

Lab Exercise 3- Working with Docker Networking

Step 1: Understanding Docker Default Networks

Docker provides three default networks:

- bridge: The default network when a container starts.
- host: Bypasses Docker's network isolation and attaches the container directly to the host network.
- none: No networking is available for the container.

1.1. Inspect Default Networks

Check Docker's default networks using:

```
docker network ls
```

```
[mohdanas@Mohds-MacBook-Air ~ % docker network ls

NETWORK ID      NAME      DRIVER      SCOPE
cb95aec1e635   bridge    bridge      local
0a34c112a331   host      host       local
bbd3b5b30427   none     null       local
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```

This command will show detailed information about the bridge network, including the connected containers and IP address ranges.

Step 2: Create and Use a Bridge Network

2.1. Create a User-Defined Bridge Network

A user-defined bridge network allows containers to communicate by name instead of IP.

```
docker network create my_bridge
```

```
mohdanas@Mohds-MacBook-Air ~ % docker network create my_bridge  
9ea6a0a689d2968352ea80d621aab573cce275e8a87260a650ddcae17b91291b
```

2.2. Run Containers on the User-Defined Network

Start two containers on the newly created my_bridge network:

```
docker run -dit --name container1 --network my_bridge busybox
```

```
docker run -dit --name container2 --network my_bridge busybox
```

```
[mohdanas@Mohds-MacBook-Air ~ % docker run -dit --name container1 --network my_bridge busybox  
Unable to find image 'busybox:latest' locally  
latest: Pulling from library/busybox  
b85757a5ca1a: Pull complete  
Digest: sha256:e226d6308690dbe282443c8c7e57365c96b5228f0fe7f40731b5d84d37a06839  
Status: Downloaded newer image for busybox:latest  
470679e8152ff4a170c708923058c1979677619f773b08d54f7a7594b0b91b51  
[mohdanas@Mohds-MacBook-Air ~ % docker run -dit --name container2 --network my_bridge busybox  
db6f1e958c4692c607273e3ca6e12a019ad8aa3a32b70f4a854513a6cc00441e
```

2.3. Test Container Communication

Execute a ping command from container1 to container2 using container names:

```
docker exec -it container1 ping container2
```

```
[mohdanas@Mohds-MacBook-Air ~ % docker exec -it container1 ping container2  
PING container2 (172.18.0.3): 56 data bytes  
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.111 ms  
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.266 ms  
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.237 ms  
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.193 ms  
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.153 ms  
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.108 ms  
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.149 ms  
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.195 ms  
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.151 ms  
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.160 ms  
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.198 ms  
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.190 ms  
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.149 ms  
^C  
--- container2 ping statistics ---  
13 packets transmitted, 13 packets received, 0% packet loss  
round-trip min/avg/max = 0.108/0.173/0.266 ms
```

The containers should be able to communicate since they are on the same network.

Step 3: Disconnect and Remove Networks

3.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

```
docker network disconnect my_bridge container1
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

Step 4: Clean Up

Stop and remove all containers created during this exercise:

```
docker rm -f container1 container2
```

```
[mohdanas@Mohds-MacBook-Air ~ % docker rm -f container1 container2
container1
container2
```