Containers, Docker and Kubernetes – Revision Notes

Containers: Overview

- **Virtualization** enables shared use of resources across customers in cloud computing.
- A **Container** is a lightweight virtualization technique.
- It **packages the application code and its dependencies**, enabling consistent operation across environments.
- Containers **virtualize the OS**, not the hardware, making them **lightweight, portable, and efficient**.
- Run environments: personal laptops, data centers, public cloud.

Why Containers?

- **Environment Decoupling**: Run consistently regardless of underlying infrastructure.
- **Agile Development**: Avoids dependency issues, speeds up dev workflows.
- **Efficient Operations**: Use only the required resources, increasing efficiency.
- **Portability**: Run on Linux, Windows, Mac, VMs, physical servers, and cloud.
- **Isolation**: Each container has its own space (CPU, memory, storage, network).

Benefits of Containers

- **Separation of Responsibilities**:
  - Devs focus on code & dependencies.
  - Ops handle deployment & infrastructure.
- **Faster Deployment & Scaling**
- **Microservices-friendly Architecture**
- **Easy Sharing & Collaboration**

Deployment Models

Model	Description
Traditional	Apps run on physical servers with no resource boundaries.
Virtualized	Apps run in VMs, each with its own OS; better isolation, more overhead.
Containerized	Apps share the OS kernel, lightweight, faster startup, highly portable.

Containers vs Virtual Machines (VMs)

Feature	Containers	Virtual Machines
Virtualization	OS level	Hardware level
Resource Use	Lightweight (share OS kernel)	Heavy (separate OS per VM)
Speed	Fast startup, low overhead	Slower startup, more overhead

Docker: Container Platform

- **Docker Engine**: Core runtime for building and running containers.
- **Docker Container Image**: Complete standalone software package.
- **Cross-platform**: Works across Linux and Windows environments.
- **Benefits**:
  - Replaces heavy VMs.
  - Quick prototyping & sandboxing.
  - Great for microservices & full-stack offline development.
  - Improves collaboration & debugging.
  - Enables **Continuous Delivery (CD)** pipelines.

Docker Analogy:

Like traditional dockers fit odd-shaped goods into ships, Docker fits software with different dependencies into containers for reliable shipping to any environment.

Common Docker Commands

Command	Description
docker build	Build a Docker image
docker run	Run a container from an image
docker commit	Save container changes as a new image
docker tag	Tag Docker image for versioning/sharing

Docker Architecture

- **Docker Daemon**: Background service managing images, containers.
- **Docker Client**: CLI tool that communicates with the daemon.

- **Docker Registry**: Stores Docker images (e.g., Docker Hub, private registries).

Kubernetes: Container Orchestration

- Open-source platform for **automating deployment, scaling, and management** of containerized apps.
- **Origin**: "Kubernetes" means helmsman or pilot in Greek.
- Works at the **container level**, not hardware.

Key Features:

- **Declarative configuration & automation**
- **Scalability, Load balancing, Fault tolerance**
- **Pluggable architecture (monitoring, logging, etc.)**

Kubernetes Components:

Component	Description
Cluster	Group of machines (nodes)
Node	A worker machine hosting containers
Pod	Smallest unit containing one/more containers
Control Plane	Manages the cluster and scheduling of Pods