**Containerization**: A lightweight virtualization method that packages applications and their dependencies into isolated containers using a shared operating system kernel. Examples: Docker, Kubernetes.

**Virtualization**: A technology that creates multiple virtual machines (VMs) on a single physical server, each with its own operating system and resources, managed by a hypervisor. Examples: VMware, VirtualBox

**Docker Architecture**

Docker follows a **client-server** architecture consisting of the following key components:

1. **Docker Client**
   * The user interacts with Docker using the client (docker CLI or Docker API).
   * It sends commands to the Docker daemon to build, run, and manage containers.
2. **Docker Daemon (dockerd)**
   * Runs in the background and manages containers, images, networks, and volumes.
   * Listens for API requests from the Docker client.
3. **Docker Engine**
   * The core component that includes the Docker daemon, CLI, and API.
   * Responsible for building and running containers.
4. **Docker Images**
   * Read-only templates used to create containers.
   * Built using Dockerfile and stored in a registry.
5. **Docker Containers**
   * Lightweight, isolated environments that run applications using Docker images.
   * Share the host OS kernel but have their own file system and dependencies.
6. **Docker Registry**
   * A storage system for Docker images (e.g., Docker Hub, private registry).
   * Allows users to push and pull images.
7. **Docker Network**
   * Enables communication between containers and external services.
   * Supports bridge, host, and overlay networks.
8. **Docker Volume**
   * Persistent storage mechanism for containers.
   * Allows data sharing between containers.

**Dockerfile Explained**

A **Dockerfile** is a script that contains a set of instructions to automate the creation of a Docker image. It defines the base image, dependencies, configurations, and commands needed to build and run a containerized application.

**Basic Structure of a Dockerfile**

A Dockerfile consists of multiple instructions, each performing a specific function. Below are the key instructions:

1. **FROM**
   * Specifies the base image for the container.
   * Example:
   * FROM ubuntu:latest
2. **MAINTAINER (Deprecated, use LABEL instead)**
   * Specifies the author of the image.
   * Example:
   * LABEL maintainer="yourname@example.com"
3. **RUN**
   * Executes commands during the image build process.
   * Example:
   * RUN apt-get update && apt-get install -y nginx
4. **COPY**
   * Copies files or directories from the host machine to the container.
   * Example:
   * COPY index.html /usr/share/nginx/html/
5. **ADD**
   * Similar to COPY, but supports automatic extraction of compressed files.
   * Example:
   * ADD app.tar.gz /app/
6. **WORKDIR**
   * Sets the working directory inside the container.
   * Example:
   * WORKDIR /app
7. **CMD**
   * Defines the default command to run when the container starts.
   * Example:
   * CMD ["nginx", "-g", "daemon off;"]
   * Unlike RUN, CMD executes when the container starts, not during the image build.
8. **ENTRYPOINT**
   * Similar to CMD, but allows passing additional arguments when running the container.
   * Example:
   * ENTRYPOINT ["python"]
   * CMD ["app.py"]
9. **EXPOSE**
   * Specifies the port the container will listen on.
   * Example:
   * EXPOSE 80
10. **ENV**
    * Sets environment variables in the container.
    * Example:
    * ENV APP\_ENV=production
11. **VOLUME**
    * Creates a mount point for persistent storage.
    * Example:
    * VOLUME /data
12. **ARG**
    * Defines variables that can be passed during build time.
    * Example:
    * ARG VERSION=1.0
13. **HEALTHCHECK**
    * Monitors the health of the running container.
    * Example:
    * HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
    * CMD curl -f http://localhost || exit 1
14. **USER**
    * Specifies the user to run the container as.
    * Example:
    * USER nonroot
15. **ONBUILD**
    * Executes commands when the image is used as a base for another build.
    * Example:
    * ONBUILD COPY . /app

**Example Dockerfile for a Node.js App**

# Use the official Node.js image as the base image

FROM node:18

# Set the working directory inside the container

WORKDIR /app

# Copy package.json and install dependencies

COPY package.json .

RUN npm install

# Copy the rest of the application files

COPY . .

# Expose the application port

EXPOSE 3000

# Start the application

CMD ["node", "server.js"]

**Building and Running a Dockerfile**

1. **Build the Docker image**
2. docker build -t mynodeapp .
3. **Run the container**
4. docker run -p 3000:3000 mynodeapp