Docker Advanced: Networks, Volumes, and Microservices

Objectives :

- Run multiple containers together using Docker Compose
- Understand microservices communication using RabbitMQ
- Observe container-to-container networking
- Understand data persistence using Docker volumes
- Edit and understand YAML configuration files

Concept Overview :

Microservices

Microservices architecture is a modern software design approach in which an application is developed as a collection of small, independent, and loosely coupled services. Each microservice is responsible for a specific business capability, runs as an independent process, and communicates with other services using lightweight protocols such as HTTP/REST, gRPC, or message queues. So, instead of building one large application, modern systems are split into small services that do one job well and communicate with each other.

In this lab:

- One service produces messages
- One service consumes and stores messages

- RabbitMQ acts as the message broker

RabbitMQ

RabbitMQ is an open-source message broker that enables asynchronous communication between different components of an application. It acts as an intermediary that routes messages from producers to consumers by implementing the Advanced Message Queuing Protocol (AMQP).

Advantages of using RabbitMQ as a message queue system:

1) Asynchronous Communication
   Send messages without waiting for an immediate response. This helps services work independently and improves overall system performance.

2) Loose Coupling Between Services
   The sender and receiver do not need to know about each other. RabbitMQ sits in between, making the system easier to modify and scale.

3) Reliable Message Delivery
   Messages can be stored safely in queues until they are processed. This prevents data loss even if a consumer service temporarily fails.

4) Scalable Message Handling
   Multiple consumers can read from the same queue, allowing RabbitMQ to handle increased workload efficiently in cloud environments.

5) Flexible Message Routing
   RabbitMQ supports different routing methods (direct, topic, fanout), making it easy to send messages to the right service based on rules.

Docker Networks

Docker automatically creates a private network for containers defined in Docker Compose. This allows:

- Containers can talk to each other using container names instead of IP addresses, simplifying configuration in cloud deployments.

- Secure internal communication (without being exposed to the public)
- Containers on different networks cannot see each other, improving security and preventing unwanted access

Docker Volumes

By default, data inside a container is lost when the container stops. Volumes store data outside containers, ensuring data is not lost when containers stop or restart. Additionally, multiple containers can access the same volume, making it useful for shared data in cloud applications.

In this lab:

- Chat history is stored in a file

- You will compare behavior with and without volumes

<u>Prerequisites before starting the lab :</u>

1. Ubuntu (native or WSL)

2. Docker installed

3. Docker Compose installed

Do NOT attempt this lab directly on Windows CMD or PowerShell. (This lab uses Linux-based containers, volume mounts, and networking features that behave inconsistently on Windows CMD/PowerShell). To avoid environment-related issues, use Ubuntu (native or WSL).

Step 1: Verify Docker Installation