**Docker Compose and Networking**

**Lecture Notes**

**Introduction to Docker Compose**

Docker Compose is a tool used for defining and running multi-container Docker applications. It uses a YAML configuration file to define services, networks, and volumes.

**Why Use Docker Compose?**

1. **Multi-container management**: Allows running multiple containers as a single service.
2. **Simplified configuration**: All configurations are in a single YAML file.
3. **Easy environment replication**: Developers can share the configuration easily.
4. **Networking management**: Defines custom networks between containers.

**Installing Docker Compose**

If not installed, run:

sudo apt update

sudo apt install docker-compose -y

Verify the installation:

docker-compose --version

**Writing a Basic Docker Compose File**

A docker-compose.yml file defines services. Below is an example for running a web application with a database.

version: '3.8'

services:

web:

image: nginx

ports:

- "8080:80"

networks:

- app-network

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: mydb

networks:

- app-network

networks:

app-network:

driver: bridge

**Explanation:**

* Defines two services: web (Nginx) and db (MySQL).
* Maps host port 8080 to container port 80.
* Uses app-network for internal communication.
* MYSQL\_ROOT\_PASSWORD and MYSQL\_DATABASE set environment variables for MySQL.

**Running the Compose File**

To start the application:

docker-compose up -d

To view running services:

docker-compose ps

To stop services:

docker-compose down

**Docker Networking**

Docker networks allow communication between containers. There are three types:

1. **Bridge Network** (default): Used for containers on the same host.
2. **Host Network**: Removes network isolation; uses the host’s networking.
3. **Overlay Network**: Used for multi-host Docker Swarm deployments.

**Creating and Managing Networks**

List existing networks:

docker network ls

Create a custom network:

docker network create mynetwork

Inspect a network:

docker network inspect mynetwork

Attach a running container to a network:

docker network connect mynetwork container\_name

Disconnect a container from a network:

docker network disconnect mynetwork container\_name

Remove a network:

docker network rm mynetwork

**Hands-on Exercise**

**Task 1: Create a Multi-Container Application with Custom Network**

1. Write a docker-compose.yml file with two services (web and database) using a custom network.
2. Run docker-compose up -d.
3. Verify the network using docker network ls and docker network inspect app-network.
4. Access the web service from a browser at http://localhost:8080.

**Task 2: Network Communication Test**

1. Start a container and attach it to a network:
2. docker run -dit --name test-container --network app-network alpine sh
3. Access the MySQL container from the test-container:
4. docker exec -it test-container sh
5. apk add --no-cache mysql-client
6. mysql -h db -u root -p

**Exam-Oriented Questions**

1. Explain Docker Compose and its advantages.
2. Write a Docker Compose file for a Python web application and PostgreSQL database.
3. Explain the types of Docker networks with examples.
4. Write commands to create, inspect, and remove a Docker network.
5. What is the difference between docker-compose up and docker-compose down?

**Why Do You Need Docker Network?**

Docker networking allows containers to communicate with each other and with external systems. Here’s why it is essential:

**1. Isolation and Security**

* By default, containers are isolated from each other and the host.
* Networks allow controlled communication while maintaining isolation.
* You can define custom networks to limit access between services.

**2. Container-to-Container Communication**

* Services running in different containers (e.g., a web app and a database) need a way to interact.
* Without networks, containers would require explicit IP addresses, making management complex.
* Docker’s internal DNS resolves container names automatically, so you can refer to containers by name instead of IP.

**3. Load Balancing and Scalability**

* Custom bridge networks enable service discovery, making it easier to scale applications dynamically.
* Overlay networks allow multi-host deployments in Docker Swarm.

**4. Multi-Environment Deployment**

* In production, different services (databases, cache, application servers) need to communicate while remaining accessible from the outside.
* A well-structured network helps configure public and private connections efficiently.

**Example of Networking in Docker Compose**

With Docker Compose, we define a custom network for controlled communication:

version: '3.8'

services:

web:

image: nginx

networks:

- mynetwork

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: root

networks:

- mynetwork

networks:

mynetwork:

driver: bridge

* Both web and db services can communicate on mynetwork without exposing the database externally.

**Why Docker Compose is Preferred?**

Docker Compose simplifies the process of managing multi-container applications by providing a single YAML configuration file. Here’s why it’s preferred:

**1. Simplifies Multi-Container Setup**

* Without Compose, you’d need to manually start each container with docker run and link them.
* Compose manages multiple services, networks, and volumes effortlessly.

**2. Readable and Maintainable Configuration**

* A docker-compose.yml file keeps all configurations in one place.
* Easy to version control, share, and modify.

**3. One-Command Deployment**

* Running docker-compose up -d sets up and starts everything at once.
* docker-compose down stops all services cleanly.

**4. Better Networking and Service Discovery**

* Automatically creates an isolated network (default or custom).
* Containers can refer to each other by service name (e.g., db instead of an IP).

**5. Environment Variables and Port Mapping**

* Compose allows defining environment variables dynamically.
* Ports can be exposed and mapped to the host system easily.

**Example Without Compose (Manual Method)**

docker network create mynetwork

docker run -d --name db --network mynetwork -e MYSQL\_ROOT\_PASSWORD=root mysql:5.7

docker run -d --name web --network mynetwork -p 8080:80 nginx

* Requires multiple commands.
* Hard to manage and scale.

**Example With Compose**

docker-compose up -d

* **One command** to start all services with proper networking.

**Final Takeaway**

* **Use Docker networks** to enable secure, efficient communication between containers.
* **Use Docker Compose** to simplify multi-container orchestration, manage networking, and improve maintainability.