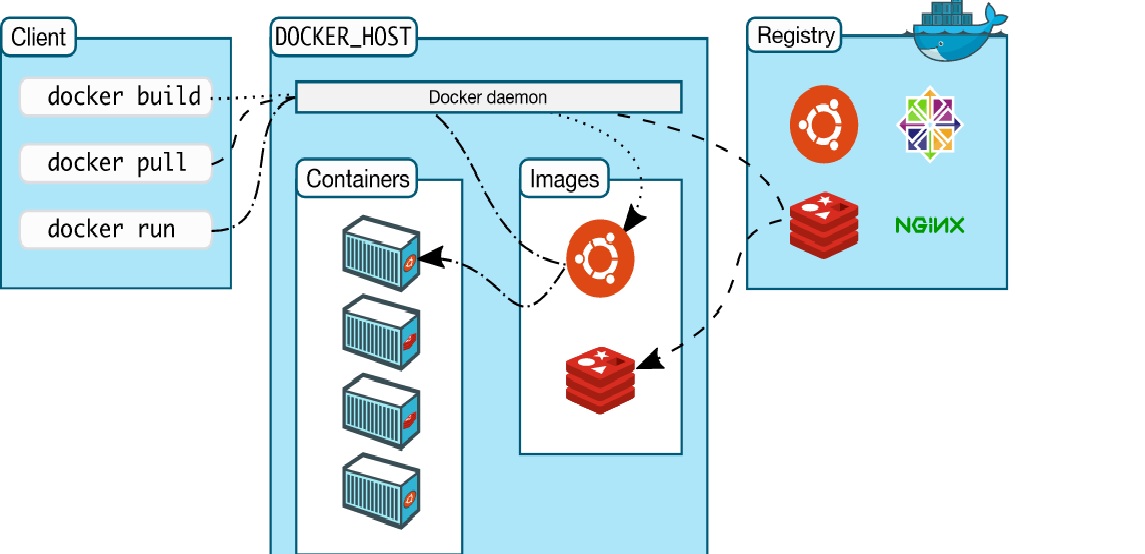
## **Docker architecture**

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.



### The 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.

### The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

### Docker registries

A Docker registry stores Docker images. Docker Hub and Docker Cloud are public registries 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. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).

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.

Docker store allows you to buy and sell Docker images or distribute them for free. For instance, you can buy a Docker image containing an application or service from a software vendor and use the image to deploy the application into your testing, staging, and production environments. You can upgrade the application by pulling the new version of the image and redeploying the containers.

### Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

#### **Images**

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

#### **Containers**

A container is a runnable instance of an image. You can create, run, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

**Docker Image= OS(Linux) +ApacheHTTP +TomcatServer+Java+OracelDB**

**Container= Runtime of Images**

**Technology Used in Docker**

Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality.

### Namespaces

Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

* **The pid namespace:** Process isolation (PID: Process ID).
* **The net namespace:** Managing network interfaces (NET: Networking).
* **The ipc namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
* **The mnt namespace:** Managing filesystem mount points (MNT: Mount).
* **The uts namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

### Control groups

Docker Engine on Linux also relies on another technology called control groups (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

### Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

### Container format

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones