

# Lab Exercise 3- Working with Docker Networking

Name Viraj bhidola

Sap 500121825

B2 Devops

## 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

```
PS C:\Users\ASUS> docker network ls
NETWORK ID          NAME       DRIVER      SCOPE
5db3989cb25d        bridge    bridge      local
3e643fb3cddf        host      host        local
77ae9d2f3372        none     null        local
PS C:\Users\ASUS> |
```

## 1.2. Inspect the Bridge Network

docker network inspect bridge

```
PS C:\Users\ASUS> docker network inspect bridge
[
  {
    "Name": "bridge",
    "Id": "5db3989cb25d33338bd9be290d2f83f5779ed6e3666c01150787827c14ef6435",
    "Created": "2026-02-02T13:20:32.036884652Z",
    "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": {}
  }
]
PS C:\Users\ASUS> |
```

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

```
]
PS C:\Users\ASUS> docker network create my_bridge
5895a1cafca5b7c2f5ae9534eb737849191fdde14dbbe9b7f787de60f9c80415
PS C:\Users\ASUS> |
```

## 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
```

```
PS C:\Users\ASUS> 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:e226d6308690dbe282443c8c7e57365c96b5228f0fe7f40731b5d84d37a06839
Status: Downloaded newer image for busybox:latest
4d529b4b01df0eb630f4c3595bb3d0e27b0ea7adc74a5bc7c7b454437f7559
PS C:\Users\ASUS> docker run -dit --name container2 --network my_bridge busybox
4598e2c30567707ea60a0b47fdf73a649af9d964b5c32f18087a662faf8691d0
PS C:\Users\ASUS> |
```

## 2.3. Test Container Communication

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

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

```

PS C:\Users\ASUS> 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=0.098 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.071 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.063 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.066 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.063 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.074 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.049 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.059 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.060 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.063 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.069 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.066 ms
64 bytes from 172.18.0.3: seq=16 ttl=64 time=0.071 ms
64 bytes from 172.18.0.3: seq=17 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=18 ttl=64 time=0.059 ms
64 bytes from 172.18.0.3: seq=19 ttl=64 time=0.067 ms
64 bytes from 172.18.0.3: seq=20 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=21 ttl=64 time=0.065 ms
64 bytes from 172.18.0.3: seq=22 ttl=64 time=0.068 ms
64 bytes from 172.18.0.3: seq=23 ttl=64 time=0.066 ms
64 bytes from 172.18.0.3: seq=91 ttl=64 time=0.071 ms
64 bytes from 172.18.0.3: seq=92 ttl=64 time=0.081 ms
64 bytes from 172.18.0.3: seq=93 ttl=64 time=0.149 ms
64 bytes from 172.18.0.3: seq=94 ttl=64 time=0.133 ms
^C
--- container2 ping statistics ---
95 packets transmitted, 95 packets received, 0% packet loss
round-trip min/avg/max = 0.049/0.073/0.172 ms
PS C:\Users\ASUS>

```

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
```

```
PS C:\Users\ASUS> docker network disconnect my_bridge container1  
PS C:\Users\ASUS> |
```

## 4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

```
PS C:\Users\ASUS> docker network rm my_bridge  
PS C:\Users\ASUS> docker network rm my_bridge  
Error response from daemon: error while removing network: network my_bridge has active endpoints (name:"container2" id:"1351adf6d123")  
exit status 1  
PS C:\Users\ASUS> |
```

## Step 4: Clean Up

Stop and remove all containers created during this exercise:

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

```
exit status 1  
PS C:\Users\ASUS> docker rm -f container1 container2  
container1  
container2  
PS C:\Users\ASUS> |
```