**🚢 1. What is Docker Networking?**

Docker networking allows containers to **communicate** with each other, with the Docker host, and with external networks (like the internet). It is a critical aspect of container orchestration, especially in multi-container apps like microservices.

**✅ Purpose:**

* Container-to-container communication
* Container-to-host communication
* Isolating containers on different networks
* Load balancing and service discovery

**🌐 2. Types of Docker Networks**

Docker provides several types of networks out-of-the-box:

| **Network Type** | **Description** | **Use Case** |
| --- | --- | --- |
| **bridge** *(default)* | Creates a private internal network on the host | Good for standalone container apps |
| **host** | Shares the host’s network stack | Use when container needs max network performance or host-level networking |
| **none** | Disables all networking | Useful for security, testing isolated behavior |
| **overlay** | Enables communication between containers across multiple Docker hosts (Swarm) | Distributed multi-host deployments |
| **macvlan** | Assigns MAC address to container; appears as physical device | For applications that need to appear on the physical network |
| **custom bridge** | User-defined bridge with custom subnet, DNS | More control over inter-container comms |

**Note**: You can inspect a network using:

docker network inspect <network-name>

**🔁 3. Docker Container Lifecycle**

Represents the **journey of a container** from creation to deletion.

1. **Create**
   * Command: docker create
   * Container is created but not started.
2. **Start**
   * Command: docker start <container\_id>
   * Container begins execution.
3. **Running**
   * Actively executing processes inside container.
4. **Pause** *(optional)*
   * Command: docker pause <container\_id>
   * Suspends processes without killing.
5. **Unpause** *(optional)*
   * Command: docker unpause <container\_id>
6. **Stop**
   * Command: docker stop <container\_id>
   * Gracefully stops all processes.
7. **Kill**
   * Command: docker kill <container\_id>
   * Immediately kills all processes.
8. **Restart**
   * Command: docker restart <container\_id>
9. **Remove**
   * Command: docker rm <container\_id>
   * Deletes the container.

🧠 You can check all container states via: docker ps -a

**📄 4. Dockerfile Lifecycle (Image Build Lifecycle)**

The **Dockerfile** defines a set of instructions to create a Docker image. Lifecycle refers to the **step-by-step process Docker follows** to turn that file into an image.

**✅ Common Dockerfile Lifecycle Stages:**

| **Stage** | **Instruction** | **Purpose** |
| --- | --- | --- |
| **Base Image** | FROM | Sets the base OS/image |
| **Maintainer** | LABEL | Metadata about the image |
| **Run Commands** | RUN | Executes shell commands during build |
| **Working Directory** | WORKDIR | Sets working directory for following instructions |
| **Copy/Add Files** | COPY / ADD | Moves files into image |
| **Install Dependencies** | RUN apt-get install | Adds required packages |
| **Set Environment** | ENV | Sets environment variables |
| **Expose Ports** | EXPOSE | Declares ports to open |
| **Set Entry Command** | CMD / ENTRYPOINT | What to run when container starts |

The result of the Dockerfile lifecycle is a **layered image** which you can run as a container using docker run.

**💡 Bonus: Differences Between CMD vs ENTRYPOINT**

| **CMD** | **ENTRYPOINT** |
| --- | --- |
| Provides default arguments | Defines executable |
| Can be overridden in docker run | Harder to override unless using --entrypoint |
| Best for configuration | Best for setting the actual command |

Absolutely, Sreyas! Here's the **complete Docker Workflow**, explained step-by-step — from writing code to deploying it using Docker — along with where the lifecycle elements fit in.

**🚀 Docker Workflow: Step-by-Step Guide**

This workflow shows **how to use Docker to containerize and deploy an application** — including image creation, networking, and lifecycle usage.

**⚙️ 1. Write Your Application Code**

You develop your application in any language (Java, Spring Boot, Python, Node.js, etc.). Alongside it, you prepare:

* Dockerfile
* Optional: .dockerignore
* Optional: docker-compose.yml (for multi-container apps)

**📄 2. Create Dockerfile**

A Dockerfile contains **instructions to build your custom image**.

**Sample for Spring Boot:**

FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/myapp.jar app.jar

EXPOSE 8080

ENTRYPOINT ["java", "-jar", "app.jar"]

🧠 This is where the **Dockerfile Lifecycle** happens — each instruction becomes an image layer.

**🧱 3. Build the Docker Image**

You use the Dockerfile to **build the image**:

docker build -t myapp:latest .

This creates an image locally in your Docker host.

**🧪 4. Run a Container from the Image**

You now **start a container** from the built image:

docker run -d -p 8080:8080 --name myapp-container myapp:latest

This triggers the **Container Lifecycle**:

* Create ➝ Start ➝ Running

**🔗 5. Docker Networking Setup**

By default, containers are attached to the **bridge network**. But you can:

docker network create --driver bridge mycustomnetwork

docker network connect mycustomnetwork myapp-container

Or use docker-compose.yml to define services with isolated or shared networks easily.

**🧠 6. Inspect, Log, and Debug**

Check container state:

docker ps

docker logs myapp-container

docker inspect myapp-container

Pause, unpause, stop, or restart containers based on the **lifecycle** stage you’re managing.

**🧹 7. Stop & Remove Containers/Images (Cleanup)**

docker stop myapp-container

docker rm myapp-container

docker rmi myapp:latest

This covers:

* Stop
* Remove

**🧬 8. Use Docker Compose (Optional for Microservices)**

docker-compose.yml example:

version: '3'

services:

backend:

build: .

ports:

- "8080:8080"

networks:

- app-network

db:

image: mysql:8

environment:

MYSQL\_ROOT\_PASSWORD: root

networks:

- app-network

networks:

app-network:

driver: bridge

Run using:

docker-compose up --build -d

**☁️ 9. Deploy to Cloud / CI-CD**

You can push the image to:

* **Docker Hub**:
* docker tag myapp username/myapp:latest
* docker push username/myapp:latest
* **AWS ECR / Azure / GCP**
* Use in **Kubernetes**, **ECS**, or **EC2** using Docker CLI or Compose

**📊 Summary Workflow Diagram (Text Version)**

Write Code

↓

Create Dockerfile

↓

Build Image → docker build

↓

Create & Run Container → docker run

↓

Container Lifecycle (start, pause, stop, kill, remove)

↓

Docker Networking (bridge, host, overlay, etc.)

↓

Log & Monitor → docker logs, inspect

↓

Cleanup or Deploy to Cloud/Registry

Let me know if you'd like:

* A **real working example** (like Spring Boot + MySQL + Docker Compose)
* A **diagram flowchart** (I'll generate it for you)
* Extension into **Docker + Kubernetes Workflow**

Just say the word, Sreyas!