

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

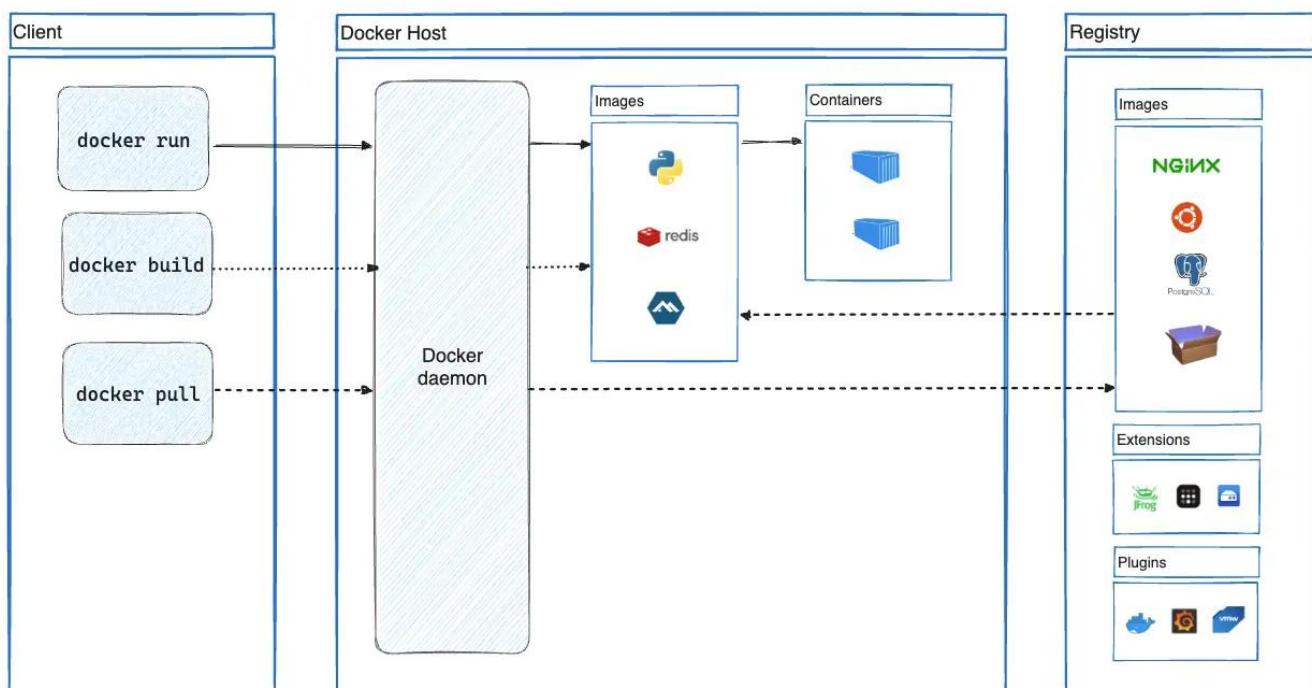
**Aim:** To explore Docker architecture and core commands by containerizing a Python Flask using a Dockerfile.

### 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"]

```

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure: `student-crud` containing `static`, `templates`, `index.html`, `style.css`, `app.py`, and `Dockerfile`.
- Dockerfile:** Content of the Dockerfile is displayed, defining the build process.
- Terminal:** The terminal shows the command: `(venv) amar@amar-Inspiron-3501:~/Desktop/student-crud$`
- Status Bar:** Shows the current file is `Dockerfile`, and other details like line count (14), column count (9), and encoding (UTF-8).

### 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
=> => sha256:119d43eec815e5f9a47da3a7d59454581b1e204b0c34db86f171b7ceb3336533
=> => extracting sha256:83e2eb8c4c73235e21df3ea0ce6aa840fc84f4cb368fb6b443a8a87d95850d0
=> => extracting sha256:671677b67e7671119d142c2f8548882641edfad92863ee1ccff2cd84c3b14a2a
=> => extracting sha256:3d6ef8a4ce0aaaa35261feb115464317fcba91073cba90171f7ad544a9964e
=> [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:f7dc0097aa0089d505c60a58e222a66d460a45f63a446a9752ae8233ca0b806
=> => 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.

The screenshot shows a terminal window and a web browser side-by-side. The terminal window on the right displays the command `docker run -p 3000:5000 student-crud-app` being run, followed by logs indicating the app is serving on port 5000 and listing several incoming HTTP requests. The browser window on the left shows the "Student Management System" application running at `http://localhost:3000`. The application interface includes input fields for Student Name, Roll Number, and Department, and a "Save Student" button. A table lists student records with columns for ID, Name, Roll, Department, and Actions (Edit and Delete buttons). The table contains five entries initially, and later shows an additional entry for "Amar Chavan" with ID 1.

## 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.

```
Wed Jan 28 8:00 PM
amar@amar-Inspiron-3501:~$ docker images
          i Info →   U In Use
IMAGE           ID      DISK USAGE    CONTENT SIZE    EXTRA
hello-world:latest 05813aedc15f    25.9kB       9.52kB
student-crud-app:latest 61d39538db71 197MB        48.1MB
amar@amar-Inspiron-3501:~$ _
```

```
amar@amar-Inspiron-3501:~$ sudo systemctl status docker
[sudo] password for amar:
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
  Active: active (running) since Wed 2026-01-28 17:09:45 IST; 3h 25min ago
TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
  Main PID: 1588 (dockerd)
    Tasks: 10
   Memory: 50.1M (peak: 112.0M swap: 8.9M swap peak: 9.4M)
     CPU: 2.200s
   CGroup: /system.slice/docker.service
           └─1588 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jan 28 17:09:45 amar-Inspiron-3501 dockerd[1588]: time="2026-01-28T17:09:45.425424320+05:30" level=info msg="Docker daemon" commit=3b01d64 containerd-snapshot=2026-01-28T17:09:45.469766461+05:30
Jan 28 17:09:45 amar-Inspiron-3501 dockerd[1588]: time="2026-01-28T17:09:45.469766461+05:30" level=info msg="Initializing buildkit"
Jan 28 17:09:45 amar-Inspiron-3501 dockerd[1588]: time="2026-01-28T17:09:45.869415810+05:30" level=info msg="Completed buildkit initialization"
Jan 28 17:09:45 amar-Inspiron-3501 dockerd[1588]: time="2026-01-28T17:09:45.931700388+05:30" level=info msg="Daemon has completed initialization"
Jan 28 17:09:45 amar-Inspiron-3501 dockerd[1588]: time="2026-01-28T17:09:45.931807675+05:30" level=info msg="API listen on /run/docker.sock"
Jan 28 17:09:45 amar-Inspiron-3501 systemd[1]: Started docker.service - Docker Application Container Engine.
Jan 28 17:29:50 amar-Inspiron-3501 dockerd[1588]: 2026/01/28 17:29:50 http2: server: error reading preface from client @: read unix /run/docker.sock->@: read
Jan 28 19:50:51 amar-Inspiron-3501 dockerd[1588]: 2026/01/28 19:50:51 http2: server: error reading preface from client @: read unix /run/docker.sock->@: read
Jan 28 19:50:52 amar-Inspiron-3501 dockerd[1588]: 2026/01/28 19:50:52 http2: server: error reading preface from client @: read unix /run/docker.sock->@: read
Jan 28 20:18:55 amar-Inspiron-3501 dockerd[1588]: 2026/01/28 20:18:55 http2: server: error reading preface from client @: read unix /run/docker.sock->@: read
Lines 1-22 (END)
```

```
Wed Jan 28 8:04 PM
amar@amar-Inspiron-3501:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
16978fad7dc1 student-crud-app "python app.py" 2 minutes ago Exited (0) 24 seconds ago
1f5958291933 hello-world "/hello" 13 days ago Exited (0) 13 days ago
```

## 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.