## Docker Architecture

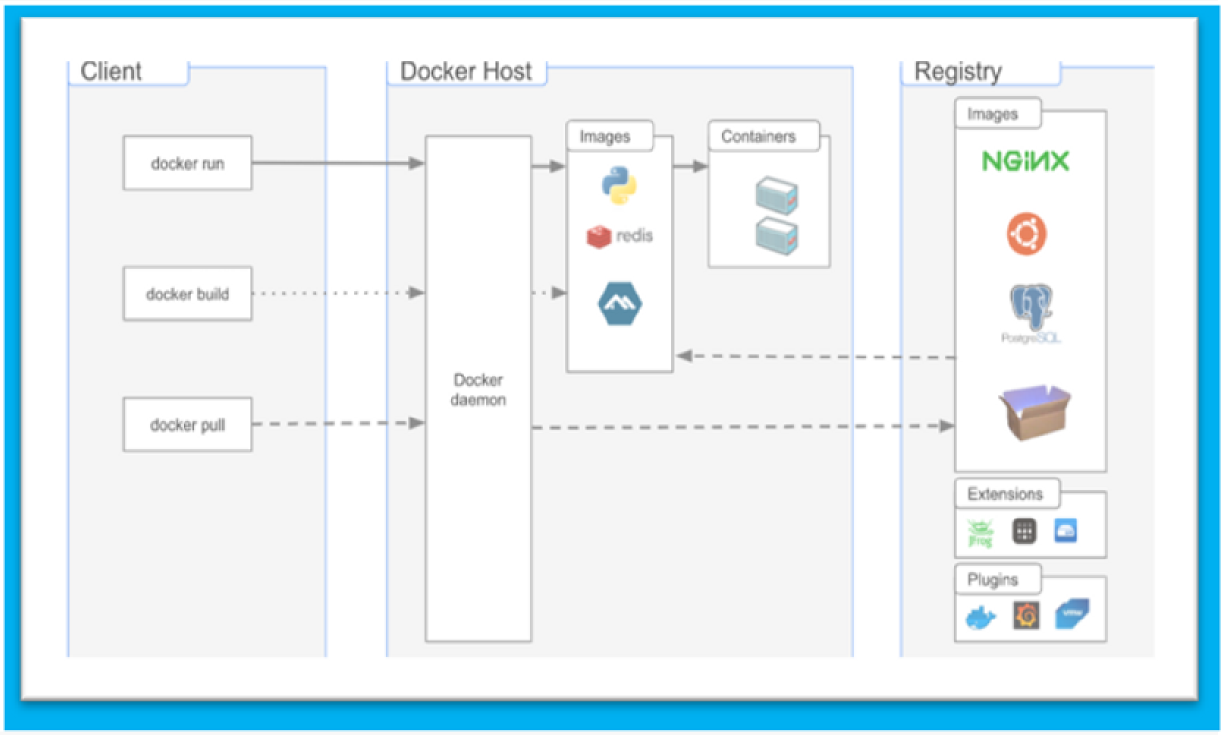
Docker makes use of a client-server architecture. The Docker client talk with the docker daemon which helps in building, running, and distributing the docker containers. The Docker client run with the daemon on the same system or we can connect the docker client with the docker daemon remotely. With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other.

**1. Docker Client**

**2. Docker Host**

**3. Network and Storage components**

**4. Docker Registry / Hub**



## 1)The Docker Client

The Docker client enables users to interact with Docker. The Docker client can reside on the same host as the daemon or connect to a daemon on a remote host. A docker

client can communicate with more than one daemon. The Docker client provides a command line interface (CLI) that allows you to issue build, run, and stop application commands to a Docker daemon.
The main purpose of the Docker Client is to provide a means to direct the pull of images from a registry and to have it run on a Docker host. Common commands issued by a client are:
docker build
docker pull
docker run

## 2)Docker Host

A Docker host is a type of machine which is responsible for running more than one container.
It comprises of the
**i. Docker Daemon**
**ii. Images**
**iii. Containers**

### i) Docker Daemon

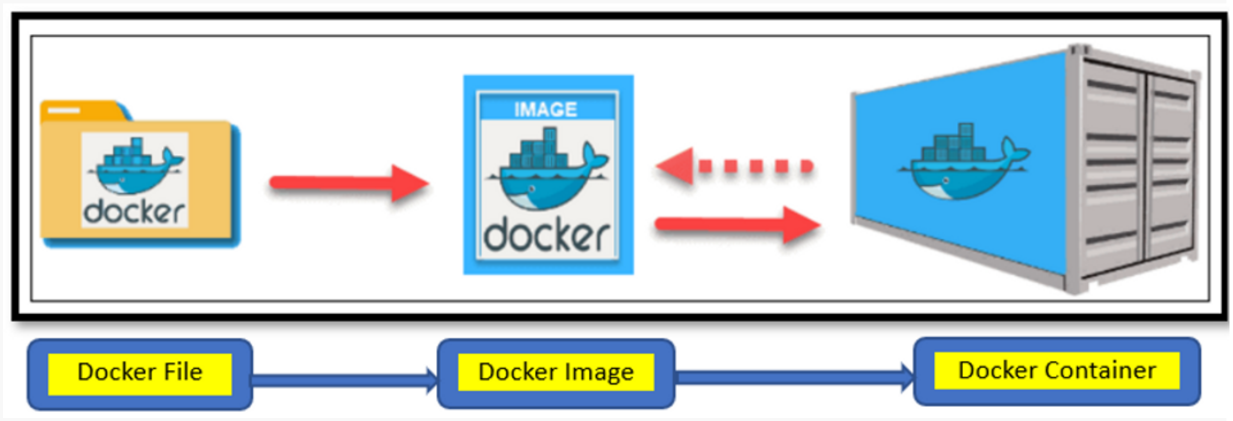
The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

A daemon can also communicate with other daemons to manage Docker services. It is a service that runs in the background that listens for Docker Engine API requests and manages Docker objects like images and containers. **Docker Engine API** is a RESTful API that's used to interact with Docker daemon.

### ii)Docker Image

A Docker Image is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run. An image contains instructions for creating a docker container. It is just a read-only template. It is used to store and ship applications. Images are an important part of the docker experience as they enable collaboration between developers in any way which is not possible earlier.

A Docker File is a script that consists of a set of instructions on how to build a Docker image. These instructions include specifying the operating system, languages, Docker environment variables, file locations, network ports, and other components needed to run the image. All the commands in the file are grouped and executed automatically.



### iii) Docker Containers

Docker containers are lightweight virtualized runtime environments for running applications. They are independent and isolated from the host and other instances running on the host.

Containers are abstractions of the app layer. Each container represents a package of software that contains code, system tools, runtime, libraries, dependencies, and configuration files required for running a specific application. This makes it possible for multiple containers to run in the same host, so you can use that host's resources more efficiently.

Each container runs as an isolated process in the user space and take up less space than regular VMs due to their layered architecture.

## 3) Network and Storage components

### Docker Storage

We can store data within the writable layer of the container but it requires a storage driver. Storage driver controls and manages the images and containers on our docker host. Keeping the persistent storage as a reference, docker offer 4 options:-

Data Volumes

Volume Container

Directory Mounts

Storage Plugins

### Docker Networking

Docker networking provides complete isolation for docker containers. It means a user can link a docker container to many networks. It requires very less OS instances to run the workload.

There are primarily 5 network drivers in docker:

> **Bridge:** It is the default network driver. We can use this when different containers communicate with the same docker host.
> **Host:** When you don't need any isolation between the container and host then it is used.
> **Overlay:** For communication with each other, it will enable the swarm services.
> **None:** It disables all networking.
> **macvlan:** This network assigns MAC(Media Access control) address to the containers which look like a physical address.

## 4) Docker Registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

## Advanced Docker Components

· **Docker-compose** is designed for running multiple containers as a single service. It does so by running each container in isolation but allowing the containers to interact with one another. You would write the compose environments using YAML.
· **Docker Swarm** is a clustering and scheduling tool for Docker containers. At the front end, it uses the Docker API, which gives us a means of controlling it with various tools. It is composed of many engines that can both be self-organizing or pluggable depending on what requirements they're trying to meet at any given point in time. Swarm enables you to use a variety of backends.