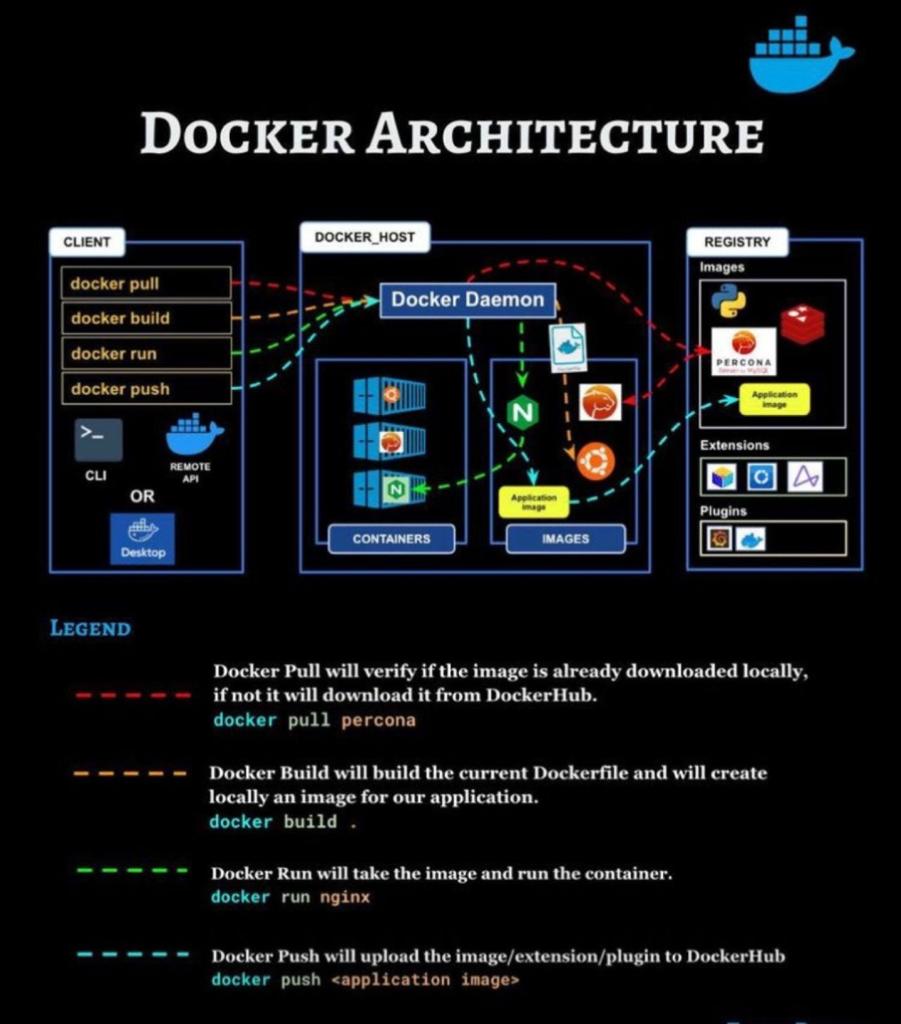
Docker Architecture

* **Docker is an open-source platform** that allows you to automate the deployment, scaling, and management of applications using containerization.
* It provides a way to package an application and its dependencies into a standardized unit called a container.
* Docker architecture consists of several components that work together to create and manage these containers. **Here's an overview of the Docker architecture:**



**Docker Engine:** At the core of Docker is the Docker Engine, which is responsible for building, running, and managing containers. **It consists of three main components**:

**Docker daemon**: Also known as dockerd, it runs on the host operating system and manages container-related operations. It receives commands from the Docker client and interacts with the host's kernel to create and manage containers.

**REST API**: The Docker daemon exposes a RESTful API that allows users to interact with it. The API enables users to manage containers, images, networks, and other Docker resources programmatically.

**Command-line interface (CLI):** The Docker CLI is a command-line tool that serves as the primary interface for users to interact with Docker. It communicates with the Docker daemon through the REST API, allowing users to execute various Docker commands.

**Images**: An image is a read-only template that contains the application's code, runtime, libraries, and dependencies. It serves as the basis for creating Docker containers. Images are typically built using a Dockerfile, which specifies the instructions to create the image. Docker images are stored in a registry, such as Docker Hub or a private registry.

**Containers**: A container is a lightweight, isolated, and portable runtime environment that encapsulates an application and its dependencies. It is created from an image and can be run, started, stopped, moved, and deleted. Containers provide process isolation, meaning each container runs as a separate process and has its own file system, network interface, and resources. Multiple containers can run on the same host, sharing the host's kernel.

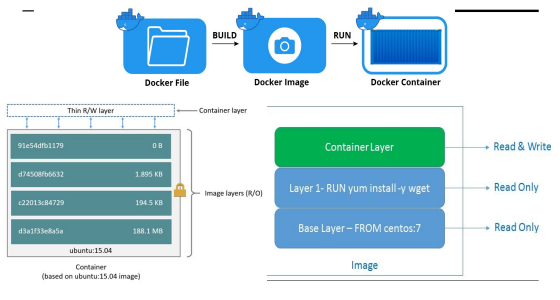
**Registries**: Registries are repositories that store Docker images. The most popular registry is Docker Hub, a public registry that hosts a vast collection of pre-built Docker images. Organizations can also set up their private registries to store and distribute their custom images securely.

**Docker Compose:** Docker Compose is a tool used for defining and managing multi-container applications. It allows you to describe a complex application setup using a YAML file, which includes information about the services, networks, and volumes required. Docker Compose simplifies the process of running interconnected containers and managing their configurations.

**What is a container?**

A container is a sandboxed process on your machine that is isolated from all other processes on the host machine. That isolation leverages [kernel namespaces and cgroups](https://medium.com/@saschagrunert/demystifying-containers-part-i-kernel-space-2c53d6979504), features that have been in Linux for a long time. Docker has worked to make these capabilities approachable and easy to use. To summarize, a container:

* Is a runnable instance of an image. You can create, start, stop, move, or delete a container using the DockerAPI or CLI.
* Can be run on local machines, virtual machines or deployed to the cloud.
* Is portable (can be run on any OS).
* Is isolated from other containers and runs its own software, binaries, and configurations.



**What is kernel in docker**

The kernel is the core component of an operating system that manages low-level system resources, such as memory, CPU, disk I/O, and network access. It acts as an intermediary between applications and the hardware. Docker leverages a feature in Linux called "containerization" to isolate and manage application processes in containers.

Docker containers share the host system's kernel, but they have their own isolated user spaces. This isolation is achieved using various Linux kernel features, such as namespaces, control groups (cgroups), and file system isolation. These features allow Docker containers to run independently and securely on a shared host, without interfering with each other or the host system.

When you create and run a Docker container, it uses the host system's kernel to execute its processes. Docker provides a lightweight runtime environment for applications by leveraging the host kernel, which eliminates the need for running a full-fledged operating system for each container. This approach enables efficient resource utilization and faster container startup times.

**What is namespace in docker**

In Docker, a **namespace** is a feature provided by the Linux kernel that isolates and separates various system resources, such as processes, network interfaces, file systems, and more. Namespaces ensure that processes running within a namespace can only see and interact with resources that belong to that specific namespace, while remaining unaware of resources outside of it.

Docker utilizes multiple types of namespaces to achieve isolation between containers and the host system. Here are some common namespaces used by Docker:

PID, Network, Mount, UTS, IPC Namespaces.

**What is Cgroups in Docker**

In Docker, **cgroups**, short for control groups, is a Linux kernel feature used for resource management and isolation. It allows you to limit and allocate system resources, such as CPU, memory, disk I/O, and network bandwidth, among different processes or groups of processes running on the same host.

Docker leverages **cgroups** to provide resource isolation for containers. Each Docker container created on a host has its own cgroup hierarchy that defines the resource constraints and limits for that container. By using cgroups, Docker can ensure that containers have a fair share of resources and prevent a single container from monopolizing the resources of the host machine.

**Cgroups** enable you to control and fine-tune the allocation of resources to containers. You can set limits on CPU usage, memory consumption, and other resources, allowing you to effectively manage and prioritize the allocation of resources among different containers running on the same host. This helps in achieving better performance, isolation, and stability of the overall system.