Lab Exercise 3- Working with Docker Networking

Name- Misha

SAP ID- 500119679

Batch-2

Step 1: Understanding Docker Default Networks

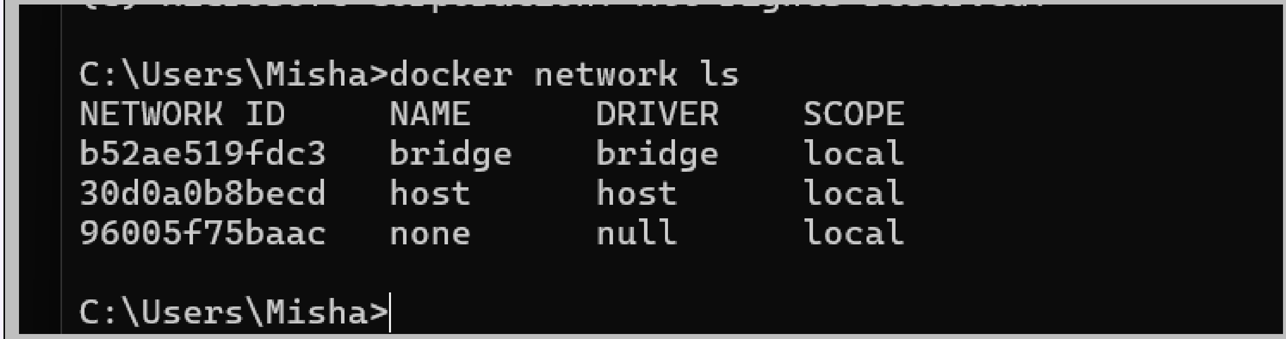
Docker provides three default networks:

- 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
```

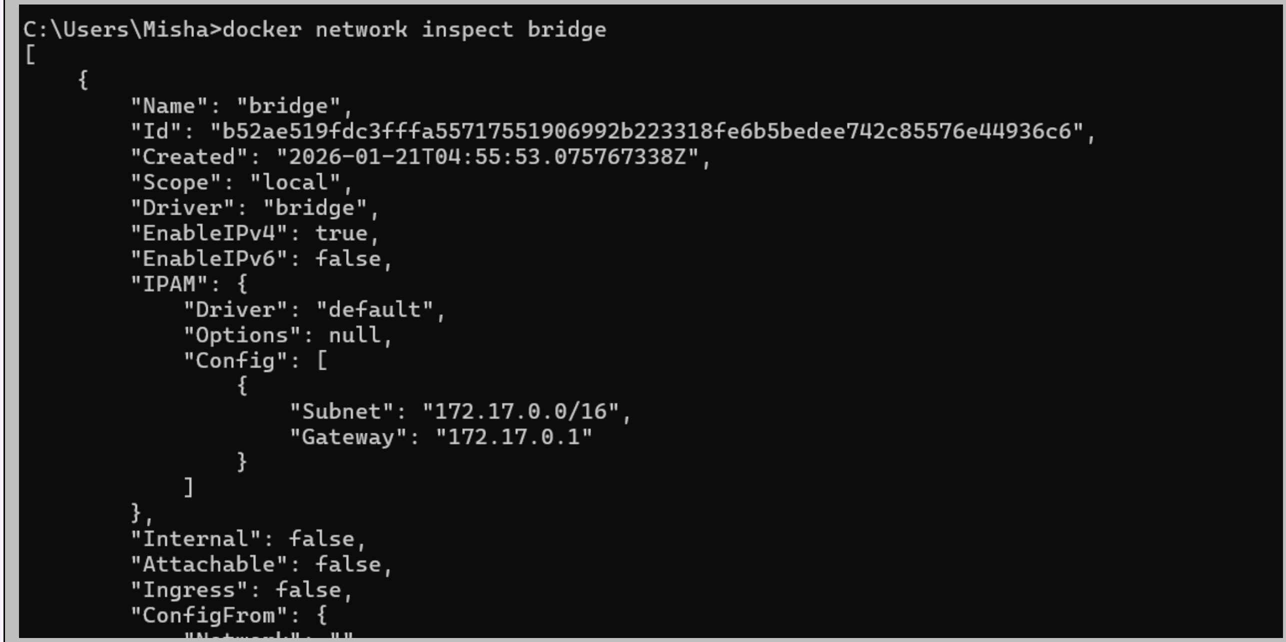


```
C:\Users\Misha>docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
b52ae519fdc3   bridge    bridge    local
30d0a0b8becd   host      host      local
96005f75baac   none      null      local

C:\Users\Misha>
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```



```
C:\Users\Misha>docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "b52ae519fdc3fffa55717551906992b223318fe6b5bedee742c85576e44936c6",
        "Created": "2026-01-21T04:55:53.075767338Z",
        "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": {
```

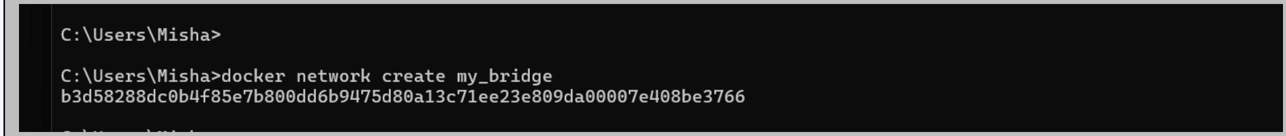
This command will show detailed information about the bridge network, including the connected containers and IP address ranges.

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
```



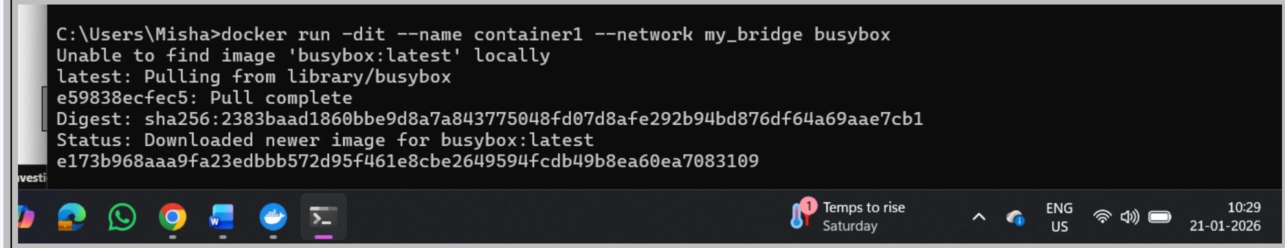
```
C:\Users\Misha>

C:\Users\Misha>docker network create my_bridge
b3d58288dc0b4f85e7b800dd6b9475d80a13c71ee23e809da00007e408be3766
```

2.2. Run Containers on the User-Defined Network

Start two containers on the newly created my_bridge network:

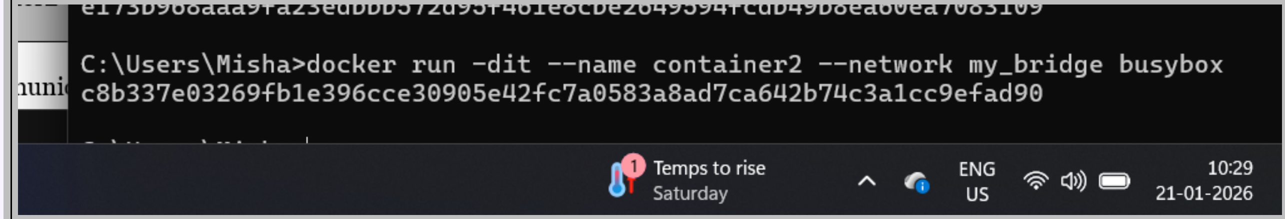
```
docker run -dit --name container1 --network my_bridge busybox
```



```
C:\Users\Misha>docker run -dit --name container1 --network my_bridge busybox
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
e59838ecfec5: Pull complete
Digest: sha256:2383baad1860bbe9d8a7a843775048fd07d8afe292b94bd876df64a69aae7cb1
Status: Downloaded newer image for busybox:latest
e173b968aaa9fa23edbbb572d95f461e8cbe2649594fcdb49b8ea60ea7083109
```

Temps to rise Saturday ENG US 10:29 21-01-2026

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



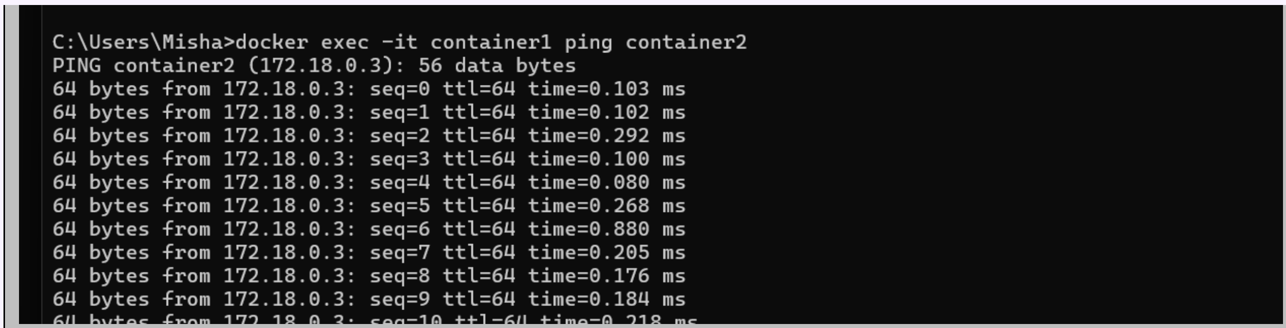
```
C:\Users\Misha>docker run -dit --name container2 --network my_bridge busybox
c8b337e03269fb1e396cce30905e42fc7a0583a8ad7ca642b74c3a1cc9efad90
```

Temps to rise Saturday ENG US 10:29 21-01-2026

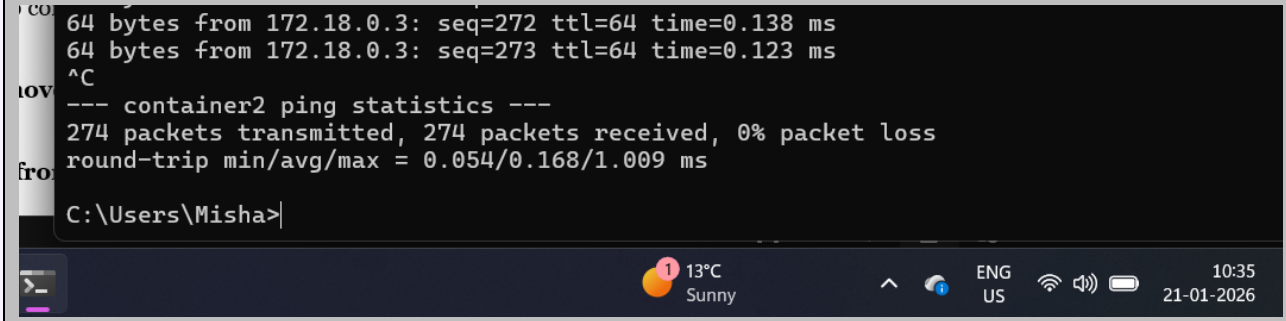
2.3. Test Container Communication

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

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



```
C:\Users\Misha>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=0.103 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.102 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.292 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.100 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.080 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.268 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.880 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.205 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.176 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.184 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.218 ms
```



```
64 bytes from 172.18.0.3: seq=272 ttl=64 time=0.138 ms
64 bytes from 172.18.0.3: seq=273 ttl=64 time=0.123 ms
^C
--- container2 ping statistics ---
274 packets transmitted, 274 packets received, 0% packet loss
round-trip min/avg/max = 0.054/0.168/1.009 ms

C:\Users\Misha>
```

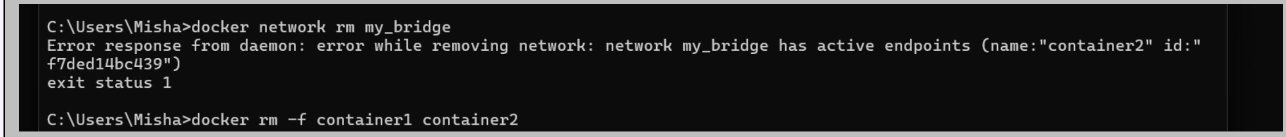
The containers should be able to communicate since they are on the same network.

Step 3: Disconnect and Remove Networks

3.1. Disconnect Containers from Networks

To disconnect container1 from my_bridge:

```
docker network disconnect my_bridge container1
```



```
C:\Users\Misha>docker network rm my_bridge
Error response from daemon: error while removing network: network my_bridge has active endpoints (name:"container2" id:"
f7ded14bc439")
exit status 1

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

4.2. Remove Networks

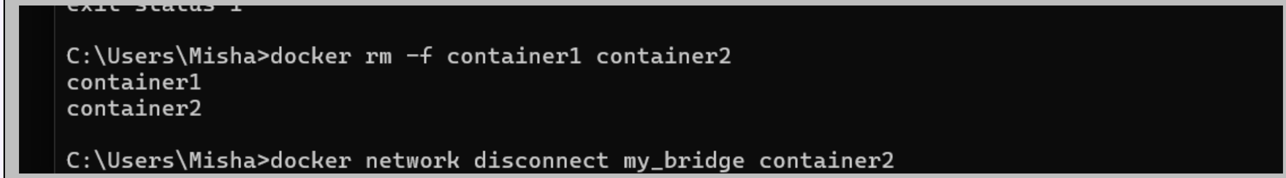
To remove the user-defined network:

```
docker network rm my_bridge
```

Step 4: Clean Up

Stop and remove all containers created during this exercise:

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



```
exit status 1

C:\Users\Misha>docker rm -f container1 container2
container1
container2

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