Chapter 3 - Docker Technology

Chapter 3: Docker Technology

1. Definition

Docker is an open-source platform that enables developers to automate the deployment, scaling, and management of applications using **containers**. Containers allow applications to run consistently across different environments by packaging the code, runtime, system tools, libraries, and dependencies together.

Key Features of Docker

- Lightweight and fast compared to virtual machines.
- Provides consistency across multiple environments (development, testing, production).
- Enables microservices architecture.
- Supports automation and CI/CD integration.

---

2. Virtualization vs Containerization

Virtualization

- Uses **Virtual Machines (VMs)** to run multiple operating systems on a single physical machine.
- Requires a **Hypervisor** (e.g., VMware, VirtualBox, KVM).
- Each VM has its own OS, libraries, and dependencies, making them **heavy** and **resource-intensive**.
- Examples: VMware, VirtualBox, Hyper-V.

Containerization

- Uses **containers** instead of full OS instances.
- Shares the **host OS kernel**, making it lightweight and efficient.
- Containers are **faster** to start and use fewer resources than VMs.
- Example: Docker, Podman, Kubernetes.

Feature	Virtualization (VM)	Containerization (Docker)
OS Dependency	Each VM has its own OS	Containers share host OS kernel
Performance	Slower, more resource-intensive	Faster, lightweight
Startup Time	Minutes	Seconds
Portability	Limited	Highly portable

---

3. Containers

A **container** is a lightweight, standalone, and executable package that contains everything needed to run an application.

Key Characteristics

- **Isolated:** Runs independently without affecting other applications.
- **Portable:** Runs on any system with Docker installed.
- **Efficient:** Shares OS resources, reducing overhead.

Basic Docker Container Commands

- `docker run <image>` → Start a container.
- `docker ps` → List running containers.
- `docker stop <container_id>` → Stop a running container.
- `docker rm <container_id>` → Remove a container.

---

4. Installation

Installing Docker on Windows, Linux, and Mac

Windows & Mac

1. Download Docker Desktop from Docker's official site.
2. Install the software and restart your machine.
3. Verify installation with:

```
docker --version
```

Linux (Ubuntu Example)

1. Update system packages:

```
sudo apt update
```

2. Install Docker:

```
sudo apt install docker.io -y
```

3. Enable and start Docker service:

```
sudo systemctl enable --now docker
```

4. Verify installation:

```
docker --version
```

---

5. Functioning and Manipulation of Docker Images

What is a Docker Image?

A **Docker Image** is a pre-configured package that includes an application, dependencies, and runtime environment.

Working with Docker Images

- **Pull an image from Docker Hub:**

```
docker pull ubuntu
```

- **List available images:**

```
docker images
```

- **Remove an image:**

```
docker rmi <image_id>
```

- **Build a custom image (Dockerfile Example):**

```
FROM ubuntu
RUN apt update && apt install -y nginx
CMD ["nginx", "-g", "daemon off;"]
```

```
docker build -t my-nginx .
```

---

6. Functioning and Manipulation of Containers

Starting a Container

```
docker run -d --name mycontainer ubuntu
```

Viewing Running Containers

```
docker ps  # Show active containers
docker ps -a  # Show all containers
```

Stopping and Restarting Containers

```
docker stop <container_id>
docker start <container_id>
docker restart <container_id>
```

Removing Containers

```
docker rm <container_id>
```

Accessing a Running Container

```
docker exec -it <container_id> /bin/bash
```

---

7. Functioning and Manipulation of Volumes

Docker **Volumes** are used to persist data even after a container stops.

Creating and Using Volumes

- **Create a volume:**

  ```
  docker volume create myvolume
  ```

- **List volumes:**

  ```
  docker volume ls
  ```

- **Run a container with a volume:**

  ```
  docker run -d -v myvolume:/data ubuntu
  ```

- **Inspect volume details:**

  ```
  docker volume inspect myvolume
  ```

- **Remove a volume:**

  ```
  docker volume rm myvolume
  ```

---

8. Docker Compose

Docker Compose is a tool that allows defining and managing multi-container applications using a YAML file.

Example `docker-compose.yml` File

```yaml
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  database:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
```

Commands to Manage Docker Compose

- **Start services:**

```bash
docker-compose up -d
```

- **Stop services:**

```bash
docker-compose down
```

---

9. Network with Docker

Types of Docker Networks

1. **Bridge (Default)** – Used for communication between containers on the same host.
2. **Host** – Shares the host's network.
3. **Overlay** – Used in multi-host Docker Swarm.
4. **None** – No network assigned.

Managing Docker Networks

- **List networks:**

```bash
docker network ls
```

- **Create a network:**

```
docker network create mynetwork
```

- **Run a container in a custom network:**

```
docker run -d --net=mynetwork nginx
```

- **Connect a running container to a network:**

```
docker network connect mynetwork mycontainer
```

- **Disconnect a container from a network:**

```
docker network disconnect mynetwork mycontainer
```

- **Remove a network:**

```
docker network rm mynetwork
```