

# Lab Exercise 4- Working with Docker Networking

## Step 1: Understanding Docker Default Networks

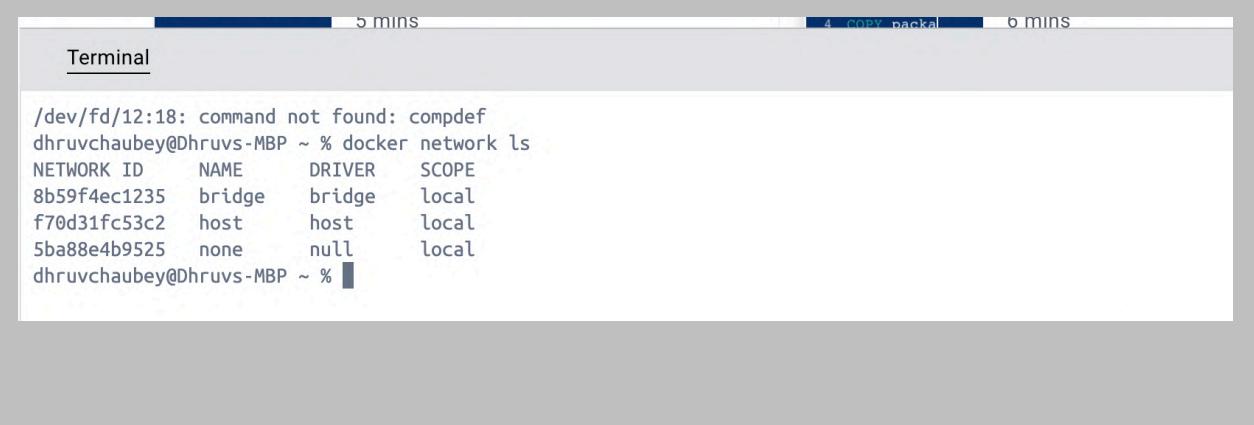
Docker provides three default networks:

- 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
```



```
5 mins 4 COPY packal 6 mins
Terminal

/dev/fd/12:18: command not found: compdef
dhruvchaubey@Dhruvs-MBP ~ % docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
8b59f4ec1235   bridge    bridge      local
f70d31fc53c2   host      host      local
5ba88e4b9525   none      null      local
dhruvchaubey@Dhruvs-MBP ~ %
```

### 1.2. Inspect the Bridge Network

```
dhruvchaubey@Dhruvs-MBP ~ % docker network inspect bridge
[{"Name": "bridge",
 "Id": "8b59f4ec1235ecf85be9b408952c620d46f78c19637ac2f14419309830f81670",
 "Created": "2024-09-09T05:42:28.384466083Z",
 "Scope": "local",
 "Driver": "bridge",
 "EnableIPv6": false,
 "IPAM": {
     "Driver": "default",
     "Options": null,
     "Config": [
         {
             "Subnet": "172.17.0.0/16",
             "Gateway": "172.17.0.1"
         }
     ]
 },
 "Internal": false,
```

docker network

inspect bridge

```
        },
        "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": "65535"
        },
        "Labels": {}
    }
]
dhruvchaubey@Dhruvs-MBP ~ %
```

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
```

```
dhruvchaubey@Dhruvs-MBP ~ % docker network create my_bridge
03d39d82a6727758aad3f7cd1f0036010474594c7a88f02445ee06343f75b13
dhruvchaubey@Dhruvs-MBP ~ % docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
8b59f4ec1235   bridge     bridge     local
f70d31fc53c2   host      host      local
03d39d82a672   my_bridge  bridge     local
5ba88e4b9525   none      null      local
dhruvchaubey@Dhruvs-MBP ~ %
```

## 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
dhruvchaubey@Dhruv-MBP ~ % docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
75e8ca8f509f: Pull complete
Digest: sha256:34b191d63fbc93e25e275bfccf1b5365664e5ac28f06d974e8d50090fbba49f41
Status: Downloaded newer image for busybox:latest
1e6fdf681bddef42e344827dda7d59e5e6ca042bd9f7526901e0ef3961275bb2

docker run -dit --name container2 --network my_bridge busybox
dhruvchaubey@Dhruv-MBP ~ % docker run -dit --name container2 --network my_bridge busybox
e1d646f9c2676ce13063817087cae139870af9c19c1d62ba80ac0563fd86019f
dhruvchaubey@Dhruv-MBP ~ %
```

## 2.3. Test Container Communication

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

```
docker exec -it container1 ping container2
dhruvchaubey@Dhruv-MBP ~ % docker run -dit --name container2 --network my_bridge busybox
e1d646f9c2676ce13063817087cae139870af9c19c1d62ba80ac0563fd86019f
dhruvchaubey@Dhruv-MBP ~ % 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=0.077 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.235 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.233 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.201 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.201 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.203 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.194 ms
```

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

## Step 3: Create and Use a Host Network

### 3.1. Run a Container Using the Host Network

The host network allows the container to use the host machine's networking stack:

```
docker run -d --name host_network_container --network host nginx
```

```
/dev/fd/12:18: command not found: compdef
dhruvchaubey@Dhruv-MBP ~ % docker run -d --name host_network_container --network host nginx
8fbc55010c70bae6fe67bebdafb137f12adc48ff062916416aa3981ad88a3827
dhruvchaubey@Dhruv-MBP ~ %
```

Access the NGINX server via localhost:80 in your browser to verify the container is using the host network.

### 3.2. Check Network

```
docker network
8fbc55010c70bae6fe67bebdafb137f12adc48ff062916416aa3981ad88a3827
dhruvchaubey@Dhruv-MBP ~ % docker network inspect host
[{"Name": "host", "Id": "f70d31fc53c296c5bce7b080f51ea588c65327bdefeabf77bc69ccb05dc48f47", "Created": "2024-08-29T05:55:27.407696166Z", "Scope": "local", "Driver": "host", "EnableIPv6": false, "IPAM": {"Driver": "default", "Options": null, "Config": null}, "Internal": false, "Attachable": false, "Ingress": false, "ConfigFrom": {"Network": ""}}, {"Name": "host_network_container", "EndpointID": "c9002737e6fc0ad2414856977b3328d83e28ddcde1e2af209b2688a5d2b32476", "MacAddress": "", "IPv4Address": "", "IPv6Address": ""}], "Options": {}, "Labels": {}}
```

## Step 4: Disconnect and Remove Networks

### 4.1. Disconnect Containers from Networks

To disconnect container1 from my\_bridge:

```
docker network disconnect my_bridge container1
```

```
dhruvchaubey@Dhruvs-MBP ~ % docker network disconnect my_bridge container1
```

To verify :

```
dhruvchaubey@Dhruvs-MBP ~ % docker network disconnect my_bridge container1
```

```
Error response from daemon: container 1e6fdf681bddef42e344827dda7d59e5e6ca042bd9f7526901e0ef3961275bb2 is not connected to network my_bridge
```

### 4.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

First we disconnected container1 and container2 from my\_bridge, only then we can remove the network.

```
dhruvchaubey@Dhruvs-MBP ~ % docker network rm my_bridge  
my_bridge  
dhruvchaubey@Dhruvs-MBP ~ %
```

## Step 5: Clean Up

Stop and remove all containers created during this exercise:

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

```
my_bridge  
dhruvchaubey@Dhruvs-MBP ~ % docker rm -f container1 container2 host_network_container  
container1  
container2  
host_network_container  
dhruvchaubey@Dhruvs-MBP ~ %
```