**What is Docker?**

**Docker** is an **open-source platform** used to **develop, ship, and run applications inside lightweight containers**.

A **Docker container** is a **standalone, executable package** that includes everything needed to run an application:

* Code, Runtime, Libraries, Configuration files, System tools

Run your app on *any* system without worrying about environment issues.

|  |  |
| --- | --- |
| **Image** | A snapshot of your app + environment (read-only template) |
| **Container** | A running instance of an image (like a lightweight virtual machine) |
| **Dockerfile** | A script to create an image |
| **Docker Hub** | A public registry to share images (like GitHub for Docker) |
| **Docker Engine** | The runtime that runs containers |

**1. Docker Image**

A **Docker image** is a **read-only template** that contains:

* Your application code
* All its dependencies (like libraries, frameworks)
* Runtime (like JDK, Python interpreter)
* OS-level tools and configuration

Think of it like a **snapshot** or **blueprint** of your application environment.

Docker images are built in **layers**, and every command in the Dockerfile creates a new layer. This makes them **efficient** and **cacheable.**

**2. Docker Container**

A **container** is a **running instance** of a Docker image.the image is the template, the container is the actual, live app.

Containers are:

* Isolated (runs independently)
* Lightweight (shares host OS kernel)
* Fast to start/stop

**3. Dockerfile**

A **Dockerfile** is a **text file** that contains a list of instructions to build a Docker image.

It defines the **environment** for your app: what base image to use, what files to copy, what commands to run, etc.

**4. Docker Hub**

**Docker Hub** is a **cloud-based registry** where:

* You can **publish** and **share** Docker images publicly or privately
* It’s the **default registry** used by Docker when pulling images

**5. Docker Engine**

**Docker Engine** is the **core runtime** that runs containers and

* Builds Docker images
* Manages networking, volumes, etc.

It is a **client-server application** made of:

1. **Docker Daemon (dockerd)** – Runs in background and manages containers.
2. **Docker CLI (docker)** – Command-line interface to interact with the daemon.

**Docker Daemon?**

The **Docker daemon** (dockerd) is the **core background service** of Docker that:

* **Build, pull, push, and store Docker images**
* **Run,** Start, stop, remove, restart containers
* **Manages** networks, volumes, and other Docker components

It listens for Docker API requests and handles all the heavy lifting behind the scenes.

### Containers Vs. Virtual Machine

|  |  |  |
| --- | --- | --- |
| **Feature** | **Virtual Machines** | **Docker Containers** |
| **Startup Time** | Slow (minutes) | Fast (seconds or less) |
| **OS per Unit** | Yes (full guest OS per VM) | No (shares host OS kernel) |
| **Resource Usage** | Heavy (RAM + disk + CPU) | Lightweight |
| **Isolation** | Strong (via hypervisor & OS) | Moderate (OS-level isolation) |
| **Performance** | Slower due to full OS | Near-native performance |
| **Size** | Large (GBs) wastage of memory. | Small (MB) No wastage of memory. |
| **Portability** | VM image often OS-specific | Highly portable across systems |
| **Security** | Strong OS-level isolation | Less secure (but improving with namespaces) |
| **Use Case** | Running multiple OSes or legacy apps | Microservices, CI/CD, fast deployment |

### Docker architecture

Docker follows Client-Server architecture, which includes the three main components that are **Docker Client**, **Docker Host**, and **Docker Registry**.

**What is Docker Client?**

The **Docker Client** is the command-line tool (docker) that you use to **interact with the Docker Daemon**.

It sends your commands to the Docker Daemon (via REST API), which then **performs the requested actions** like building images, running containers, or managing networks.

**Note**: Docker Client has an ability to communicate with more than one docker daemon.

**What is Docker Host?**

The **Docker Host** is the **physical or virtual machine** where the **Docker Daemon runs** and manages:

* Docker **containers**
* Docker **images**
* Docker **volumes**
* Docker **networks**

It provides the **execution environment** for your Docker-based applications.

**Docker Registry**

Docker Registry manages and stores Docker images. It’s where you **push (upload)** and **pull (download)** Docker images from.

Think of it like a GitHub for Docker images.

**🔧 How It Works**

* **Developers** build Docker images using a Dockerfile
* The image is **pushed** to a registry
* Other users or systems can **pull** the image from the registry to run containers

### **Docker Storage**

**Docker Storage** refers to **how data is stored and managed** by Docker containers—both **temporary (ephemeral)** and **persistent (long-term)**.

By default, data written inside a container is **lost** when the container is removed. Docker provides storage mechanisms to retain that data outside the container lifecycle.

**1. Volumes (Recommended)**

* **Managed by Docker**
* Stored under /var/lib/docker/volumes/
* Persist even if container is removed(Good for databases, file uploads, logs, etc.)

**2. Bind Mounts**

* Mount a **specific host path** into a container
* Good for **development**, not preferred in production

**Docker networking**

Docker networking allows containers to **communicate with each other**, with the host, and with external networks. Understanding how Docker networking works is essential for building scalable, secure, and efficient containerized applications.

|  |  |
| --- | --- |
| **Bridge:** | Default network for standalone containers. Best for simple apps on a single host. |

docker network create my-custom-net

**Host**: Container shares the host's network stack (no isolation). Best for performance.

docker run --network host my-app

**Docker vs. Kubernetes**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Docker** | **Kubernetes** |
| **What it is** | A **platform** to build, run, and manage containers | A **container orchestration system** to manage many containers |
| **Main Purpose** | Package applications into containers | Deploy, scale, and manage containers across multiple servers |
| **Level** | **Container runtime** (single container/app focus) | **Cluster manager** (manages multiple containers/apps) |
| **Scope** | Focuses on **creating and running containers** | Focuses on **scaling, healing, and managing** containers in clusters |
| **Installation** | Simple, runs on a single machine | Complex, requires cluster setup |
| **Scaling** | Manual or with Docker Swarm | Automatic horizontal scaling supported |
| **Load Balancing** | Basic (via Docker Swarm or custom NGINX setup) | Built-in service load balancing |
| **Self-Healing** | Limited (needs manual handling) | Yes — restarts crashed containers, replaces failed nodes |
| **Storage Management** | Volumes and bind mounts | Supports persistent volumes, volume claims, etc. |
| **Monitoring** | Minimal (external tools needed) | Built-in health checks, metrics APIs |
| **Networking** | Bridge, host, overlay networks (manual setup) | Built-in network policies, pod-to-pod communication |
| **Use Case** | Great for development & lightweight container apps | Ideal for **production** with **many containers** and **high scale** |
| **Example** | Running a single Spring Boot app in a container | Running a full microservices system with 10+ services |