ANSHIKA SRIVASTAVA

ROLL NUMBER – R2142220907

SAP ID – 500107049

DEVSECOPS BATCH B1 HONS.

EXPERIMENT – 4

WORKING WITH DOCKER NETWORKING

Step 1: Understanding Docker Default Networks

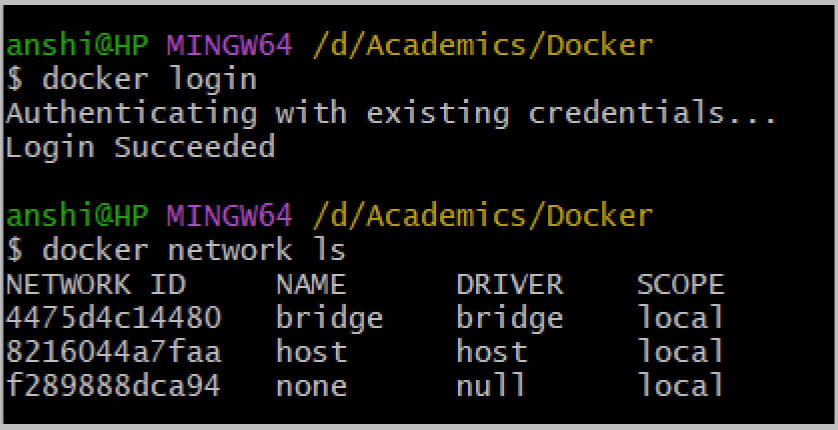
Docker provides three default networks:

- 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:

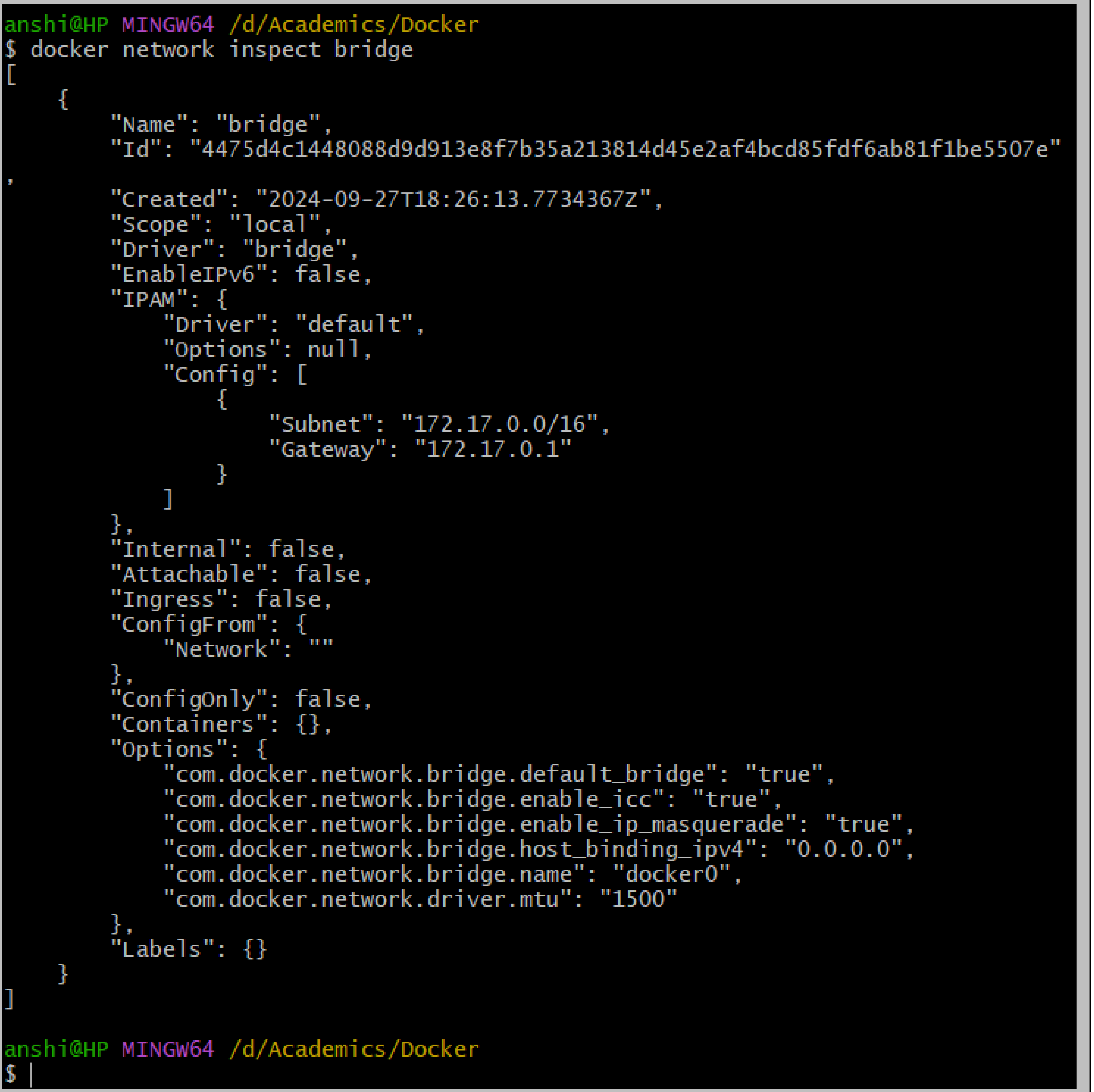
```
docker network ls
```



```
anshi@HP MINGW64 /d/Academics/Docker
$ docker login
Authenticating with existing credentials...
Login Succeeded

anshi@HP MINGW64 /d/Academics/Docker
$ docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
4475d4c14480   bridge    bridge    local
8216044a7faa   host      host      local
f289888dca94   none      null      local
```

1.2. Inspect the Bridge Network

```
docker network inspect bridge
```



```
anshi@HP MINGW64 /d/Academics/Docker
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "4475d4c1448088d9d913e8f7b35a213814d45e2af4bcd85fdf6ab81f1be5507e"
,
        "Created": "2024-09-27T18:26:13.7734367Z",
        "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": {}
    }
]

anshi@HP MINGW64 /d/Academics/Docker
$ |
```

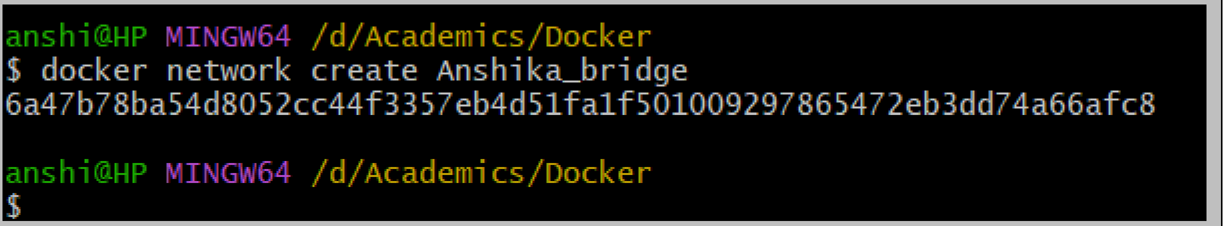
This command will show detailed information about the bridge network, including the connected containers and IP address ranges.

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 Anshika_bridge
```

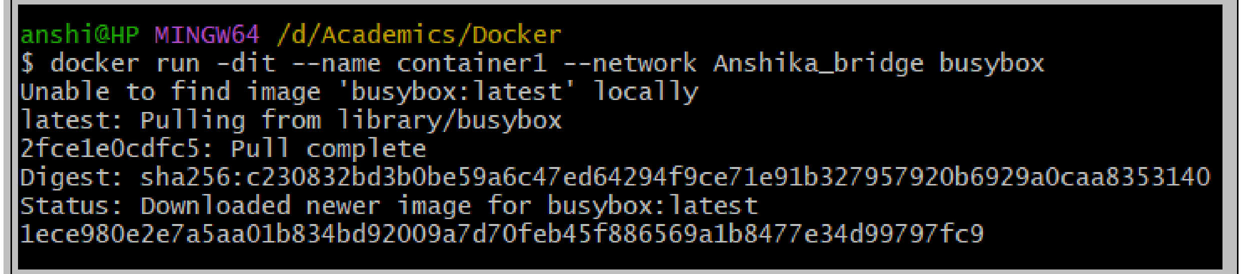


```
anshi@HP MINGW64 /d/Academics/Docker
$ docker network create Anshika_bridge
6a47b78ba54d8052cc44f3357eb4d51fa1f501009297865472eb3dd74a66afc8

anshi@HP MINGW64 /d/Academics/Docker
$
```

2.2. Run Containers on the User-Defined Network

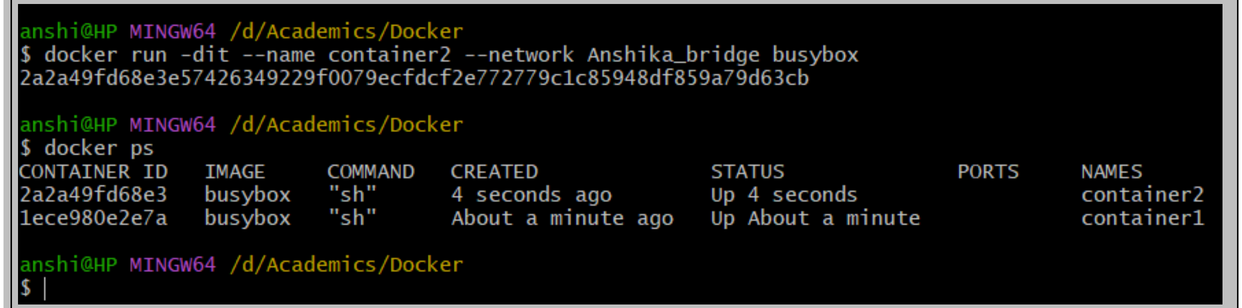
Start two containers on the newly created my_bridge network:

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



```
anshi@HP MINGW64 /d/Academics/Docker
$ docker run -dit --name container1 --network Anshika_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
1ece980e2e7a5aa01b834bd92009a7d70feb45f886569a1b8477e34d99797fc9
```

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

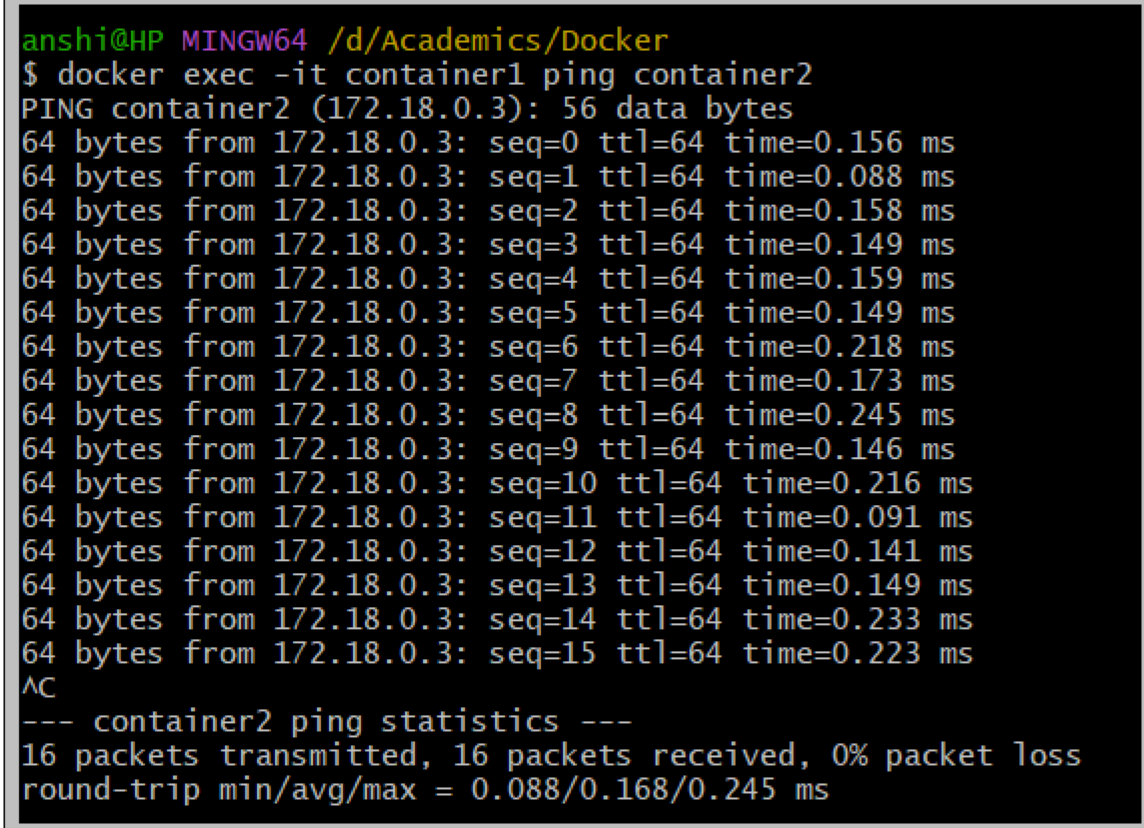


```
anshi@HP MINGW64 /d/Academics/Docker
$ docker run -dit --name container2 --network Anshika_bridge busybox
2a2a49fd68e3e57426349229f0079ecfdcf2e772779c1c85948df859a79d63cb

anshi@HP MINGW64 /d/Academics/Docker
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED              STATUS              PORTS     NAMES
2a2a49fd68e3   busybox   "sh"      4 seconds ago        Up 4 seconds                  container2
1ece980e2e7a   busybox   "sh"      About a minute ago   Up About a minute             container1

anshi@HP MINGW64 /d/Academics/Docker
$ |
```

2.3. Test Container Communication

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

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



```
anshi@HP MINGW64 /d/Academics/Docker
$ 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=0.156 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.088 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.158 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.149 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.159 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.149 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.218 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.173 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.245 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.146 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.216 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.091 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.141 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.149 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.233 ms
64 bytes from 172.18.0.3: seq=15 ttl=64 time=0.223 ms
^C
--- container2 ping statistics ---
16 packets transmitted, 16 packets received, 0% packet loss
round-trip min/avg/max = 0.088/0.168/0.245 ms
```

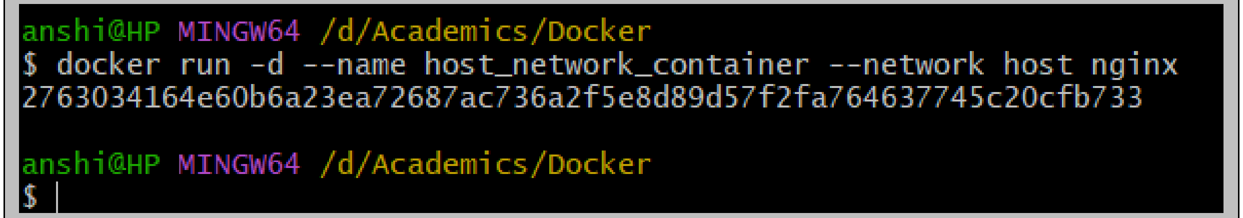
The containers should be able to communicate since they are on the same network.

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
```

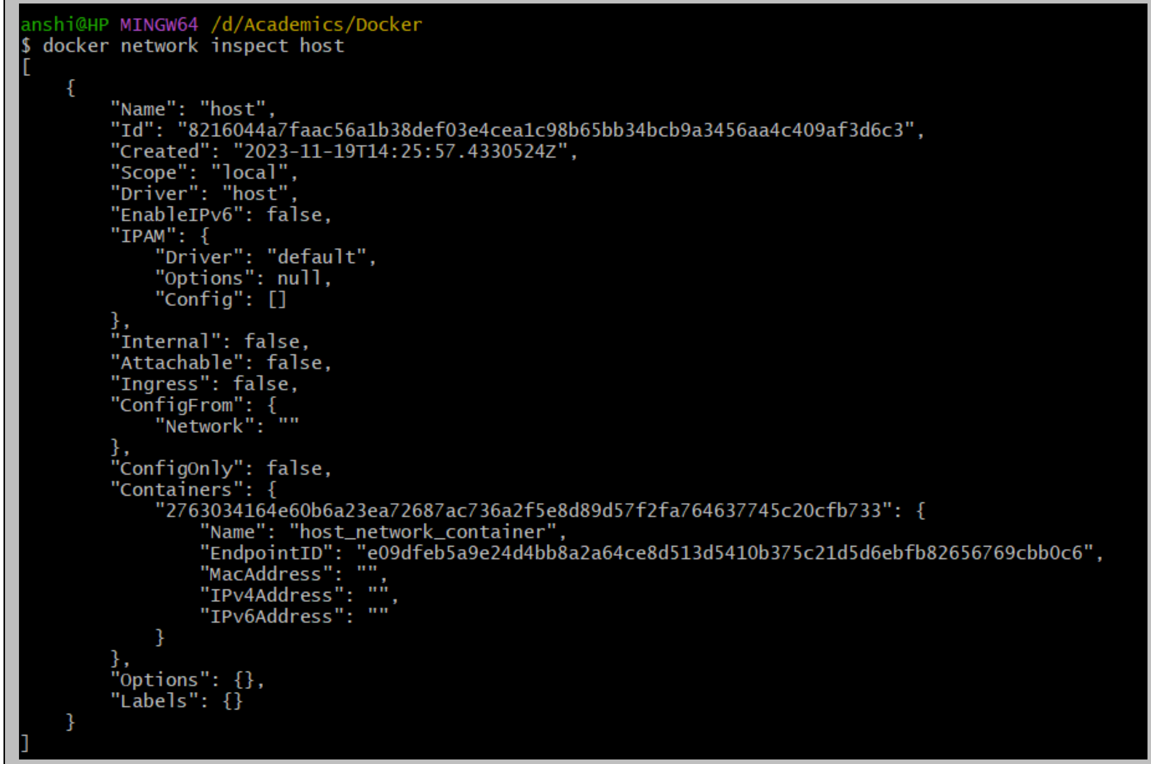


```
anshi@HP MINGW64 /d/Academics/Docker
$ docker run -d --name host_network_container --network host nginx
2763034164e60b6a23ea72687ac736a2f5e8d89d57f2fa764637745c20cfb733

anshi@HP MINGW64 /d/Academics/Docker
$ |
```

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 inspect host
```



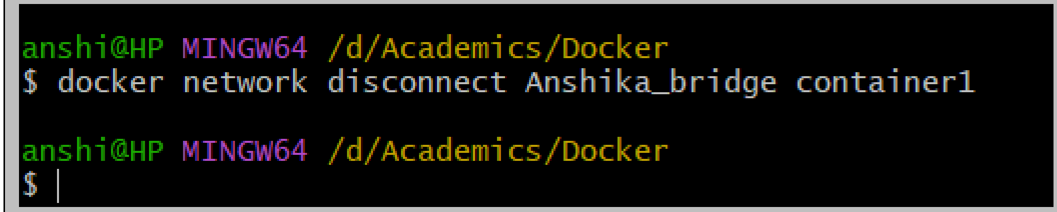
```
anshi@HP MINGW64 /d/Academics/Docker
$ docker network inspect host
[
    {
        "Name": "host",
        "Id": "8216044a7faac56a1b38def03e4cea1c98b65bb34bcb9a3456aa4c409af3d6c3",
        "Created": "2023-11-19T14:25:57.4330524Z",
        "Scope": "local",
        "Driver": "host",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": []
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "2763034164e60b6a23ea72687ac736a2f5e8d89d57f2fa764637745c20cfb733": {
                "Name": "host_network_container",
                "EndpointID": "e09dfeb5a9e24d4bb8a2a64ce8d513d5410b375c21d5d6ebfb82656769cbb0c6",
                "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 Anshika_bridge container1
```

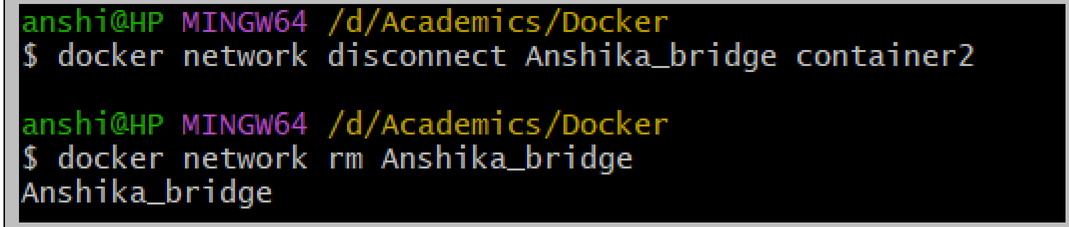


```
anshi@HP MINGW64 /d/Academics/Docker
$ docker network disconnect Anshika_bridge container1

anshi@HP MINGW64 /d/Academics/Docker
$ |
```

4.2. Remove Networks

To remove the user-defined network:

```
docker network rm Anshika_bridge
```

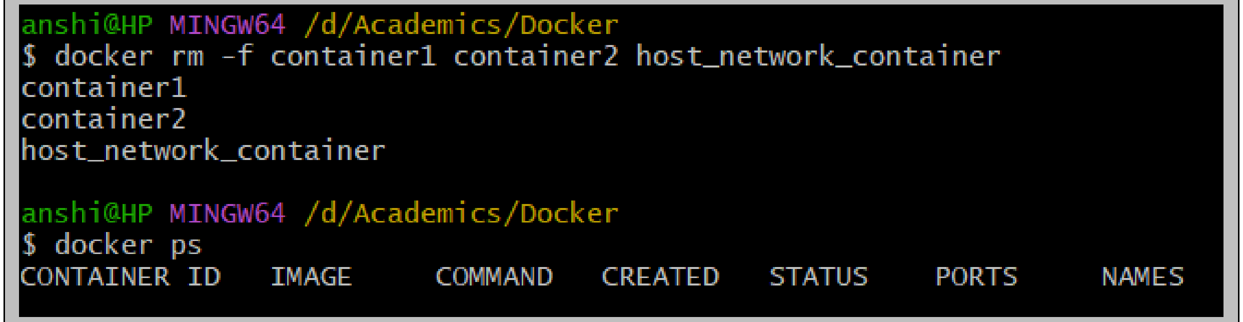


```
anshi@HP MINGW64 /d/Academics/Docker
$ docker network disconnect Anshika_bridge container2

anshi@HP MINGW64 /d/Academics/Docker
$ docker network rm Anshika_bridge
Anshika_bridge
```

Step 5: Clean Up

Stop and remove all containers created during this exercise:

docker rm -f container1 container2



```
anshi@HP MINGW64 /d/Academics/Docker
$ docker rm -f container1 container2 host_network_container
container1
container2
host_network_container

anshi@HP MINGW64 /d/Academics/Docker
$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```