Day 28 – Docker Networking (Complete DevOps Co

Why Docker?
Because so far, all the container concepts we have learned were demonstrated using **Docker**. Even in the last few sessions, we completely focused on Docker. So, I'll continue with Docker to help you practice container networking and related projects.

If you haven't watched the previous videos, I strongly recommend going back. We are following a structured hierarchy that will help you not only with **Docker** but also with **Kubernetes** later on.

Why Networking in Docker?

The Need

Networking allows:

- **Containers to communicate with each other**
- **Containers to communicate with the host system**

Basic Example

- You have a **host** (could be an EC2 instance, physical server, or local machine).
- On top of the host, you install **Docker**.
- You run multiple containers:
    - **Container 1** (Frontend)
    - **Container 2** (Backend or Finance)

Two Scenarios

1. **Communication Needed**:
    - Frontend container must talk to the Backend container.
    - A network must exist to enable communication between container IPs.
2. **Isolation Needed**:
    - Example: Login container vs. Finance container.
    - Finance stores sensitive data (credit card info, user details).
    - Login container should not access Finance container.
    - We need **logical isolation** between them.

👉 Docker Networking provides solutions for **both scenarios**.

Networking in Virtual Machines vs. Containers

- **Virtual Machines (VMs):**
    - Each VM has its **own OS + Applications**.
    - You can assign **different subnets** (e.g., `172.16.x.x` vs. `172.18.x.x`).
    - By default, isolation is stronger.
- **Containers:**
    - Lightweight, don’t have full OS.
    - Must rely on Docker’s networking mechanisms for communication or isolation.

How a Container Talks to the Host

Default Setup

- Host has a network interface (`eth0`) → Example: `192.168.3.4`
- Container has another subnet → Example: `172.17.0.2`

By default, **containers cannot ping the host** because they are in **different subnets**.

The Solution → Bridge Network

- Docker creates a **virtual Ethernet interface** called **docker0**.
- Acts as a **bridge** between host and container networks.
- This is why the **default Docker network = Bridge Network**.

⚠️ If you delete or disable `docker0`, containers cannot talk to the host, meaning applications inside containers won't be reachable.

Types of Docker Networking

1. Bridge Networking (Default)

- Containers get IPs from a separate subnet.
- Communication happens via **docker0 bridge**.
- Advantage: Isolation between host and containers.
- Disadvantage: All containers share the same bridge, which may create a **common attack path**.

2. Host Networking

- Containers share the **host network directly**.
- Example: Host = `192.168.3.4`, Container = `192.168.3.6`.
- No virtual bridge; communication is direct.

- Advantage: Fast, direct access.
- Disadvantage: **Insecure** – no isolation between host and containers.

3. Overlay Networking

- Used in **multi-host environments** (e.g., Docker Swarm, Kubernetes).
- Creates a virtual network spanning multiple hosts.
- Useful for clusters but **overkill for standalone Docker setups**.

Problem with Default Bridge Networking

- All containers share **docker0**.
- Example:
  - Login container + Finance container both use the same bridge.
  - A malicious user from Login container could try to access Finance container.

👉 Not secure enough for sensitive workloads.

Solution → Custom Bridge Networks

- Docker allows creating **custom bridge networks**.
- This lets you **separate containers logically**.
- Example:
  - Login + Logout containers → Default bridge (`docker0`)
  - Finance container → Custom secure bridge (`secure_network`)

Result: Finance is isolated and cannot be reached by Login/Logout containers.

Practical Demo

Step 1 – Run Containers with Default Bridge

```
docker run -d --name login nginx
docker run -d --name logout nginx
```

Inspect networks:

```
docker inspect login
docker inspect logout
```

Both containers are on the same **bridge network**, so they can **ping each other**.

Step 2 – Create Custom Bridge Network

```
docker network create secure_network
docker network ls
```

Step 3 – Run Secure Container

```
docker run -d --name finance --network secure_network nginx
```

Inspect Finance container:

```
docker inspect finance
```

- Notice: Network = `secure_network`
- IP range different from `bridge` containers.
- Login/Logout cannot ping Finance → **Isolation achieved** ✅

Step 4 – Host Networking Example

```
docker run -d --name host_demo --network host nginx
docker inspect host_demo
```

- Network = `host`
- No separate container IP (shares host's IP).
- Insecure but direct.

Summary

- **Bridge Networking**: Default, creates `docker0`, supports communication but not secure enough.
- **Host Networking**: Shares host's network, faster but insecure.
- **Overlay Networking**: For clusters (Swarm/Kubernetes).
- **Custom Bridge Networks**: Best way to **achieve isolation** inside Docker.

Next Steps

In the next session:

- We will discuss **Docker interview questions**.
- After that, we'll move on to **Kubernetes networking**, where many of these challenges are solved (scaling, auto-healing, service discovery, etc.).

✅ If you found this useful:

- **Like the video**
- **Comment with questions**
- **Share with your friends & colleagues**

Thank you!
See you in the next session. 🙌