Docker Concepts

🧠 1. Images

- A **Docker image** is like a **template or blueprint** for creating containers.
- It contains everything your app needs: code, runtime, libraries, and configuration files.
- Images are **read-only** — you can't change them once built.
- You can **pull** existing images (like `ubuntu`, `python`, or `nginx`) from **Docker Hub** or **build** your own using a `Dockerfile`.

Example:

docker pull python:3.11

This downloads the official Python 3.11 image.

📦 2. Containers

- A **container** is a **running instance** of an image.
- You can think of it as a lightweight, isolated environment for your application.
- Containers share the host's OS kernel, making them fast and efficient.
- You can start, stop, restart, or remove containers easily.

Example:

docker run -d -p 8080:80 nginx

Runs an NGINX web server in the background (`-d`) and maps port 8080 on your machine to port 80 inside the container.

🏗️ 3. Dockerfile

- A **Dockerfile** is a script with a list of instructions to build an image.
- Each line in a Dockerfile creates a new **layer** in the image.

Example:

```
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

This defines how to build your app's Docker image.

🧱 4. Layers

- Every command in a Dockerfile adds a **layer** to the image (e.g., `RUN`, `COPY`).
- Layers are **cached**, which makes rebuilds faster — Docker only rebuilds changed layers.
- Layers are **read-only**, and when a container runs, Docker adds a **writable layer** on top.

🔧 5. Volumes

- **Volumes** are used to **store data** outside a container's writable layer.
- This means your data **persists** even if the container is deleted.

- Commonly used for databases or logs.

Example:

docker run -v mydata:/data mongo

Creates a volume named `mydata` and mounts it at `/data` inside the MongoDB container.

🌐 6. Networks

- Docker **networks** allow containers to communicate with each other or the outside world.
- By default, containers on the same network can talk to each other by name.

Example:

```
docker network create mynetwork
docker run -d --network mynetwork --name db mongo
docker run -d --network mynetwork --name app myapp
```

Now `app` can reach `db` via hostname `db`.

🏭 7. Docker Compose

- **Docker Compose** lets you define and run **multi-container applications** using a YAML file (`docker-compose.yml`).
- It's great for defining app stacks (e.g., web + database + cache).

Example:

```
version: '3'
services:
  web:
    build: .
```

```
    ports:
      - "8000:8000"
  db:
    image: postgres:15
    volumes:
      - db_data:/var/lib/postgresql/data
volumes:
  db_data:
```

Then run:

docker compose up

🏗️ 8. Registry

- A **registry** is where Docker images are stored and shared.
- **Docker Hub** is the default public registry, but companies often use private registries (like AWS ECR, GitHub Container Registry).

Example:

docker push myusername/myapp:latest

🧩 9. Docker Daemon, CLI, and Client

Component	Description
Docker Daemon (`dockerd`)	The background service that builds and runs containers.
Docker CLI (`docker`)	The command-line tool you use to talk to the daemon.
Docker Client / API	Allows other tools (like VS Code or Docker Desktop) to communicate with Docker.

⚙️ How They Work Together

1. You write a `Dockerfile`.
2. Build an **image** from it.
3. Run **containers** from that image.
4. Optionally, connect them using **networks** and store persistent data using **volumes**.
5. Share or deploy your image using a **registry**.