**Docker**:

Docker is a **lightweight containerization tool** that allows applications to run in isolated environments while sharing the same OS kernel. It is **faster, more efficient, and portable** compared to virtual machines, making it ideal for **microservices, cloud deployments, and CI/CD pipelines**.

**Docker Image:**

A **Docker Image** is a **lightweight, standalone package** that contains everything needed to run an application, including:  
✅ Code 📝  
✅ Dependencies 📦  
✅ Libraries 📚  
✅ System tools ⚙️

It acts like a **blueprint** 🏗️ for creating **Docker Containers** (running instances of the image).

Think of it like a **frozen pizza** 🍕—you can store it, share it, and quickly "cook" (run) it whenever needed! 🚀

#### Images are immutable. Once an image is created, it can't be modified. You can only make a new image or add changes on top of it.

#### Container images are composed of layers. Each layer represents a set of file system changes that add, remove, or modify files.

#### These two principles let you to extend or add to existing images. For example, if you are building a Python app, you can start from the [Python image](https://hub.docker.com/_/python) and add additional layers to install your app's dependencies and add your code. This lets you focus on your app, rather than Python itself.

**Docker Registry:**

An image registry is a centralized location for storing and sharing your container images. It can be either public or private. [Docker Hub](https://hub.docker.com/) is a public registry that anyone can use and is the default registry.

Registery has many repositories and repositories have multiple Images.

**Container**

A **Docker Container** is a **running instance** of a **Docker Image**. It is an isolated, lightweight environment where an application runs with all its dependencies.

Think of it like a **microwaved pizza** 🍕—the **image** is the frozen pizza, and when you "run" it (container), it becomes hot and ready to eat! 😃

**Virtual Machine vs Docker**

**VMs** = Separate toy boxes 🗃️ (each has its own OS, takes more space, slower).  
**Docker** = One big toy box with sections 🏠 (shares OS, faster, uses less space).

Use **VMs** for full isolation, and **Docker** for lightweight, fast app deployment. 🚀

**Kernel:**

The **kernel** is the **core part of an operating system (OS)** that acts as a bridge between **hardware** (CPU, memory, storage) and **software** (applications).

**Think of it like:**

🧠 **Brain of the OS** – It controls everything.  
🗣️ **Translator** – It helps apps talk to hardware.  
🚦 **Traffic Controller** – Manages CPU, memory, and processes efficiently.

**Port**

Port numbers are **16-bit values** (0-65535) assigned by the **Internet Assigned Numbers Authority (IANA)**

A **port** is a **communication endpoint** for sending and receiving data over a network.

### ****Think of it like a door 🚪:****

* Your **computer (house)** has many doors (ports).
* Each **door (port)** is assigned to a specific service (e.g., **Port 80 for websites, Port 22 for SSH**).
* Apps use ports to **talk** to each other over the internet or a local network.

**Common Ports & Their Uses**

* **Port 80** → HTTP (Web Browsing, not secure)
* **Port 443** → HTTPS (Secure Web Browsing)
* **Port 22** → SSH (Secure Shell)
* **Port 3306** → MySQL Database
* **Port 5432** → PostgreSQL Database

#### **Example: Talking to a Web Server**

🧑‍💻 **You (Browser)** → "Hey, server! Show me http://example.com!" (Sent via Port 80)  
🌐 **Server (Website)** → "Here’s your webpage!" (Reply via Port 80)

So, **Port 80 is used both on your PC (to send requests) and on servers (to receive them)**.

**Ports in Docker:**

docker run -p <host-port>:<container-port> my-container

**Host:**

A **host** is any device **(computer, server, or router)** that connects to a network and can **send or receive data**.

### ****2️⃣ Host vs. Port vs. IP Address****

📌 **Host** = The device (computer/server) running services.  
📌 **IP Address** = The unique identifier of a host on a network.  
📌 **Port** = The specific entry point on the host for a service (e.g., Port 80 for websites).

🔹 **Docker Host (Your Computer Running Docker)**

* If you run Docker, your **PC acts as a host** for containers.
* Containers inside Docker talk to the **host machine** using special networking rules.

There are **no "host numbers"** because a **host is identified by an IP address, not a number**. However, there are **common hostnames & IP addresses** that are frequently used in networking.

**1️⃣ Local Host (Your Own Computer)**

| **Host** | **Description** |
| --- | --- |
| localhost | Refers to **your own PC** (loopback address) |
| 127.0.0.1 | Same as localhost, used to test local applications |
| 0.0.0.0 | Means "all available network interfaces" (used in servers) |

Example: Running http://localhost:5000/ accesses a web server on **your own machine**.

**# Use Python as the base image**

**FROM python:3.9**

**# Set the working directory**

**WORKDIR /app**

**# Copy files into the container**

**COPY . .**

**# Install dependencies**

**RUN pip install -r requirements.txt**

**# Set environment variables**

**ENV PYTHONUNBUFFERED=1**

**# Expose port 5000**

**EXPOSE 5000**

**# Set default command**

**CMD ["python", "app.py"]**

**Summary of Important Dockerfile Commands**

| **Command** | **Purpose** |
| --- | --- |
| FROM | Set the base image |
| WORKDIR | Change working directory |
| COPY / ADD | Copy files from host to container |
| RUN | Execute a command inside the container (during build) |
| ENV | Set environment variables |
| EXPOSE | Open a port for the app |
| CMD / ENTRYPOINT | Set the default startup command |
| VOLUME | Create persistent storage |
| HEALTHCHECK | Monitor if the app is running properly |

Docker Hub provides **official images** for common programming languages & OS.

| **Base Image** | **Purpose** |
| --- | --- |
| FROM ubuntu:latest | Ubuntu OS (Linux-based apps) |
| FROM alpine:latest | Lightweight Linux (small, fast) |
| FROM python:3.9 | Python development |
| FROM node:18 | Node.js for JavaScript apps |
| FROM mysql:8.0 | MySQL database |
| FROM postgres:15 | PostgreSQL database |

or a **Machine Learning (ML)** project, you'll want a base image that provides **popular ML libraries** (like **TensorFlow**, **PyTorch**, **scikit-learn**, etc.), and is **optimized for data science** tasks.

**Summary: Which Base Image to Choose? 🤔**

| **Base Image** | **Best For** | **Why Choose It?** |
| --- | --- | --- |
| tensorflow/tensorflow | TensorFlow-based projects | Pre-configured for TensorFlow (CPU & GPU support) |
| pytorch/pytorch | PyTorch-based projects | Optimized for PyTorch (with GPU support) |
| continuumio/miniconda3 | Custom dependency management | Flexibility with Conda environments |
| jupyter/base-notebook | Jupyter Notebook development | Ideal for interactive experimentation |
| nvidia/cuda | GPU-accelerated training | Provides CUDA for **NVIDIA GPU** acceleration |
| ubuntu | Custom or complex setups | Full control over your environment |

**# Step 1: Build the image**

docker build -t YOUR\_DOCKER\_USERNAME/docker-quickstart .

**# Step 2: Verify the built image**

docker images

**# Step 3: Run a container from the image**

docker run -d -p 8080:8080 YOUR\_DOCKER\_USERNAME/docker-quickstart

**# Step 4: Check if the container is running**

docker ps

**# Step 5: Tag the image with a version**

docker tag YOUR\_DOCKER\_USERNAME/docker-quickstartYOUR\_DOCKER\_USERNAME/docker-quickstart:1.0

**# Step 6: Push the image to Docker Hub**

docker push YOUR\_DOCKER\_USERNAME/docker-quickstart:1.0

**Docker pull**

docker pull YOUR\_DOCKER\_USERNAME/docker-quickstart:1.0

**Docker compose yml**

If we have multiple containers and run them all together one for database, one for backend etc.