4.5 TOOLS

1. Docker

Introduction

- **Docker** is an open-source platform used to **develop, package, and run applications inside containers**.
- A **container** is a lightweight environment that includes the **application and all its dependencies**, such as libraries and configuration files.
- This ensures that the application runs **the same way in development, testing, and production environments**.
- Docker uses **containerization technology**, which allows multiple applications to run on the same system while remaining isolated from each other.

How Docker Works:

Docker works using three main components:

1. Docker Image:

A **Docker image** is a read-only template that contains:

- application code
- libraries
- dependencies
- system tools

Images are used to create containers.

Example:

A Python application image may contain **Python runtime + required libraries**.

2. Docker Container:

A **container** is a running instance of a Docker image.

Containers are:

- lightweight
- fast to start
- isolated from other containers

Example:

A company can run **multiple containers for web servers, databases, and APIs** on the same machine.

3. Docker Engine:

The **Docker Engine** is the core service that manages Docker containers.

It performs tasks such as:

- building images
- running containers
- managing container resources

Docker Architecture:

Docker architecture consists of the following components:

Docker Client:

The Docker client is the **command-line interface** used to interact with Docker.

Example commands:

- docker build
- docker run
- docker pull

Docker Daemon

- The Docker daemon runs in the background and **manages containers and images**.
- It listens to Docker API requests from the Docker client.

Docker Registry:

A **Docker registry** stores Docker images.

Example registry:

Docker Hub: Developers can download pre-built images from the registry.

Advantages of Docker: Docker provides several advantages:

Portability: Applications run consistently across different environments.

Lightweight: Containers share the host OS kernel and require fewer resources.

Fast Deployment: Containers start in seconds.

Scalability: Applications can be scaled easily by creating multiple containers.

Isolation: Each container runs independently.

Example of Docker:

Consider a **web application** developed using Python.

Without Docker:

- Install Python
- Install dependencies
- Configure environment

With Docker:

- Create a Docker image containing everything
- Run the container anywhere

The application will run **exactly the same on every system**.

2. KVM (Kernel-based Virtual Machine)

Introduction

- **Kernel-based Virtual Machine** is a **virtualization technology built into the Linux kernel**.
- KVM allows a Linux system to act as a **hypervisor**, enabling it to run multiple **virtual machines**.
- Each virtual machine can run its **own operating system**, such as Windows, Linux, or other operating systems.

How KVM Works

- KVM converts the **Linux kernel into a hypervisor**.
- A hypervisor is software that allows **multiple operating systems to run on a single physical machine**.

Each VM created by KVM has its own:

- virtual CPU
- virtual memory
- virtual disk
- virtual network interface

The virtualization is supported by hardware features such as:

- Intel VT-x
- AMD-V

Components of KVM

KVM Kernel Module

- The KVM kernel module provides **core virtualization support** in Linux.
- It allows the system to create and manage virtual machines.

QEMU

QEMU works together with KVM.

QEMU provides:

- hardware emulation
- device virtualization

KVM improves QEMU performance by using hardware virtualization.

Libvirt:

libvirt is a management tool used to control KVM virtual machines.

It allows administrators to:

- create VMs
- manage storage
- configure networking

Features of KVM:

KVM provides several features:

Full Virtualization: Each VM runs a complete operating system.

High Performance: Uses hardware virtualization for faster performance.

Security Isolation: Each VM is isolated from others.

Scalability: Supports multiple virtual machines on one server.

Live Migration: VMs can be moved from one host machine to another without downtime.

Example of KVM:

A cloud provider can use KVM to create multiple virtual machines on a server.

Example: Server hardware → KVM hypervisor → Multiple VMs

VM1 → Linux

VM2 → Windows

VM3 → Ubuntu

Each VM runs independently.

Difference Between Docker and KVM

Feature	**Docker**	**KVM**
Technology	Containerization	Virtualization
OS Requirement	Uses host OS kernel	Runs separate OS
Resource Usage	Lightweight	Higher resource usage
Startup Time	Seconds	Minutes
Isolation Level	Process-level	Full machine-level