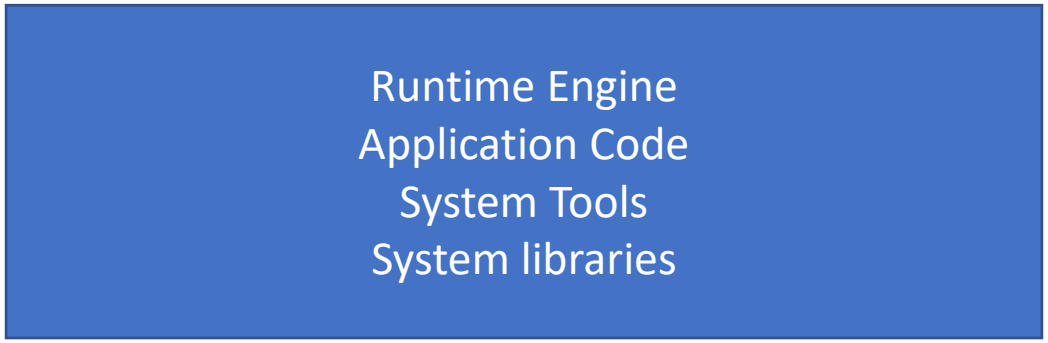
Docker Commands – From Basics to Networking

Objective

- Able to understand containers
- Understand basic docker commands
- Understand Volumes in Docker
- Understand Docker Networking
- Demo

Introduction to Containers and Docker

- A container is a lightweight, standalone software package that contains everything that a software application needs to run.



- A *container* is a standardized unit of software designed to run quickly and reliably on any computing environment that runs the containerization platform.

Docker containers

- Docker is a portable runtime application environment.
- You can package an application and its dependencies into a single, immutable artifact that is called an *image*.
- After you create a container image, it can go anywhere that Docker is supported.
- You can run different application versions with different dependencies simultaneously.

Basic Docker Commands

- Docker Version Check
  - docker --version
  - docker info
- Pulling Images
  - docker pull ubuntu
  - docker pull nginx
- Listing Images
  - docker images
- Running a Container
  - docker run hello-world
  - docker run -it ubuntu bash

- Listing Running Containers
  - docker ps
  - docker ps -a
- Stopping and Removing Containers
  - docker stop <container_id>
  - docker rm <container_id>
  - docker rmi <image_id>

Volume in Docker

- A **volume** is a **persistent storage mechanism** managed by Docker. It allows **data to persist** even when the container is stopped or deleted.
- Needs:
  - Data persistence (e.g., databases)
  - Sharing data between containers
  - Easier backup, migration, and inspection
  - Decouples container lifecycle from data lifecycle

Volume Command Basics

- Create a Volume
  - docker volume create myvolume
- List Volumes
  - docker volume ls
- Inspect Volume
  - docker volume inspect myvolume

Working Demo: Volume with Ubuntu Container

- Step 1: Create a Volume
  - docker volume create datavol
- Step 2: Start a Container Using the Volume
  - docker run -it --name voltest -v datavol:/data ubuntu
- Step 3: Inside the Container
  - cd /data
  - echo "Docker volume is working!" > message.txt
  - cat message.txt
  - exit
- Step 4: Start Another Container Using Same Volume
  - docker run -it --name voltest2 -v datavol:/data ubuntu
  - cd /data
  - cat message.txt

Docker Networking

- Docker containers are isolated processes. Networking allows:
  - Communication between containers
  - External access (host ↔ container)
  - Service discovery within a multi-container system
- **bridge** Network (Default for Standalone Containers)
  - docker run -d --name web nginx
- Inspecting Networks
  - docker network ls
  - docker network inspect bridge

Docker supports **five main network drivers**

Network Type	Description	Use Case	Host Access	DNS Resolution
bridge	Default local network	Multi-container apps on same host	Yes (via ports)	Yes
host	Shares host network namespace	High-performance or legacy apps	Full	No (uses host)
none	Isolated container	Total isolation (testing)	No	No
overlay	Multi-host communication via Docker Swarm	Clustered apps or services	Yes	Yes
macvlan	Assigns real IP to container	Legacy or MAC-bound apps	Yes	No (acts like VM)

Demo with LMS

- library-system/
- ├── book-service/
- │   ├── server.js
- │   ├── books.json
- │   └── Dockerfile
- ├── user-service/
- │   ├── server.js
- │   └── Dockerfile

Dockerfile (Books service)

- `FROM node:18-alpine`
- `WORKDIR /app`
- `COPY . .`
- `RUN npm install express`
- `VOLUME ["/data"]`
- `CMD ["node", "server.js"]`

Dockerfile (user service)

- FROM node:14
- WORKDIR /app
- COPY . .
- RUN npm install express
- VOLUME ["/data"]
- CMD ["node", "server.js"]

docker-compose.yml

```yaml
version: '3.8'

services:
  book-service:
    build: ./book-service
    ports:
      - "3000:3000"
    volumes:
      - bookdata:/data
    networks:
      - librarynet

  user-service:
    build: ./user-service
    ports:
      - "4000:4000"
    networks:
      - librarynet

volumes:
  bookdata:

networks:
  librarynet:
```

docker compose up --build