Docker Hands-On Lab 7 — Docker Networking Deep Dive

Trainer: Sandeep Kumar Sharma

Duration: 60–90 minutes

Goal: In this lab, you'll dive deep into **Docker Networking**. You'll understand how Docker networking works under the hood, learn about network drivers, create custom networks, work with CIDR ranges, and see what happens when using host networks or overlapping CIDRs. You'll create networks named **sandeep-network** and containers named **sandeep-app1**, **sandeep-app2**, etc.

---

1) Understanding Docker Networking

Docker networking allows containers to communicate with each other and with the outside world. Each container has its own **network namespace**, including IP addresses, routing tables, and interfaces.

By default, Docker provides three main network types:

Network	Description
bridge	Default network for standalone containers. Containers can communicate using their container names.
host	Removes network isolation and uses the host's network stack directly.
none	Disables all networking.

---

2) Viewing Existing Networks

List all Docker networks:

```
docker network ls
```

You'll see something like:

```
NETWORK ID     NAME      DRIVER    SCOPE
123abc456def   bridge    bridge    local
```

```
456def789ghi   host      host      local
789ghi012jkl   none      null      local
```

Inspect a network:

```
docker network inspect bridge
```

This command shows network details including subnet, gateway, and connected containers.

---

3) Creating a Custom Bridge Network

Let's create a custom network named **sandeep-network** using the default bridge driver.

```
docker network create sandeep-network
```

Verify the network:

```
docker network ls
```

Inspect it:

```
docker network inspect sandeep-network
```

---

4) Creating Containers in the Custom Network

Now, create two containers **sandeep-app1** and **sandeep-app2** on the same network.

```
docker run -dit --name sandeep-app1 --network sandeep-network ubuntu:latest /
bin/bash
docker run -dit --name sandeep-app2 --network sandeep-network ubuntu:latest /
bin/bash
```

Check both containers' IPs:

```
docker exec -it sandeep-app1 ip addr show eth0
docker exec -it sandeep-app2 ip addr show eth0
```

Test connectivity:

```
docker exec -it sandeep-app1 ping -c 3 sandeep-app2
```

If successful, it means containers on the same custom network can communicate by name.

---

5) Creating a Custom Network with a Custom CIDR Range

Now let's create a custom network with a **specific subnet and gateway**.

```
docker network create --driver bridge
  --subnet 192.168.55.0/24
  --gateway 192.168.55.1
  sandeep-network-custom
```

Inspect the new network:

```
docker network inspect sandeep-network-custom
```

Create a container and attach it to this custom network:

```
docker run -dit --name sandeep-app3 --network sandeep-network-custom
ubuntu:latest /bin/bash
```

Check IP address:

```
docker exec -it sandeep-app3 ip addr show eth0
```

You'll see an IP from the 192.168.55.0/24 range.

---

6) Using Custom Network Drivers

You can specify a **driver** when creating a network. Docker supports multiple drivers: - **bridge** (default) - **host** - **overlay** (used in Swarm mode) - **macvlan** (used for assigning MAC addresses directly on the physical network)

Example: Create a new network with a custom bridge driver.

```
docker network create --driver bridge sandeep-bridge-net
```

Inspect to verify:

```
docker network inspect sandeep-bridge-net
```

---

7) What Happens If You Use an Overlapping CIDR Range?

If you create two networks with the **same CIDR range**, Docker will throw an error:

```
Error response from daemon: Pool overlaps with other one on this address space
```

This ensures that container IPs don't conflict.

---

8) Creating and Testing a Host Network

When using the **host** network driver, containers share the same network stack as the host machine — no IP isolation.

Create a container in the host network:

```
docker run -dit --name sandeep-app4 --network host ubuntu:latest /bin/bash
```

Check the container's IP:

```
docker exec -it sandeep-app4 ip addr show
```

You'll notice it's using the **same IP** as your host machine.

Effect: - No NAT (Network Address Translation) - Container can access host ports directly - Multiple containers cannot bind to the same host port simultaneously

---

9) Testing Communication Between Networks

Containers on different bridge networks **cannot communicate** by default. Try it:

```
docker exec -it sandeep-app1 ping -c 3 sandeep-app3
```

You'll get: `ping: sandeep-app3: Name or service not known`

To connect them:

```
docker network connect sandeep-network-custom sandeep-app1
```

Now try again:

```
docker exec -it sandeep-app1 ping -c 3 sandeep-app3
```

It should now work.

---

10) Removing Networks

```
docker network rm sandeep-network sandeep-network-custom sandeep-bridge-net
```

If containers are still attached, stop and remove them first:

```
docker stop sandeep-app1 sandeep-app2 sandeep-app3 sandeep-app4
docker rm sandeep-app1 sandeep-app2 sandeep-app3 sandeep-app4
```

Then remove the networks again.

---

11) Practice Tasks

1. Create a new network named `sandeep-dev` with subnet `10.10.0.0/24` and test container communication.
2. Try to create another network with the same subnet — note the error.
3. Create containers in both `bridge` and `host` networks and compare their IP behavior.
4. Connect and disconnect containers between multiple networks.

---

12) Summary

- Docker networks allow container communication and connectivity to the outside world.
- The **bridge** network provides isolated container communication.

- The **host** network shares the host's network stack (no isolation).
- You can create **custom networks** with custom CIDRs and gateways.
- Overlapping CIDRs are not allowed.
- Containers on different networks cannot talk until connected.

✅ **Checkpoint:** You have successfully explored Docker networking concepts including bridge, host, and custom networks using containers **sandeep-app1**, **sandeep-app2**, **sandeep-app3**, and **sandeep-app4**!