Containers & Docker Security LAB

SUBMITTED TO

Dr. Hitesh Kumar Sharma

SUBMITTED BY

Siddharth Agarwal

500107594

R2142220663

Btech CSE DevOps B1

Lab Exercise 4- Working with Docker Networking

Step 1: Understanding Docker Default Networks

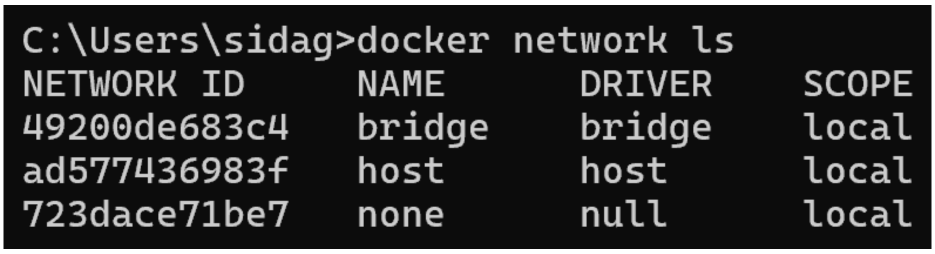
Docker provides three default networks:

- 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:

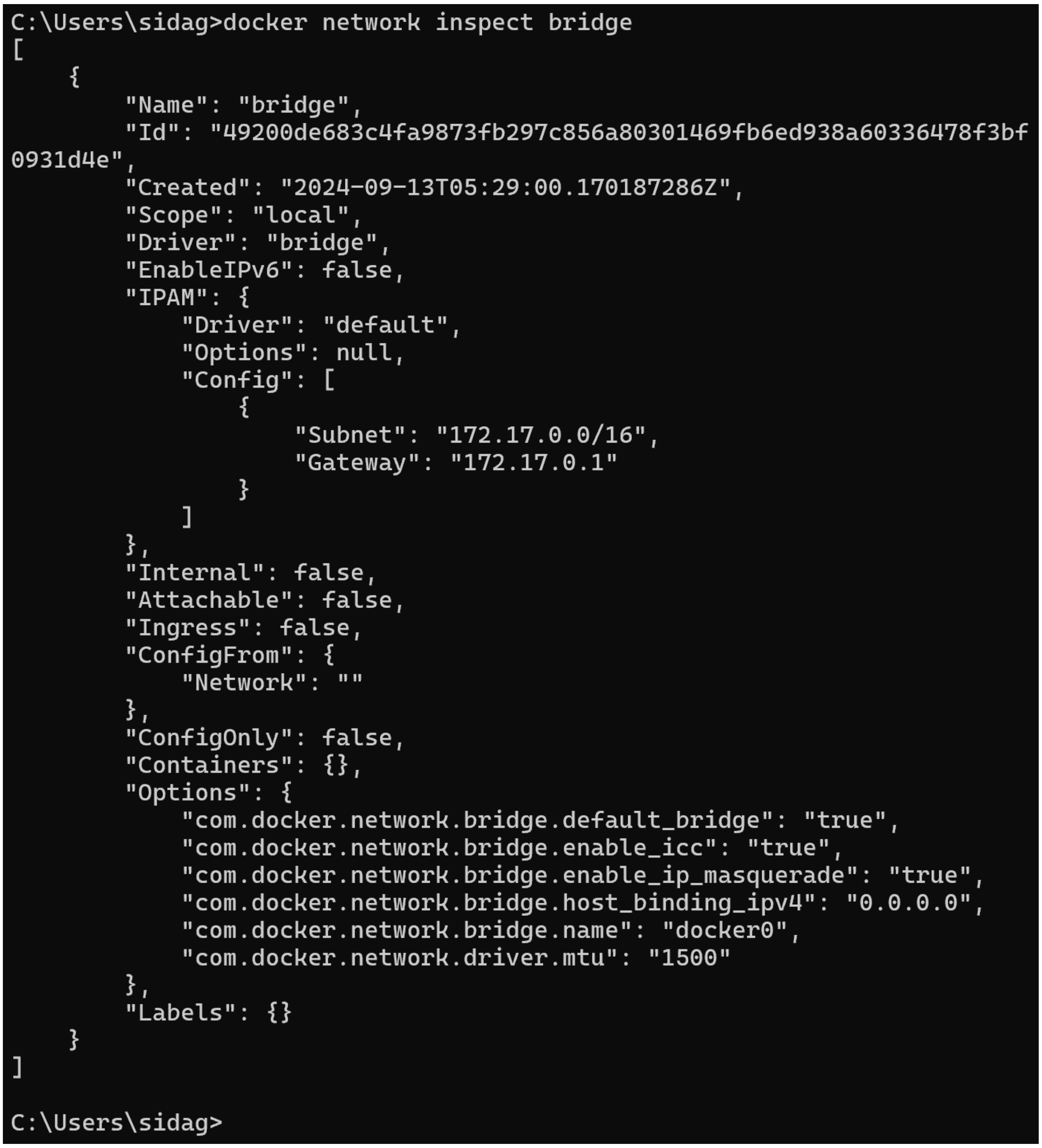
```
docker network ls
```



```
C:\Users\sidag>docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
49200de683c4   bridge    bridge    local
ad577436983f   host      host      local
723dace71be7   none      null      local
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```



```
C:\Users\sidag>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "49200de683c4fa9873fb297c856a80301469fb6ed938a60336478f3bf
0931d4e",
        "Created": "2024-09-13T05:29:00.170187286Z",
        "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,
        "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\sidag>
```

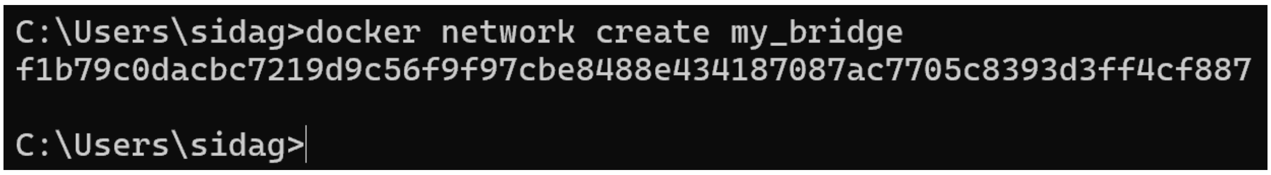
This command will show detailed information about the bridge network, including the connected containers and IP address ranges.

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
```

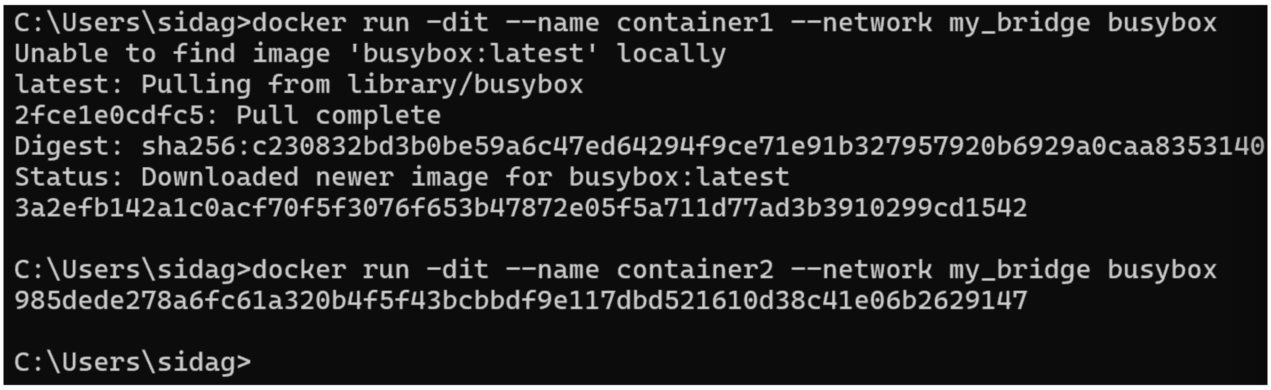


```
C:\Users\sidag>docker network create my_bridge
f1b79c0dacbc7219d9c56f9f97cbe8488e434187087ac7705c8393d3ff4cf887

C:\Users\sidag>
```

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
```



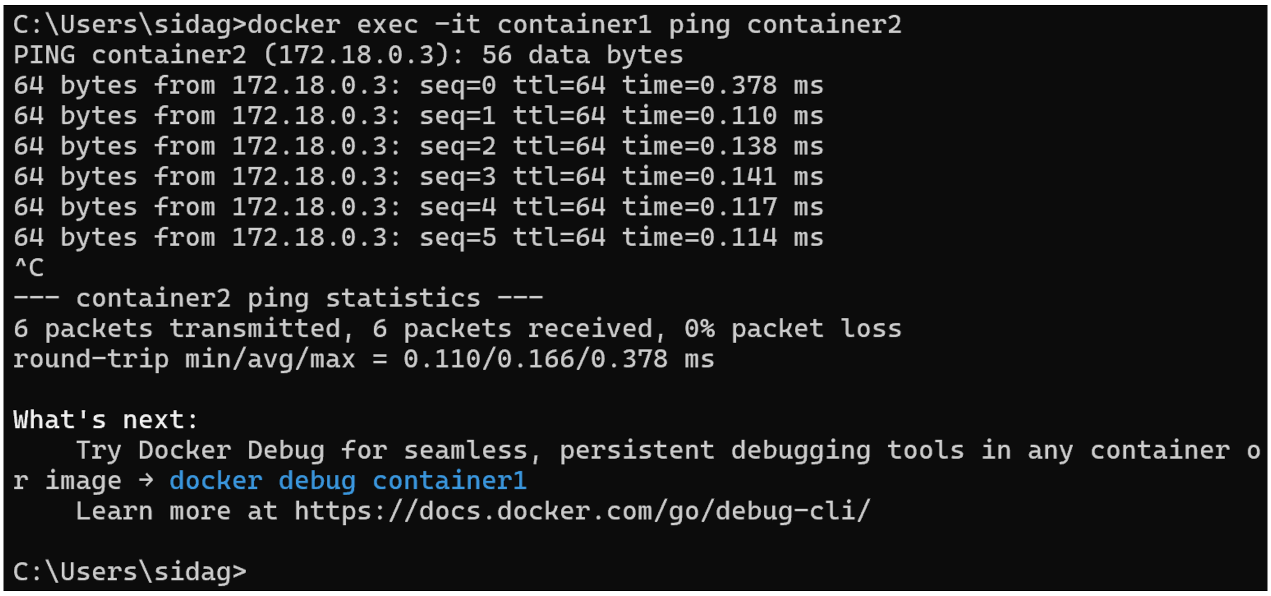
```
C:\Users\sidag>docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
2fce1e0cdfc5: Pull complete
Digest: sha256:c230832bd3b0be59a6c47ed64294f9ce71e91b327957920b6929a0caa8353140
Status: Downloaded newer image for busybox:latest
3a2efb142a1c0acf70f5f3076f653b47872e05f5a711d77ad3b3910299cd1542

C:\Users\sidag>docker run -dit --name container2 --network my_bridge busybox
985dede278a6fc61a320b4f5f43bcbbdf9e117dbd521610d38c41e06b2629147

C:\Users\sidag>
```

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.



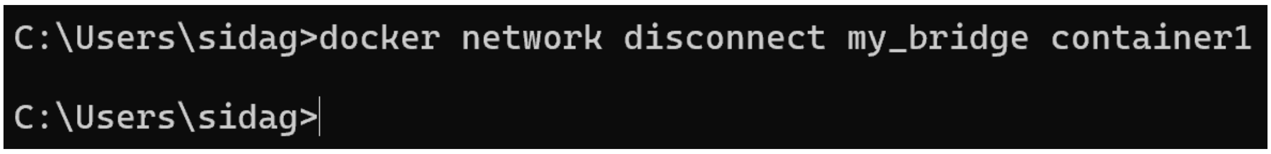
```
C:\Users\sidag>docker exec -it container1 ping container2
PING container2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.378 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.110 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.138 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.141 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.117 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.114 ms
^C
--- container2 ping statistics ---
6 packets transmitted, 6 packets received, 0% packet loss
round-trip min/avg/max = 0.110/0.166/0.378 ms

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container o
r image → docker debug container1
    Learn more at https://docs.docker.com/go/debug-cli/

C:\Users\sidag>
```

Step 3: Disconnect and Remove Networks

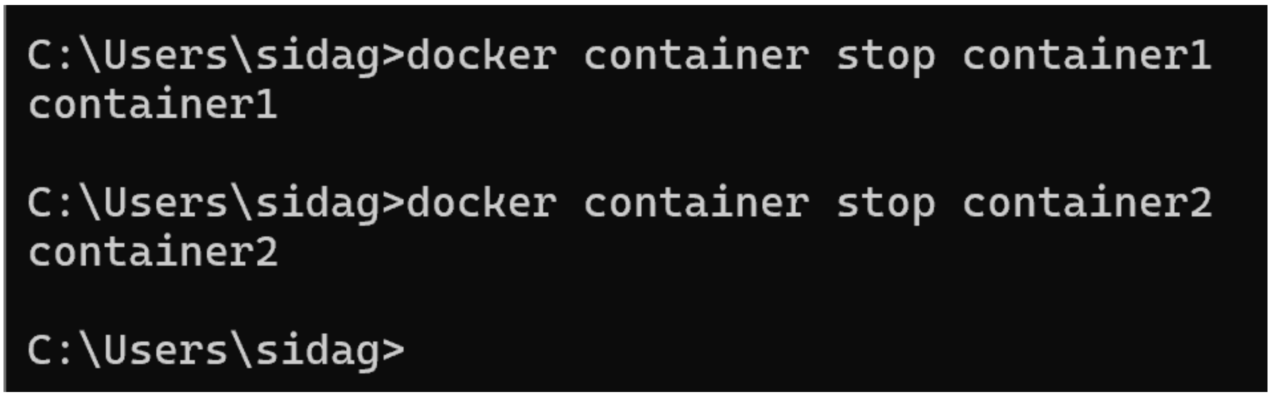
3.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

```
docker network disconnect my_bridge container1
```



```
C:\Users\sidag>docker network disconnect my_bridge container1

C:\Users\sidag>
```

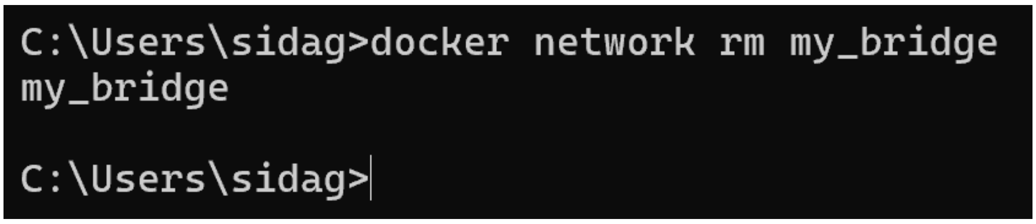


```
C:\Users\sidag>docker container stop container1
container1

C:\Users\sidag>docker container stop container2
container2

C:\Users\sidag>
```

3.2. Remove Networks

To remove the user-defined network:

```
docker network rm my_bridge
```

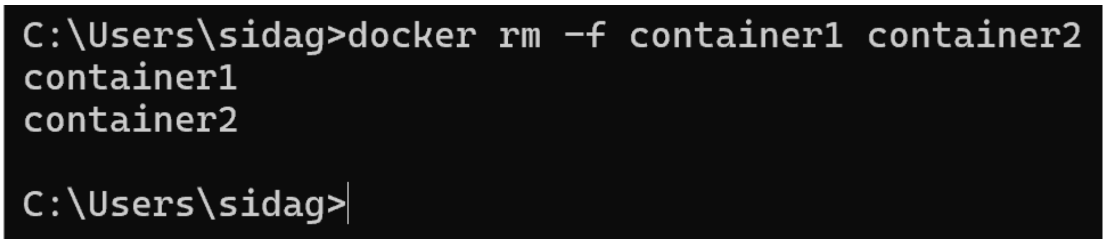


```
C:\Users\sidag>docker network rm my_bridge
my_bridge

C:\Users\sidag>
```

Step 4: Clean Up

Stop and remove all containers created during this exercise:

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



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

C:\Users\sidag>
```