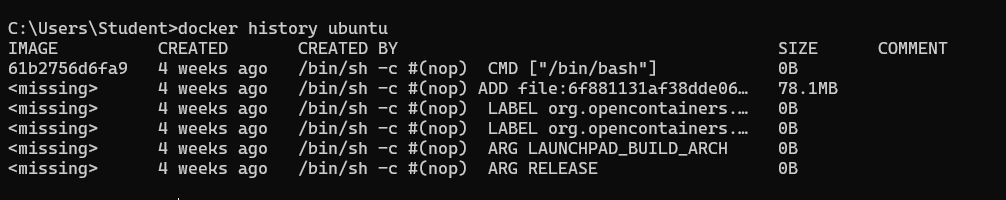


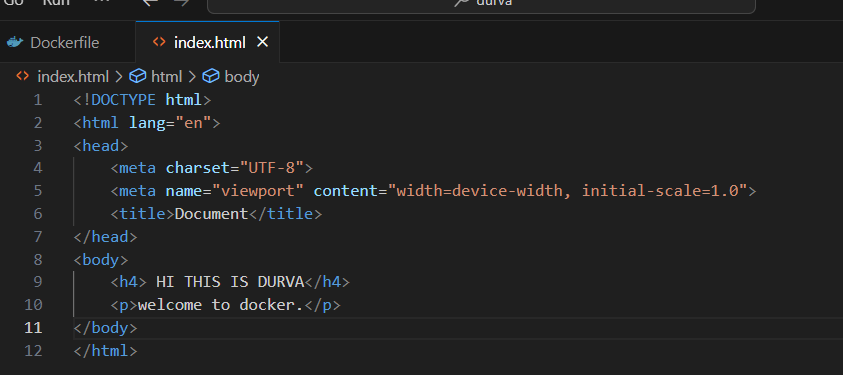
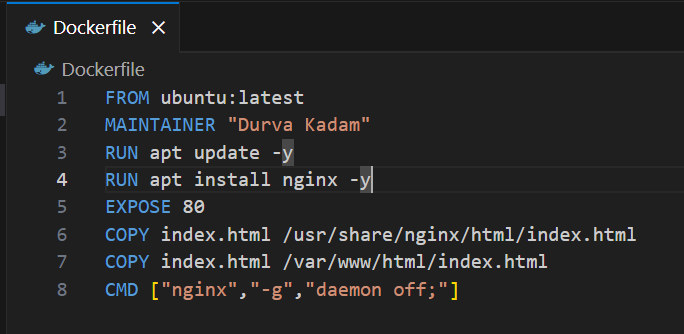
17. Docker commit

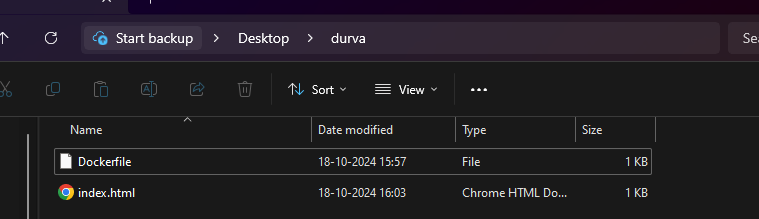
18. Docker push



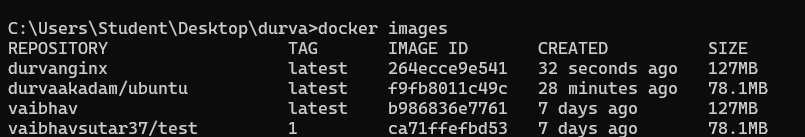
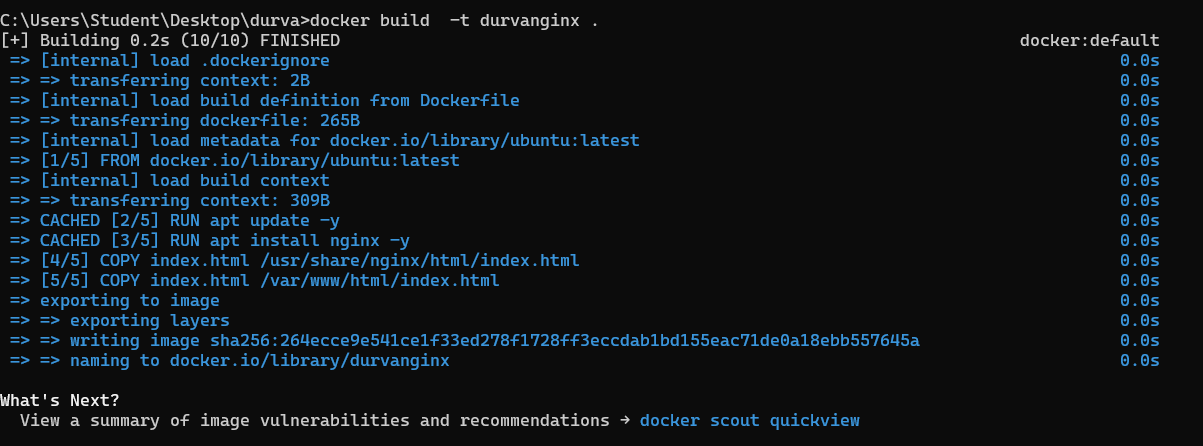
19. Docker history image



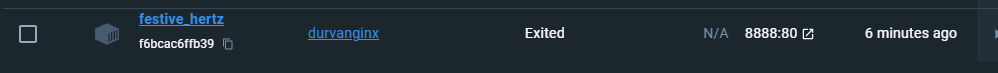
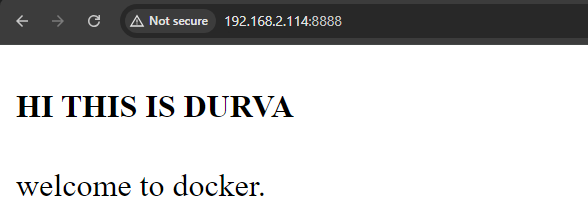
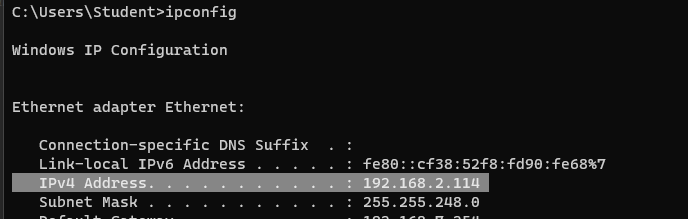
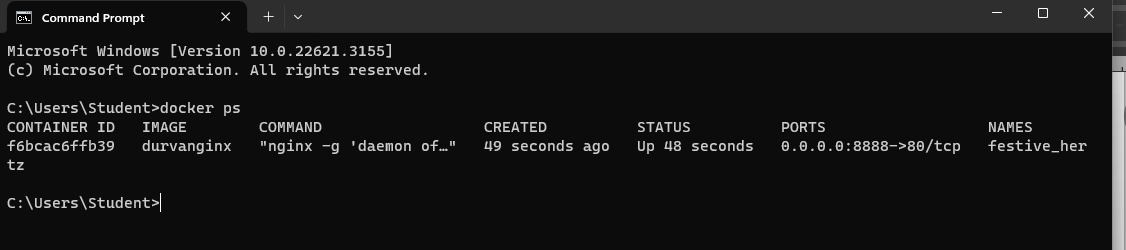
DOCKER FILE AND HTML CODE:



open the folder in cmd



open a new cmd again

\