## 4.2 DOCKER ARCHITECTURE

**What is Docker?**

- Docker is an **open source software platform** used to **create, deploy and manage applications** in virtualized environments called ***containers***.
- These containers are **lightweight, portable and self-sufficient packages** that include everything an application needs to run, such as code, libraries, runtime and system tools.
- Containers ensure **consistent performance** across different environments by encapsulating the application and its dependencies.
- Docker gives software developers a faster and more efficient way to **build and test containerized portions** of an overall software application.
- This lets developers in a team concurrently build multiple pieces of software.
- Each container includes all elements needed to create a software component and ensure it's **built, tested and deployed smoothly**.
- Docker enables **portability** so these packaged containers can be moved to **different servers or development environments.**

**Advantages of Docker:**

**Ease of Use:**

- ✓ Docker makes it **easy to create, deploy, and manage applications using containers**.
- ✓ Its **simple interface** and **user-friendly command-line tools** help developers **learn and integrate Docker quickly into their workflows**.

**Portability:**

- ✓ Applications run **consistently across environments — local machines, cloud platforms, and on-premises servers**.

**Consistency:**

- ✓ Ensures **identical behavior** in **development, testing, and production** by packaging **application code along with its dependencies**.

**Lightweight:**

- ✓ Containers do not require a **full operating system** for each application; they **share the host OS kernel**, reducing **overhead**.

**Scalability:**

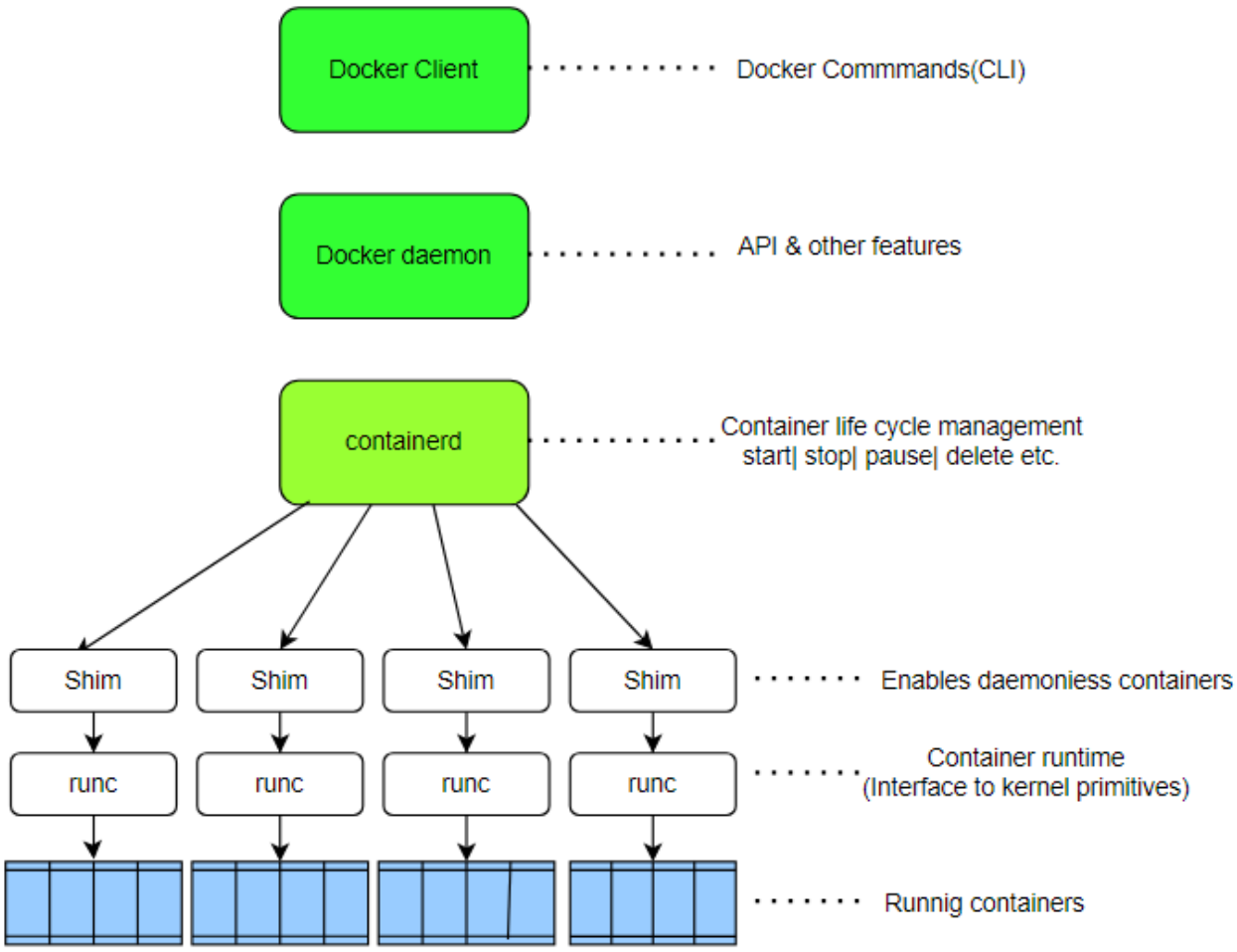
- ✓ Well-suited for **microservices architecture** and container orchestration tools such as **Kubernetes** and **Docker Swarm**.

**Efficiency:**

- ✓ Containers **start within seconds** and consume **fewer system resources** compared to **virtual machines**.

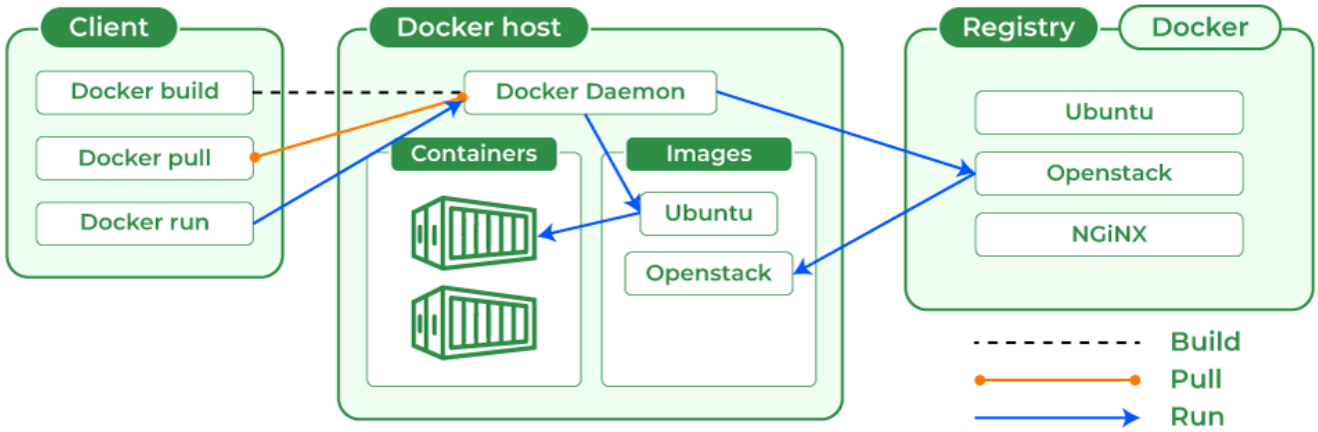
**Docker Architecture:**

- ➢ Docker is based on a **client–server model**.
- ➢ The **Docker client** sends **requests to the Docker Daemon**.
- ➢ The Docker Daemon **handles container lifecycle tasks**.
- ➢ **Communication** happens over a **REST API using sockets or networks.**




1. **Docker Client:**
   - ✓ The Docker Client is the primary interface for users.
   - ✓ When you execute commands such as docker run or docker build, the client translates them into REST API requests and sends them to the Docker Daemon.
2. **Docker Host:**
   - ✓ This is the machine where the magic happens.
   - ✓ It runs the Docker Daemon (dockerd) and provides the environment to execute and run containers.

3. **Docker Registry:**
   - ✓ This is a remote repository for storing and distributing your Docker images.




**Core Components are -**

1. **The Docker Daemon (dockerd)**:
   - ✓ The Docker Daemon is the persistent background process that **acts as the brain** of your Docker installation.
   - ✓ It runs on the **Docker Host**.
   - ✓ It listens for API requests from the Docker Client.
   - ✓ It manages all **Docker objects** such as images, containers, networks, and volumes.
   - ✓ It can communicate with other daemons to manage Docker services in a multi-host environment (like a Docker Swarm cluster.
   - ✓ A **Docker Swarm cluster** is a group of Docker **hosts (machines)** that are joined together to work as a single **virtual Docker engine)**.
   - ✓ It allows you to **deploy and manage containers across multiple machines** efficiently.
2. **The Docker Client**:
   - ✓ The Docker Client is the primary interface through which users interact with Docker. This is most commonly the Command Line Interface (CLI).
   - ✓ It translates user commands like docker ps into REST API requests.
   - ✓ These requests are sent to the Docker Daemon for processing.
   - ✓ A single client can communicate with multiple daemons.

**Common Commands:**

- ✓ **docker build**: Builds an image from a Dockerfile.

- ✓ **docker pull**: Pulls an image from a registry.
- ✓ **docker run**: Creates and starts a container from an image.

3. **The Docker Host**

The Docker Host is the physical or virtual machine that provides the complete environment for executing and running containers. It comprises:

- ✓ The Operating System (and its kernel).
- ✓ The Docker Daemon.
- ✓ Images that have been pulled or built.
- ✓ Running Containers.
- ✓ Networks and Storage components.

The Docker Host is the machine that provides the environment to run Docker, including its containers, images, networks, and storage resources**.**

**Images:**

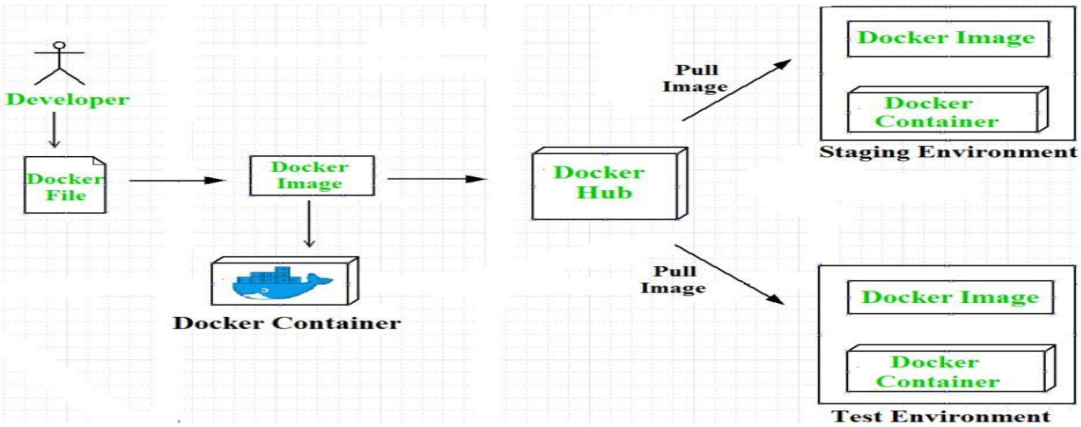
- A Docker image is a read-only template used to create containers.
- It contains the instructions for creating a Docker container.
- It contains all the necessary code, libraries, configuration files, and environment settings required for an application to run.
- That contains the instructions for creating a Docker container.
- It act as a blueprint or a class in object-oriented programming for creating docker containers.
- It provide consistency and portability across different environments. Images are created using DOckerfiles, stored in registries like Docker hub, and can be easily pulled, pushed and shared.
- Docker images are the read-only binary templates used to create a Docker container.
- It's built from a Dockerfile, a simple text file defining the steps to assemble the image.
- Images are built in layers, where each instruction in the Dockerfile corresponds to a layer. This layered architecture makes builds and distribution incredibly efficient.
- Images are **immutable**, meaning once created, they cannot be changed. Any change results in a new image.
- Images are built in layers. Each instruction in a Dockerfile creates a new

layer.

- Images can be shared through registries, allowing applications to run consistently across different systems.

**Containers:**

- ✓ A container is a runnable, live instance of an image.
- ✓ If an image is the blueprint, a container is the house built from that blueprint.
- ✓ It is portable box for software applications.
- ✓ You can create, start, stop, move, or delete containers using the Docker API or CLI.
- ✓ Each container is isolated from other containers and the host machine, having its own filesystem, networking, and process space.
- ✓ You can run multiple containers from the same image.
- ✓ Containers are lightweight because they share the host system's kernel.
- ✓ Containers can run on any system where Docker is installed, making them portable.




**Registry:**

- A registry is a scalable storage system for Docker images.
- Registries allow developers and organizations to store, share, and distribute images.
- Storage systems for images, enabling sharing and distribution.

**Types of registries:**

**Public Registry:**

- ✓ The default public registry is **Docker Hub**, which contains a vast collection of community and official images.

- ✓ Example – Docker Hub, where anyone can publish or download images.

**Private Registries:**

- ✓ Organizations often use private registries (like Harbor, AWS ECR, or Google Artifact Registry) to store proprietary images for **security** and control.
- ✓ Used within organizations to store images securely.

Registries make it easy to **store, version, and distribute images**, make application deployment faster by allowing images to be pulled directly into a host system.