	Pimpri Chinchwad Education Trust's **Pimpri Chinchwad College of Engineering (PCCoE)** (An Autonomous Institute) Affiliated to Savitribai Phule Pune University(SPPU) ISO 21001:2018 Certified by TUV	

Course: DevOps Laboratory	**Code:** BIT26VS01
Name: Amar Vaijinath Chavan	**PRN:** 124B2F001
Assignment 9: Exploration of Docker Architecture and Containerization of a Python Application.	

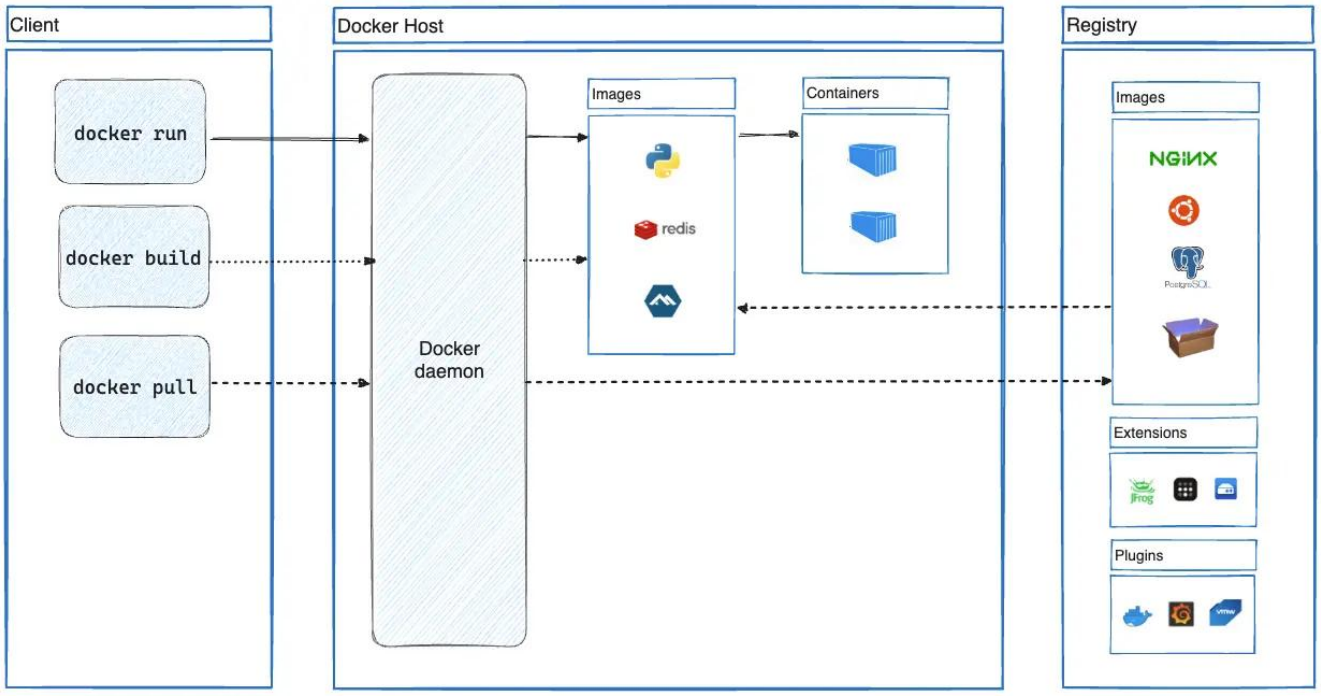
Aim: To explore Docker architecture and core commands by containerizing a Python Flask using a Dockerfile.

Objectives:

1. To understand the Docker Client-Server architecture.
2. To learn the syntax and instructions of a Dockerfile.
3. To build, run, and manage Docker containers using the CLI.

Theory

Docker Architecture



Docker uses a client-server architecture. The **Docker Client** talks to the **Docker Daemon (dockerd)**, which does the heavy lifting of building, running, and distributing your Docker containers.

- **The Docker Daemon:** A persistent background process that manages Docker objects such as images, containers, networks, and volumes.
- **The Docker Client:** The primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out.
- **Docker Registries:** A registry stores Docker images. Docker Hub is a public registry that anyone can use.
- **Docker Objects:**
- **Images:** A read-only template with instructions for creating a Docker container.
- **Containers:** A runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.

Dockerfile Instructions

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

- **FROM:** Sets the Base Image for subsequent instructions.
- **WORKDIR:** Sets the working directory for any subsequent instructions.
- **COPY:** Copies new files or directories from the source and adds them to the filesystem of the container.
- **RUN:** Executes any commands in a new layer on top of the current image and commits the results.
- **EXPOSE:** Informs Docker that the container listens on the specified network ports at runtime.
- **CMD:** Provides defaults for an executing container. There can only be one CMD instruction in a Dockerfile.

Practical Procedure / Steps

Step 1: Application Structure

The application is a **Student Management System** built using the Flask framework. The project structure consists of the following files:

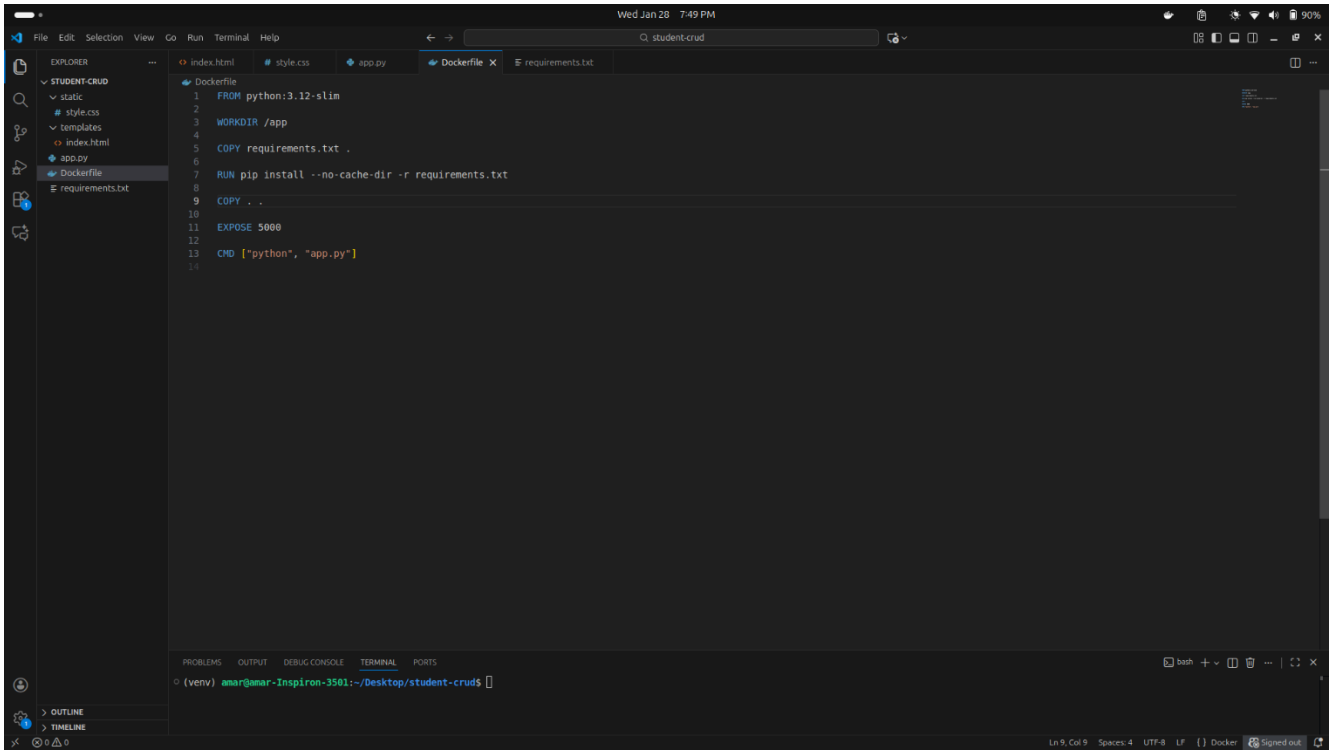
- app.py: The backend Flask logic for CRUD operations.
- requirements.txt: Lists the flask dependency.
- templates/index.html: The frontend UI.
- static/style.css: Custom CSS for the UI.

Step 2: Defining the Dockerfile

A Dockerfile was created in the root directory with the following configuration:

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
```

RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]



Step 3: Building the Docker Image

Using the terminal, the image was built and tagged as student-crud-app.
docker build -t student-crud-app .
The build process involves 10 steps, each creating a layer in the final image.

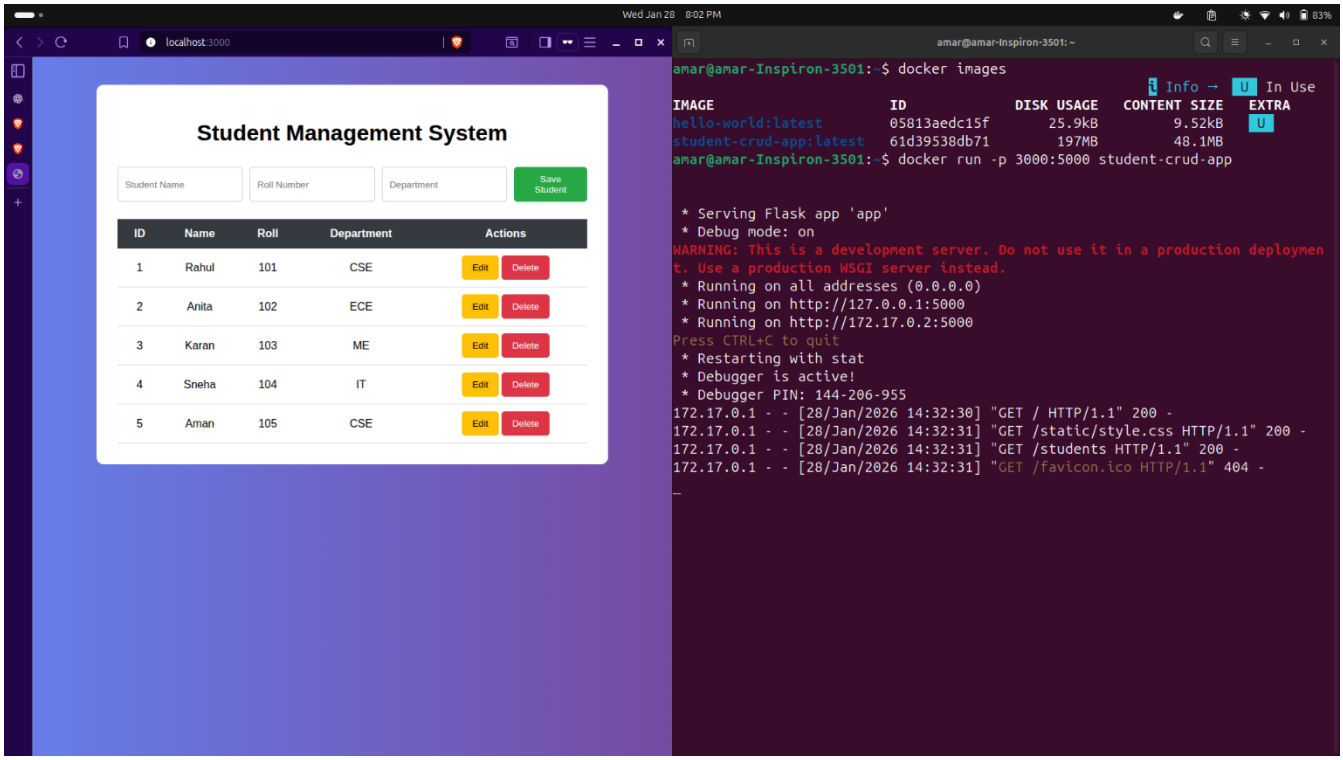
```
(venv) amar@amar-Inspiron-3501:~/Desktop/student-crud$ docker build -t student-crud-app .
[+] Building 128.6s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 201B
 => [internal] load metadata for docker.io/library/python:3.12-slim
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build context
 => => transferring context: 224B
 => [1/5] FROM docker.io/library/python:3.12-slim@sha256:5e2dbd4bbdd9c0e67412aea9463906f74a22c60f89eb7b5bbb7d45b66a2b68a6
 => => resolve docker.io/library/python:3.12-slim@sha256:5e2dbd4bbdd9c0e67412aea9463906f74a22c60f89eb7b5bbb7d45b66a2b68a6
 => => sha256:671677b67e7671119d142c2f8548882641edfad92863ee1ccff2cd84c3b14a2a 12.11MB / 12.11MB
 => => sha256:119d43eec815e5f9a47da3a7d59454581b1e204b0c34db86f171b7ceb3336533 29.77MB / 29.77MB
 => => extracting sha256:119d43eec815e5f9a47da3a7d59454581b1e204b0c34db86f171b7ceb3336533
 => => extracting sha256:83e2eb8c4c73235e21df3ea0ce6aa840fcb84f4cb368fb6b443a8a87d95850d0
 => => extracting sha256:671677b67e7671119d142c2f8548882641edfad92863ee1ccff2cd84c3b14a2a
 => => extracting sha256:3d6ef8a4ce0aaaaa35261feb115464317fcb4910873cba90171f7ad544a9964e
 => [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt .
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . .
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:08fddb10067b46350bb60831286dde8703364451e94c6d29ff4ad322d51c6d84
 => => exporting config sha256:f7dc0097aa0089d505c6c0a58e222a66d460a45f63a446a9752ae8233ca0b806
 => => exporting attestation manifest sha256:a055ebb8d73dd5c15fa10d42b287dc37128e14057ae96b8757e55ac72136b77b
 => => exporting manifest list sha256:61d39538db71b36abe91b76214285c50a70bb3ba164222c0cc9768fd765a1e10
 => => naming to docker.io/library/student-crud-app:latest
 => => unpacking to docker.io/library/student-crud-app:latest
```

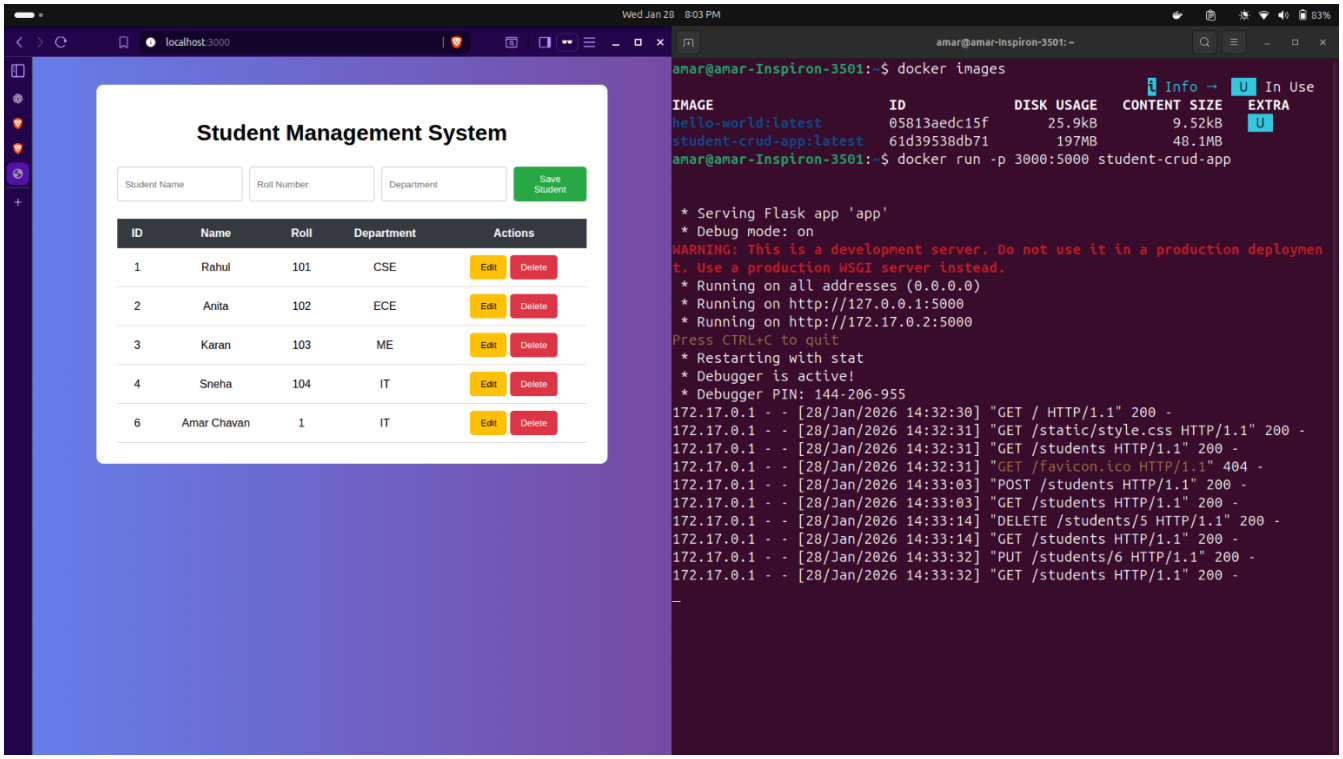
Step 4: Running the Container

The container was started by mapping the host port 3000 to the container port 5000.

docker run -p 3000:5000 student-crud-app

Verification was performed by accessing http://localhost:3000 in the browser, showing the functional Student Management System.

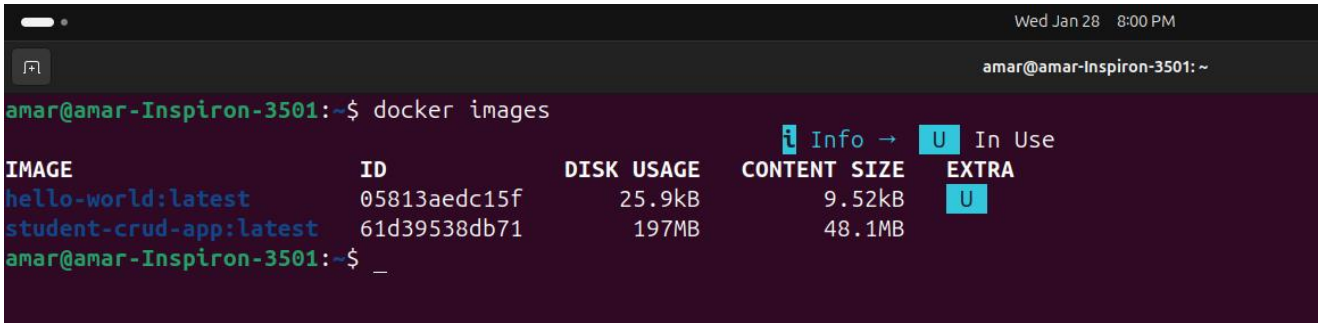


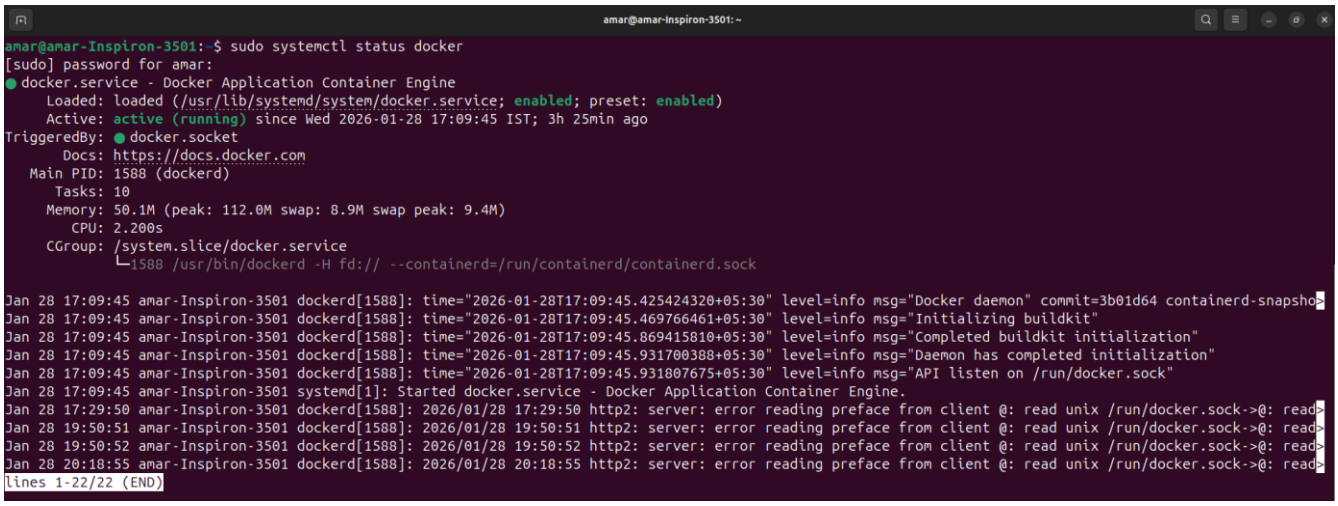


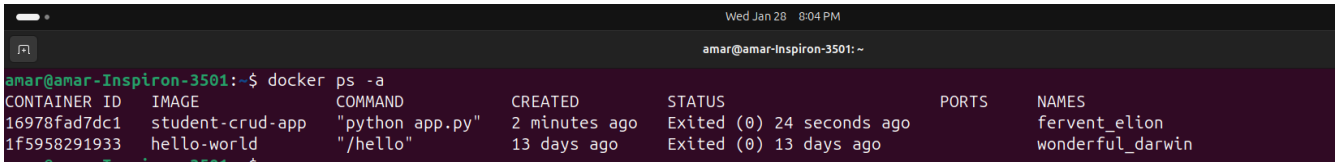
Step 5: Managing Images and Containers

The following commands were used to verify the state of the Docker environment:

- docker images: Confirmed the creation of student-crud-app:latest with a size of approximately 197MB.
- docker ps -a: Showed the status of the container, including its ID and uptime.
- sudo systemctl status docker: Verified that the Docker daemon was active and running.







Conclusion

The successful completion of Assignment 9 provides a comprehensive understanding of Docker's client-server architecture and the practical application of containerization. By developing a Dockerfile for the Student Management System, it was demonstrated how an application, along with its specific dependencies like Flask, can be packaged into a portable and immutable image. The build process highlighted the importance of Layer Caching, which optimizes efficiency by only rebuilding modified parts of the application. Furthermore, the successful verification on localhost:3000 confirmed that Docker ensures environment consistency, allowing the application to run seamlessly across different systems.