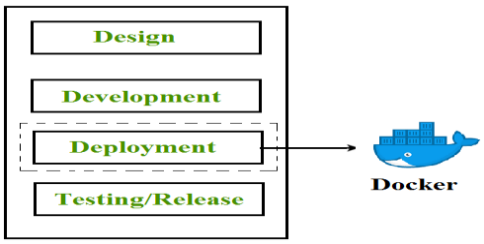
4.2 DOCKER ARCHITECTURE

Docker is an open-source **platform for developing, shipping, and running applications** inside **lightweight, portable containers**. Containers package the **application code, runtime, libraries, and dependencies** together, ensuring the application runs consistently across different environments.

Example:

- You have a **Python program** that needs Python 3.10 and some libraries.
- On your computer, you may have a different Python version installed.
- Using **Docker**, you can put your Python program and the correct Python version in a **container**, and it will run the same way on **any computer** without conflicts.

Docker follows a **client-server architecture**. The Docker client communicates with a background process, the Docker Daemon, which does the heavy lifting of building, running, and managing your containers.

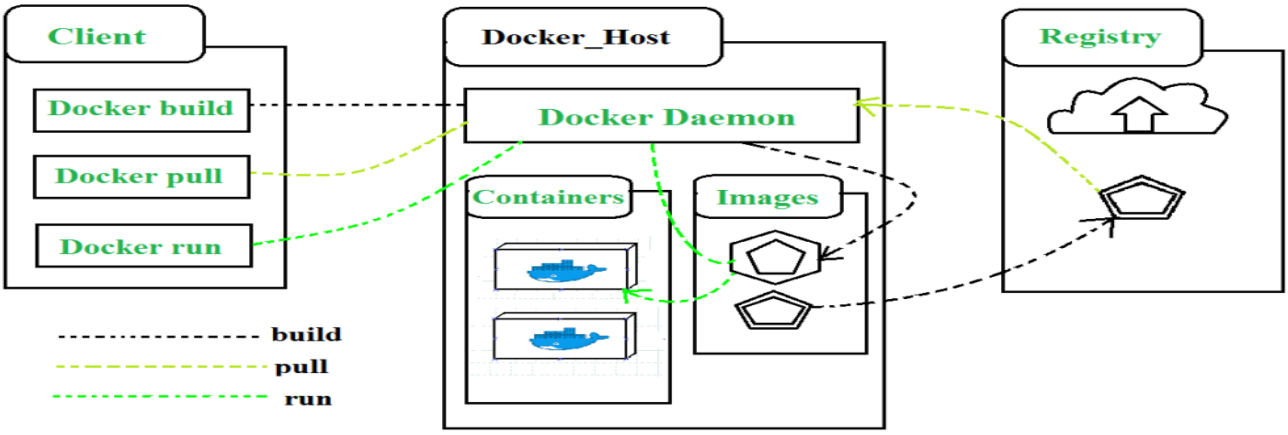


This communication happens over a REST API, typically via UNIX sockets on Linux (e.g., /var/run/docker.sock) or a network interface for remote management.

The Core Architectural Model

- **Docker Client:** This is your **command center**. When you **type commands** like docker run or docker build, you're using the Docker Client.

- **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.
- **Docker Registry:** This is a remote repository for storing and distributing your Docker images. This interaction forms a simple yet powerful loop: you use the **Client** to issue commands to the **Daemon** on the **Host**, which can pull images from a **Registry** to run as containers. In Docker the meaning of push is upload and pull means download.



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**.**

4.2.1. 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.

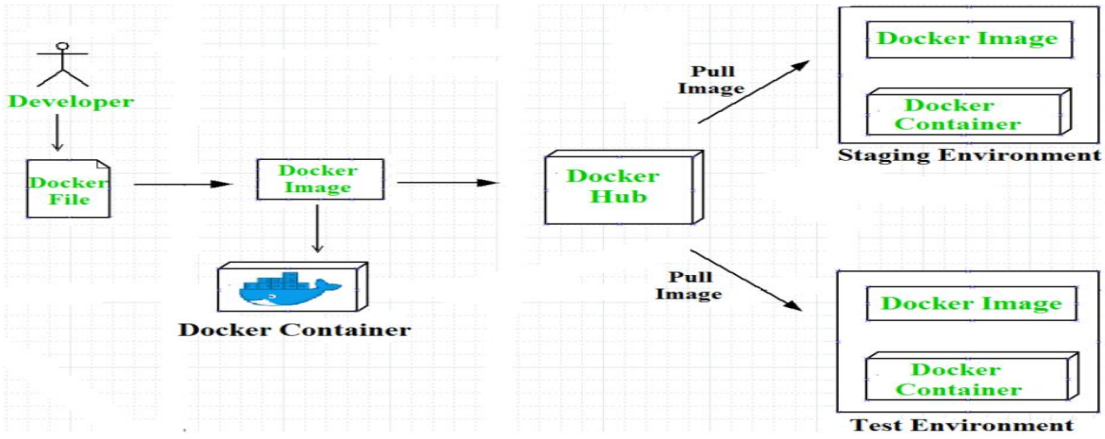
- Docker images are the read-only binary templates used to create a Docker container.
- It's built from a Dockerfile, a simple text file defining the steps to assemble the image.
- Images are built in layers, where each instruction in the Dockerfile corresponds to a layer. This layered architecture makes 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.

4.2.2. 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.



4.2.3. 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.