Docker — Complete Introduction Tutorial

---

1. What is Docker?

Definition

Docker is an **open-source platform** designed to **build, package, ship, and run applications** in **containers**.

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

- Code
- Runtime
- Libraries
- System tools
- Configuration

This ensures that an app runs **the same way everywhere** — on your laptop, a server, or in the cloud.

---

2. Why Do We Need Docker?

Before Docker (Traditional Deployment)

- You'd install your app + dependencies directly on the OS.
- Different apps might require **different library versions** (conflicts).
- Deploying the same app on another machine often required **manual setup** again.

With Docker

- Each app runs in its **own container**, isolated from others.
- Containers include all dependencies, so the environment is **consistent** everywhere.
- You can start, stop, copy, or remove containers easily.

---

3. Key Concepts and Components

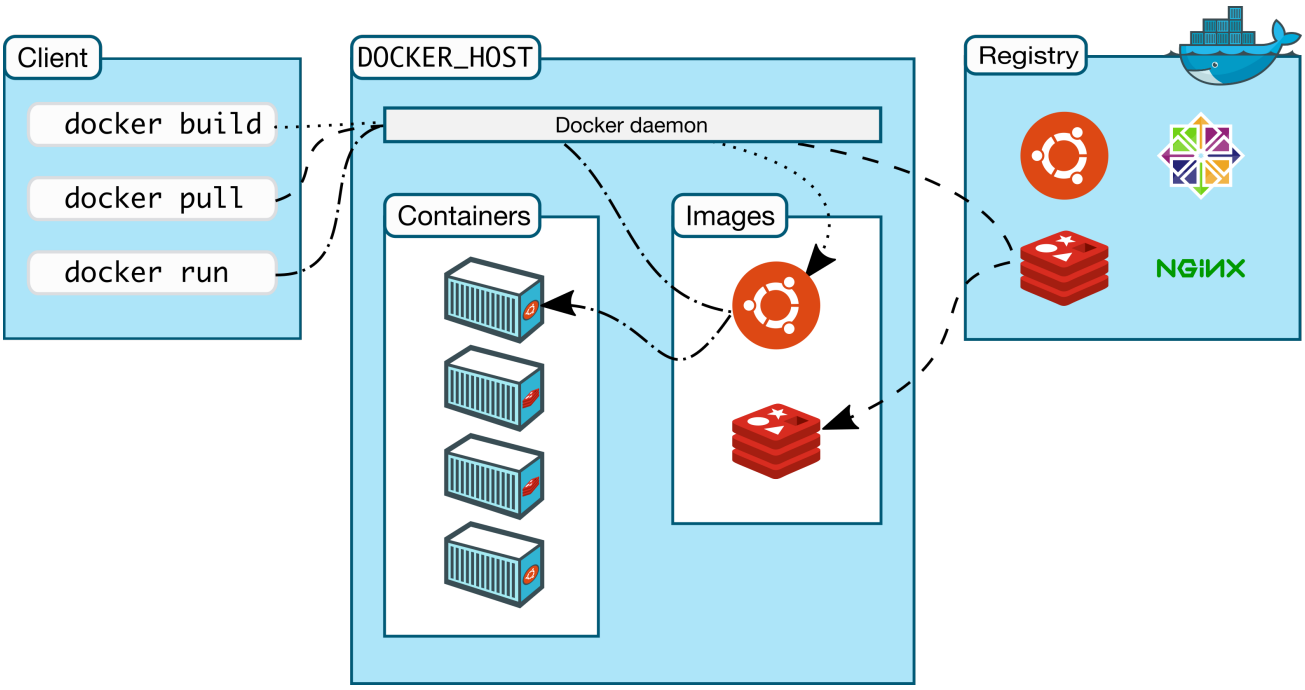
Concept	Description
Image	A read-only blueprint that defines what a container is. Built from a **Dockerfile**.
Container	A running instance of an image. You can create, start, stop, or delete containers.

Dockerfile	A text file with instructions to build an image (e.g., which OS, dependencies, commands).
Docker Engine	The runtime that builds and runs containers.
Docker Hub	Public repository of images (like GitHub for code).
Docker Compose	Tool for defining and running multi-container applications (via `docker-compose.yml`).

---

4. Docker Architecture

```
+-----------------------------------------+
|              Docker CLI (Client)        |
+-----------------------------------------+
                  ↓  talks to
+-----------------------------------------+
|           Docker Daemon (Server)        |
|  - Builds, runs, and manages containers |
+-----------------------------------------+
                  ↓
+-----------------------------------------+
|         Container Runtime Layer         |
|   - Images                              |
|   - Containers                          |
+-----------------------------------------+
                  ↓
+-----------------------------------------+
|             Host Operating System       |
+-----------------------------------------+
```



Reference: medium.com

Client-Server Model

- `docker` command (CLI) talks to the **Docker Daemon** (background service).
- The daemon does the heavy lifting (build, pull, run images).

---

5. Installing Docker

- Refer

  1. Docker documentation

     https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script

     search for convenience script

     curl -fsSL https://get.docker.com -o get-docker.sh

     sudo sh get-docker.sh

  After installation:

  ```
  docker --version
  docker run hello-world
  ```

Linux

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl enable docker
sudo systemctl start docker
docker --version
```

---

6. Basic Docker Commands

Task	Command	Description
Check version	`docker --version`	Verify Docker installation
Run a container	`docker run hello-world`	Test setup
List containers	`docker ps`	Active containers
List all (incl. stopped)	`docker ps -a`	All containers
List images	`docker images`	Installed images
Pull image	`docker pull nginx`	Download from Docker Hub
Run interactive	`docker run -it ubuntu bash`	Open terminal inside Ubuntu
Stop container	`docker stop <container_id>`	Graceful stop
Remove container	`docker rm <container_id>`	Delete container
Remove image	`docker rmi <image_id>`	Delete image

---

1. What is a Docker Image?

A **Docker image** is a **read-only, layered template** used to create Docker containers.
Each image contains everything needed to run a piece of software — code, runtime, system libraries, environment variables, and configuration files.

---

2. Layered File System (Union File System)

Docker images are made up of **multiple layers**, stacked on top of each other using a **union file system** (like OverlayFS, AUFS, or Btrfs).

Each layer represents a **set of filesystem changes** (add, modify, delete files).

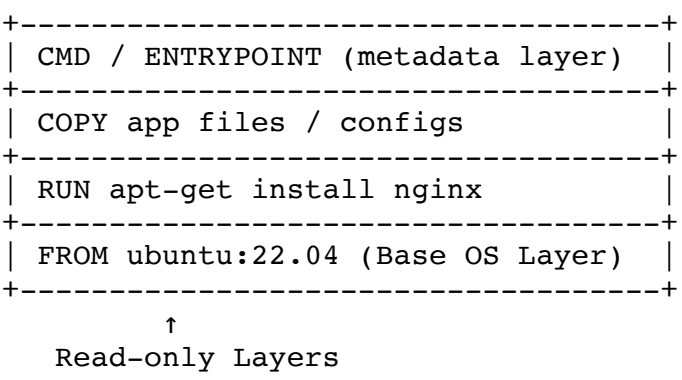
Example:

```
FROM ubuntu:22.04      → Base layer
RUN apt-get install nginx -y → Adds a new layer
COPY . /var/www/html   → Adds another layer
CMD ["nginx", "-g", "daemon off;"] → Final metadata layer
```

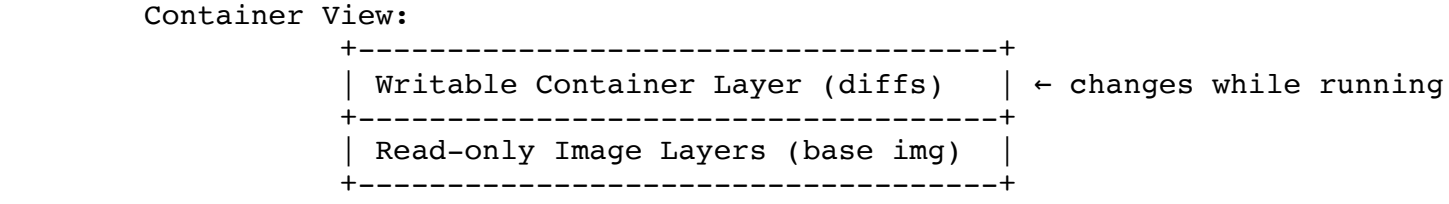
Each `RUN`, `COPY`, or `ADD` command in a Dockerfile creates a **new layer**.

---

3. Visual: Docker Image Layer Architecture



When you start a **container**, Docker adds a **read-write layer** on top of these **read-only layers**:



---

4. How Union File System Works

UnionFS merges all these layers into a **single unified view**.

If a file exists in multiple layers, Docker uses the **topmost copy** (copy-on-write mechanism).

If a file is **modified or deleted** in a container:

- It's first **copied** from the read-only layer into the writable layer.
- Modifications happen **only** in the writable layer.

---

5. Where Are Docker Images Stored?

Depends on the **storage driver** and **operating system**.

On Linux (default: `overlay2`)

Images and layers are stored under:

```
/var/lib/docker/overlay2/
```

Structure:

```
/var/lib/docker/
├── overlay2/
│   ├── <layer_id>/
│   │   ├── diff/       ← actual filesystem changes
│   │   ├── lower/      ← references to parent layers
│   │   └── work/       ← temp area used by OverlayFS
├── image/
│   └── overlay2/
│       ├── imagedb/
│       │   └── content/sha256/   ← image metadata
│       └── layerdb/              ← layer relationships
```

---

6. Inspecting Layers

You can view layers and their digests using:

```
docker image inspect <image_name> --format='{{json .RootFS.Layers}}' | jq
```

Or list all layers:

```
docker history <image_name>
```

Example output:

```
IMAGE          CREATED BY                                      SIZE
<missing>      /bin/sh -c #(nop)  CMD ...                      0B
<missing>      /bin/sh -c apt-get install nginx...             25MB
<missing>      /bin/sh -c apt-get update                       1.2MB
<missing>      /bin/sh -c #(nop)  FROM ubuntu:22.04
```

Each line = one layer.

---

7. Layer Reuse and Caching

Docker uses **content-addressable storage** — layers are identified by SHA256 hashes.

- If two images use the same base layers, Docker **does not duplicate** them.
- This makes builds **faster** and **space-efficient**.