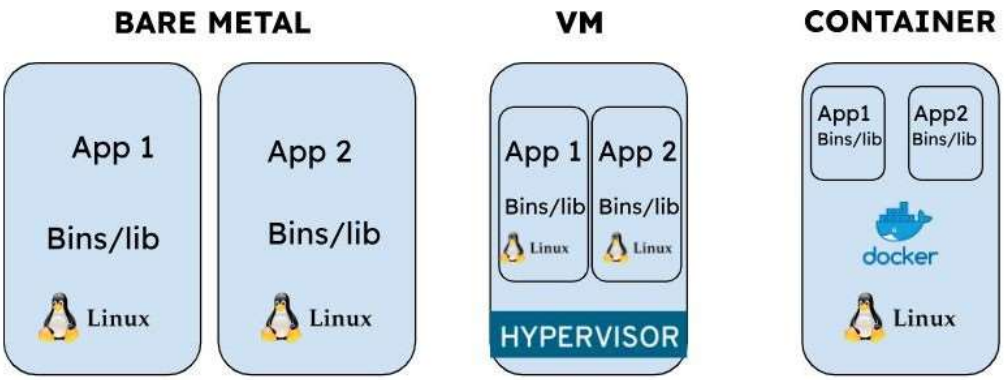
What is Docker?

Docker is a containerization platform that provides a complete system for building and running software containers. Containers package applications and their dependencies as ephemeral units that behave similarly to virtual machines but share your host's operating system kernel. Running apps in containers makes them more portable by letting you deploy anywhere Docker is available. Docker containers provide an eUicient way to package, distribute, and run applications on any infrastructure, from on-premises data centers to public clouds.Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through

well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.

Docker is a big deal because historically, the only practical ways for most teams to deploy and run applications was to operate them inside virtual machines (VMs) or on bare-metal servers – both of which are inferior to Docker in many respects.



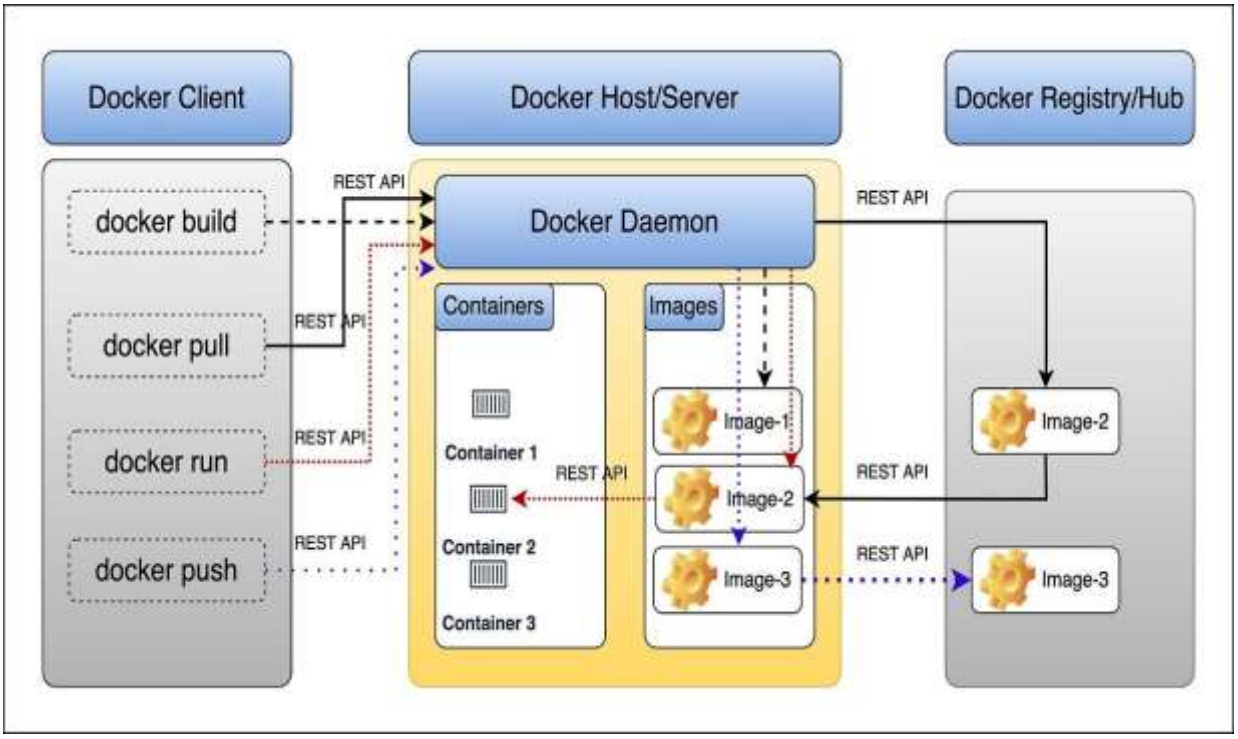
Compared to VMs (which provide more flexibility than bare-metal servers), Docker uses resources more eUiciently because it shares many resources with the host operating system instead of running a standalone guest operating system. And compared to bare-metal servers (which are more eUicient from a resource-consumption perspective than VMs), Docker provides more flexibility and isolation between applications because it allows each application to run inside its own lightweight environment rather than requiring all apps to sit alongside each

other on a shared physical server.

In this way, Docker provides access to the best of both worlds: the flexibility and agility of VMs on the one hand, and the eUiciency of baremetal servers on the other. It's worth noting that Docker containers can run on top of either VMs or bare-metal servers (as we explain below in more detail), so Docker isn't an alternative to VMs and bare-metal as much as it's a complement to them. But by using Docker, engineers get more choice and flexibility than they would if they relied on VMs or baremetal servers alone to host their applications.

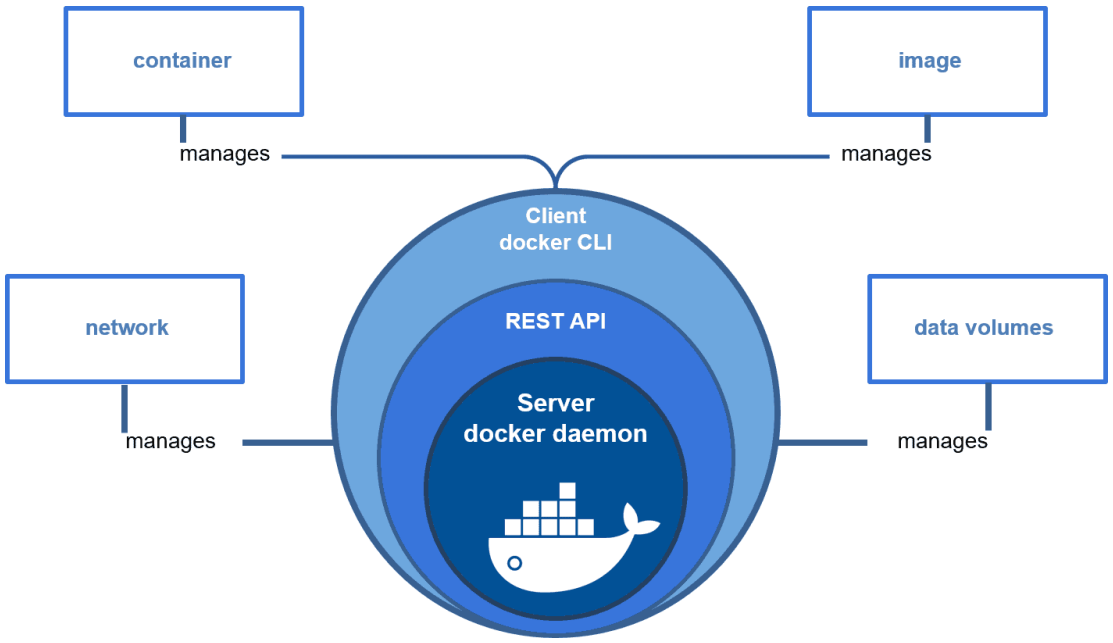
Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers



How does Docker architecture work?

Docker uses a client-server architecture and depends on several distinct components to handle low-level container interactions. As a result, installing Docker Engine on your host will include multiple software packages.



1. The Docker Engine daemon, docked, to provide the API.
2. The docker CLI that you use to interact with the daemon.
3. The container runtime that manages containers at the host level.
4. A system service that automatically starts the Docker daemon and your containers after host reboots.

5. Prepopulated config files that ensure the Docker CLI can connect to your daemon instance.

6. The docker compose tool that lets you build multi-container applications.

What is Docker Daemon?

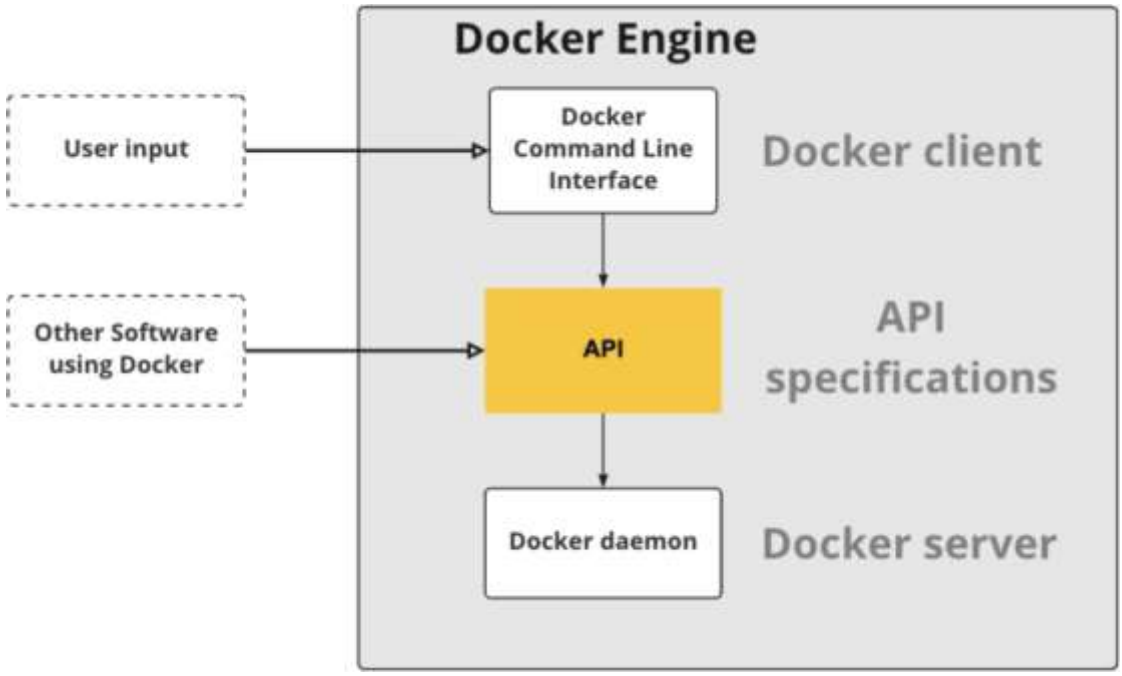
Docker daemon manages all the services by communicating with other daemons. It manages docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker.

Docker Engines

Docker Engine is an open-source containerization platform that automates the creation, deployment, and management of containerized applications. It includes a daemon (dockerd) that manages containers, images, and networks, and a command-line interface (CLI) for user interactions. Containers run on a shared operating system, providing isolated environments for applications, which ensures consistency across various stages of development and production.

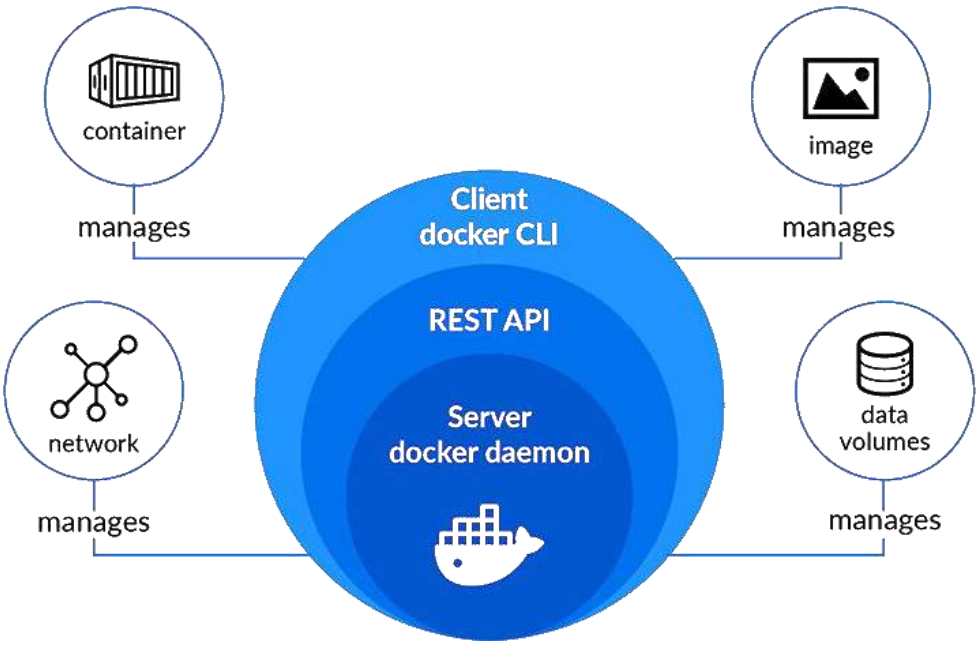
The engine uses a layered file system and copy-on-write to optimize image storage, enabling efficient use of disk space and fast deployments. Docker Engine supports both Linux and Windows, allowing cross-platform compatibility. It integrates with orchestration tools like Docker Swarm and Kubernetes, facilitating the management of container clusters

Key features include REST API for remote interaction, multi-host networking, and pluggable storage drivers. Docker Engine's portability, efficiency, and ecosystem of tools make it a cornerstone of modern DevOps, promoting continuous integration and continuous deployment (CI/CD) pipelines.



Docker Client

With the help of the docker client, the docker users can interact with the docker. The docker command uses the Docker API. The Docker client can communicate with multiple daemons. When a docker client runs any docker command on the docker terminal then the terminal sends instructions to the daemon. The Docker daemon gets those instructions from the docker client withinside the shape of the command and REST API's request.



The main objective of the docker client is to provide a way to direct the pull of images from the docker registry and run them on the docker host. The common commands which are used by clients are docker build, docker pull, and docker run.

Docker Host

A Docker host is a type of machine that is responsible for running more than one container. It comprises the Docker daemon, Images, Containers, Networks, and Storage.

Docker Registry

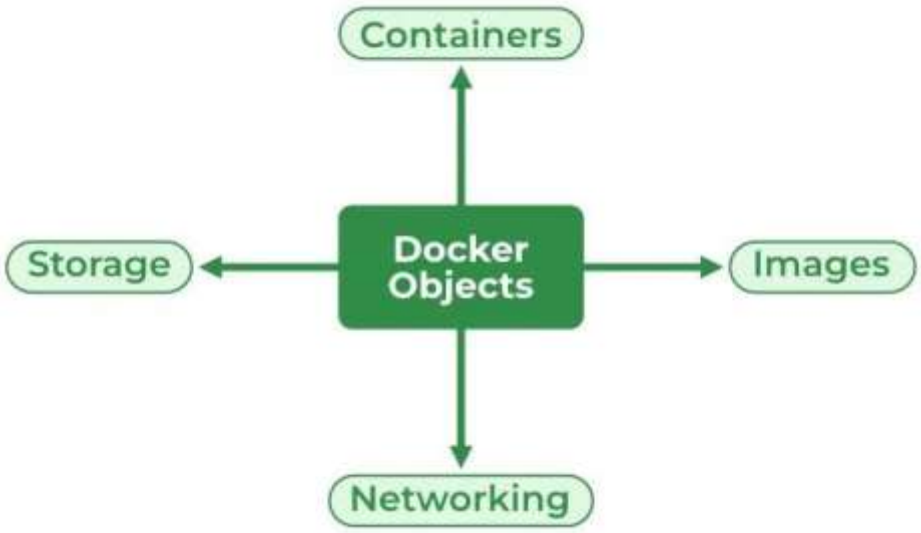
All the docker images are stored in the docker registry. There is a public registry which is known as a docker hub that can be used by anyone. We can run our private registry also. With the help of docker run or docker pull commands, we can pull the required images from our configured registry. Images are pushed into configured registry with the help of the docker push command.

Docker Containers

Containers are created from docker images as they are ready applications. With the help of Docker API or CLI, we can start, stop, delete, or move a container. A container can access only those resources which are defined in the image unless additional access is defined during the building of an image in the container.

Docker Objects

Whenever we are using a docker, we are creating and use images, containers, volumes, networks, and other objects. Now, we are going to discuss docker objects.



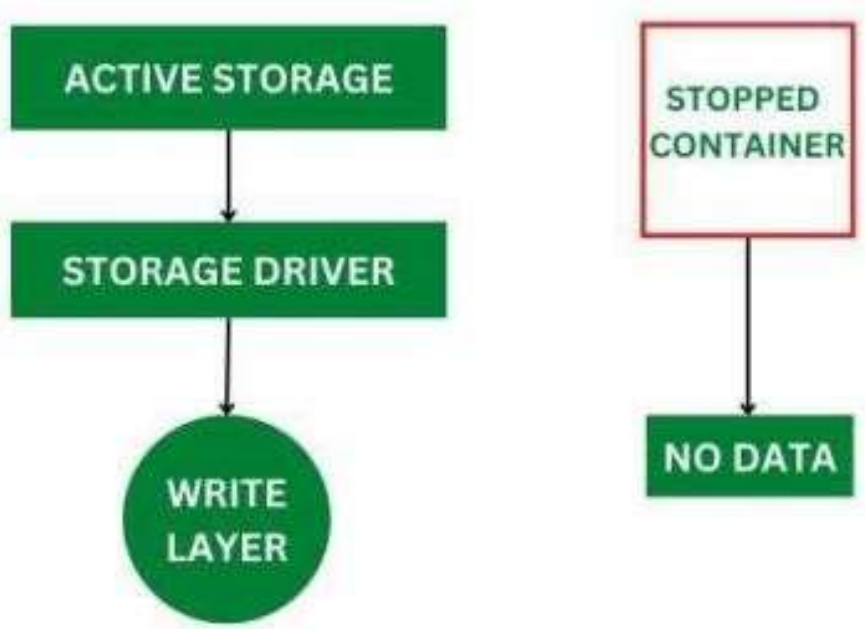
Docker Images

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.

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.



Types of Docker Storage

1. **Data Volumes:** Data Volumes can be mounted directly into the filesystem of the container and are essentially directories or files on the Docker Host filesystem.

2. **Volume Container:** To maintain the state of the containers (data) produced by the running container, Docker volumes file systems are mounted on Docker containers. independent container life cycle, the volumes are stored on the host. This makes it simple for users to exchange file systems among containers and backup data.

3. **Directory Mounts:** A host directory that is mounted as a volume in your container might be specified.

4. **Storage Plugins:** Docker volume plugins enable us to integrate the Docker containers with external volumes like Amazon EBS by this we can maintain the state of the container.

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.

Types of Docker Network

1. **Bridge:** It is the default network driver. We can use this when different containers communicate with the same docker host.

2. **Host:** When you don't need any isolation between the container and host then it is used.

3. **Overlay:** For communication with each other, it will enable the swarm services.

4. **None:** It disables all networking.

5. **macvlan:** This network assigns MAC (Media Access control) address to the containers which look like a physical address.

Benefits of the Docker architecture

Docker's client-server, daemon-based architecture places an API between docker commands and their eUects. This can seem like it's adding complexity, but it unlocks a few useful benefits for more powerful container workflows.

1. Manage multiple Docker hosts with your local CLI installation You can add remote environments (such as your production servers) as contexts you can switch between when using your local Docker CLI installation.

2. Build images remotely on a higher-powered machine You can add remote contexts to run builds on higher performance hardware or hosts that use a diUerent processor architecture to yours. This can improve
overall DevOps eUiciency.

3. Allow developers to connect to shared environments

Live, staging, and test environments are typically shared between several developers who require their own access to

make deployments and monitor operations. Remote Docker daemon connections let everyone work in an environment, without having to open up host-level access.