

# 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
```

```
C:\Users\namit>docker network ls
NETWORK ID          NAME       DRIVER  SCOPE
5ebc592e087a        bridge    bridge  local
8a6cd3284ae2        host      host    local
7d3d68c67357        none     null    local

C:\Users\namit>
```

### 1.2. Inspect the Bridge Network

```
docker network inspect bridge
```

```
Command Prompt
C:\Users\namit>docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "5ebc592e087a13ba4e33c42451ee2bcf8cce53a0acfbc444013cd9b16b8b530b",
    "Created": "2026-02-08T18:52:44.904966136Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv4": true,
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
C:\Users\namit>
```

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
```

```
C:\Users\namit>docker network create my_bridge
4eacbcc25eab6a18356cb1053b555137f6672dd056131595c8fdf2bdf3e4c237

C:\Users\namit>
```

## 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
```

```
C:\Users\namit>docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
61dfb50712f5: Pull complete
Digest: sha256:b3255e7dfbcd10cb367af0d409747d511aeb66dfac98cf30e97e87e4207dd76f
Status: Downloaded newer image for busybox:latest
b78f0c0137384c2b2b6c5ec109a0ac834ca0ca602e1960230f1f8f24c9ed3170

C:\Users\namit>docker run -dit --name container2 --network my_bridge busybox
f21777137710eb8c66e2d7b5620ff8de6894acf00db7cbd51b6e975be907c7cf

C:\Users\namit>
```

## 2.3. Test Container Communication

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

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

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

```
Command Prompt
C:\Users\namit>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=0.236 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.199 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.296 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.182 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.243 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.209 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.194 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.180 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.140 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.300 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.234 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.206 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.229 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.206 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.195 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.137 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.205 ms
64 bytes from 172.18.0.3: seq=17 ttl=64 time=0.321 ms
64 bytes from 172.18.0.3: seq=18 ttl=64 time=0.204 ms
64 bytes from 172.18.0.3: seq=19 ttl=64 time=0.188 ms
64 bytes from 172.18.0.3: seq=20 ttl=64 time=0.213 ms
64 bytes from 172.18.0.3: seq=21 ttl=64 time=0.174 ms
64 bytes from 172.18.0.3: seq=22 ttl=64 time=0.208 ms
64 bytes from 172.18.0.3: seq=23 ttl=64 time=0.188 ms
64 bytes from 172.18.0.3: seq=24 ttl=64 time=0.259 ms
64 bytes from 172.18.0.3: seq=25 ttl=64 time=0.248 ms
64 bytes from 172.18.0.3: seq=26 ttl=64 time=0.181 ms
64 bytes from 172.18.0.3: seq=27 ttl=64 time=0.214 ms
64 bytes from 172.18.0.3: seq=28 ttl=64 time=0.198 ms
64 bytes from 172.18.0.3: seq=29 ttl=64 time=0.215 ms
64 bytes from 172.18.0.3: seq=30 ttl=64 time=0.198 ms
64 bytes from 172.18.0.3: seq=31 ttl=64 time=0.209 ms
64 bytes from 172.18.0.3: seq=32 ttl=64 time=0.234 ms
64 bytes from 172.18.0.3: seq=33 ttl=64 time=0.384 ms
64 bytes from 172.18.0.3: seq=34 ttl=64 time=0.230 ms
64 bytes from 172.18.0.3: seq=35 ttl=64 time=0.212 ms
64 bytes from 172.18.0.3: seq=36 ttl=64 time=0.206 ms
64 bytes from 172.18.0.3: seq=37 ttl=64 time=0.238 ms
64 bytes from 172.18.0.3: seq=38 ttl=64 time=0.198 ms
```

## Step 3: Disconnect and Remove Networks

### 3.1. Disconnect Containers from Networks

To disconnect container1 from my\_bridge:

```
docker network disconnect my_bridge container1
```

```
--- container2 ping statistics ---
91 packets transmitted, 91 packets received, 0% packet loss
round-trip min/avg/max = 0.131/0.217/0.968 ms

C:\Users\namit>docker network disconnect my_bridge container1

C:\Users\namit>
```

```
C:\Users\namit>docker network disconnect my_bridge container2

C:\Users\namit>
```

#### 4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
C:\Users\namit>docker network rm my_bridge
my_bridge

C:\Users\namit>
```

#### Step 4: Clean Up

Stop and remove all containers created during this exercise:

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

```
C:\Users\namit>docker rm -f container1 container2
container1
container2

C:\Users\namit>
```

