# **Docker architecture**

Docker is a platform for developing, shipping, and running applications inside containers. The architecture of Docker consists of several key components that work together to ensure the effective management of containerized applications. Here is an overview of the Docker architecture:

**1. Docker Client**

The Docker client is the primary way users interact with Docker. When you use commands like docker build, docker pull, or docker run, the client sends these commands to the Docker daemon, which carries them out. The Docker client can communicate with more than one daemon.

**2. Docker Daemon (Docker Engine)**

The Docker daemon (also known as 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.

**3. Docker Objects**

Docker objects are used to describe an application and its dependencies. The primary Docker objects include:

**a. Images**

* **Images** are read-only templates with instructions for creating a Docker container. They can include the application code, libraries, environment variables, and runtime. Images are stored in Docker registries.

**b. Containers**

* **Containers** are runnable instances of Docker images. They are isolated environments where applications can run. Containers are created from Docker images and can be started, stopped, moved, and deleted. Each container is an isolated environment, which includes its own file system, networking, and isolated process tree.

**c. Networks**

* **Networks** allow Docker containers to communicate with each other. Docker provides several network drivers for different use cases, including bridge, host, and overlay networks.

**d. Volumes**

* **Volumes** are used for persistent data storage. They allow data to persist across container restarts and can be shared among multiple containers.

**4. Docker Registries**

Docker registries are repositories for Docker images. Docker Hub is a public registry provided by Docker, Inc., where users can share images. Private registries can also be set up to store images securely within an organization.

**5. Docker Engine Components**

**a. Docker Engine API**

* The Docker Engine API is a REST API used by applications to communicate with the Docker daemon. It provides endpoints for managing Docker objects such as containers, images, networks, and volumes.

**b. Docker CLI**

* The Docker CLI is the command-line interface that users interact with to manage Docker objects. It provides commands for building, running, and managing containers, images, and more.

**c. Docker Compose**

* Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you can use a YAML file to configure your application's services, and then start and manage them with a single command.

**6. Docker Workflow**

The typical Docker workflow involves the following steps:

1. **Develop**: Developers write code and create Dockerfiles to define the application's environment.
2. **Build**: The Docker client sends a build command to the Docker daemon, which creates an image based on the Dockerfile.
3. **Ship**: The built image is pushed to a Docker registry, such as Docker Hub or a private registry.
4. **Run**: The Docker client sends a run command to the Docker daemon, which pulls the image from the registry and creates a container based on it.
5. **Manage**: The Docker daemon manages the lifecycle of containers, including starting, stopping, and scaling them.