Name-Anmol Yadav

SAP ID -500083814

EXPERIMENT 4

AIM: Working with Docker Network

Steps to Complete:

Step 1 - Create Network

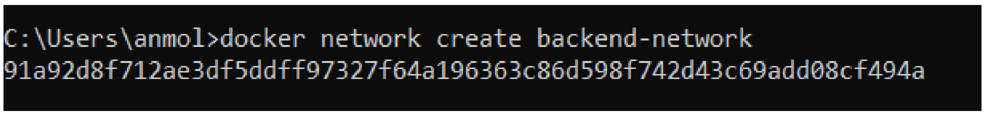
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

In this example, we're going to start by creating a *backend-network*. All containers attached to our backend will be on this network.

Task: Create Network

To start with we create the network with our predefined name.

```
docker network create backend-network
```

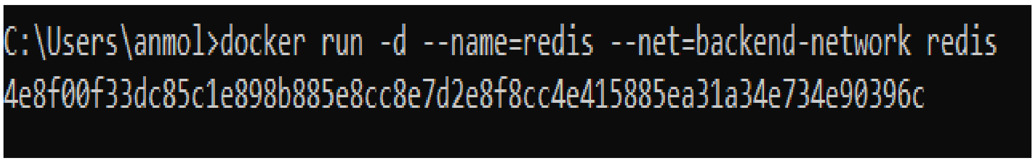


```
C:\Users\anmol>docker network create backend-network
91a92d8f712ae3df5ddff97327f64a196363c86d598f742d43c69add08cf494a
```

Task: Connect To Network

When we launch new containers, we can use the --*net* attribute to assign which network they should be connected to.

```
docker run -d --name=redis --net=backend-network redis
```



```
C:\Users\anmol>docker run -d --name=redis --net=backend-network redis
4e8f00f33dc85c1e898b885e8cc8e7d2e8f8cc4e415885ea31a34e734e90396c
```

In the next step we'll explore the state of the network.

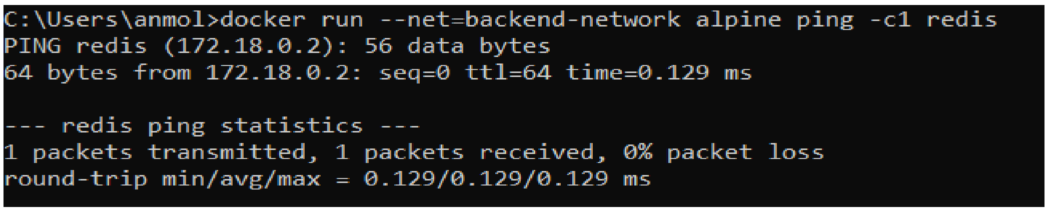
Step 2 - Network Communication

Unlike using links, *docker network* behave like traditional networks where nodes can be attached/detached.

Task: Explore

The first thing you'll notice is that Docker no longer assigns environment variables or updates the hosts file of containers. Explore using the following two commands and you'll notice it no longer mentions other containers.

```
docker run --net=backend-network alpine ping -c1 redis
```



```
C:\Users\anmol>docker run --net=backend-network alpine ping -c1 redis
PING redis (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.129 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.129/0.129/0.129 ms
```

Step 3 - Connect Two Containers

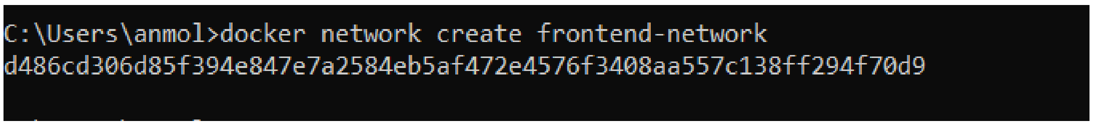
Docker supports multiple networks and containers being attached to more than one network at a time.

For example, let's create a separate network with a Node.js application that communicates with our existing Redis instance.

Task

The first task is to create a new network in the same way.

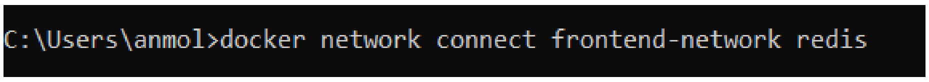
```
docker network create frontend-network
```



```
C:\Users\anmol>docker network create frontend-network
d486cd306d85f394e847e7a2584eb5af472e4576f3408aa557c138ff294f70d9
```

When using the *connect* command it is possible to attach existing containers to the network.

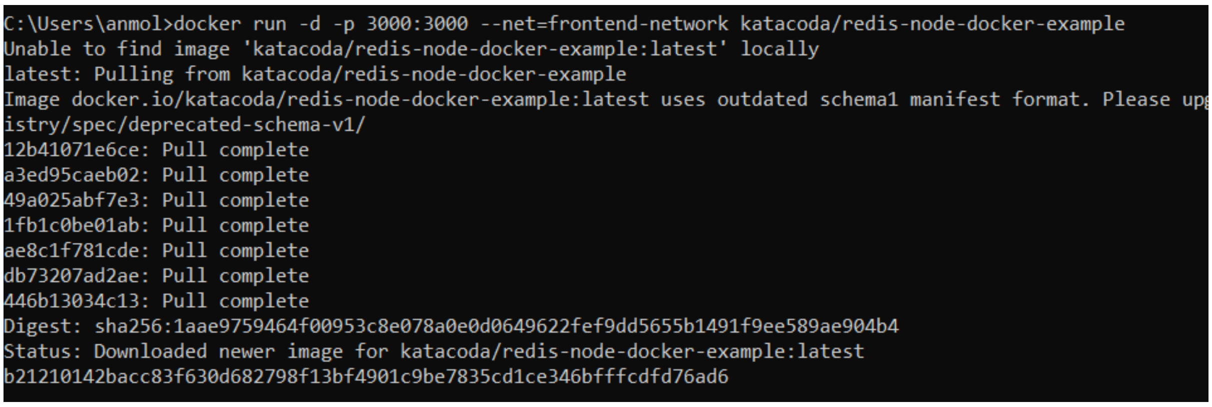
`docker network connect frontend-network redis`



```
C:\Users\anmol>docker network connect frontend-network redis
```

When we launch the web server, given it's attached to the same network it will be able to communicate with our Redis instance.

`docker run -d -p 3000:3000 --net=frontend-network katacoda/redis-node-docker-example`



```
C:\Users\anmol>docker run -d -p 3000:3000 --net=frontend-network katacoda/redis-node-docker-example
Unable to find image 'katacoda/redis-node-docker-example:latest' locally
latest: Pulling from katacoda/redis-node-docker-example
Image docker.io/katacoda/redis-node-docker-example:latest uses outdated schema1 manifest format. Please upg
istry/spec/deprecated-schema-v1/
12b41071e6ce: Pull complete
a3ed95caeb02: Pull complete
49a025abf7e3: Pull complete
1fb1c0be01ab: Pull complete
ae8c1f781cde: Pull complete
db73207ad2ae: Pull complete
446b13034c13: Pull complete
Digest: sha256:1aae9759464f00953c8e078a0e0d0649622fef9dd5655b1491f9ee589ae904b4
Status: Downloaded newer image for katacoda/redis-node-docker-example:latest
b21210142bacc83f630d682798f13bf4901c9be7835cd1ce346bfffcdfd76ad6
```

You can test it using `curl docker:3000`

Step 4 - Create Aliases

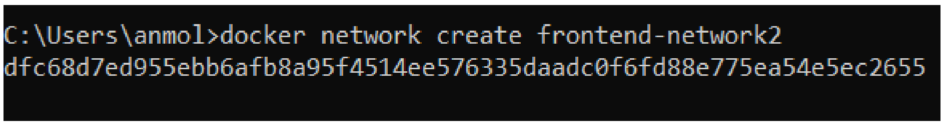
Links are still supported when using *docker network* and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where the --link is used.

The other approach is to provide an alias when connecting a container to a network.

Connect Container with Alias

The following command will connect our Redis instance to the frontend-network with the alias of *db*.

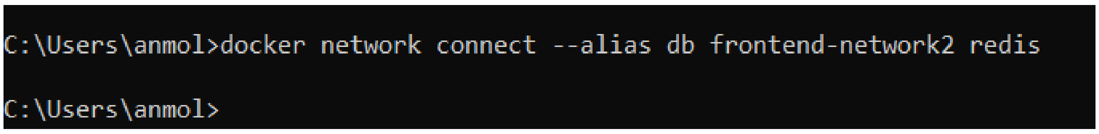
docker network create frontend-network2



```
C:\Users\anmol>docker network create frontend-network2
dfc68d7ed955ebb6afb8a95f4514ee576335daadc0f6fd88e775ea54e5ec2655
```

docker network connect --alias db frontend-network2

redis

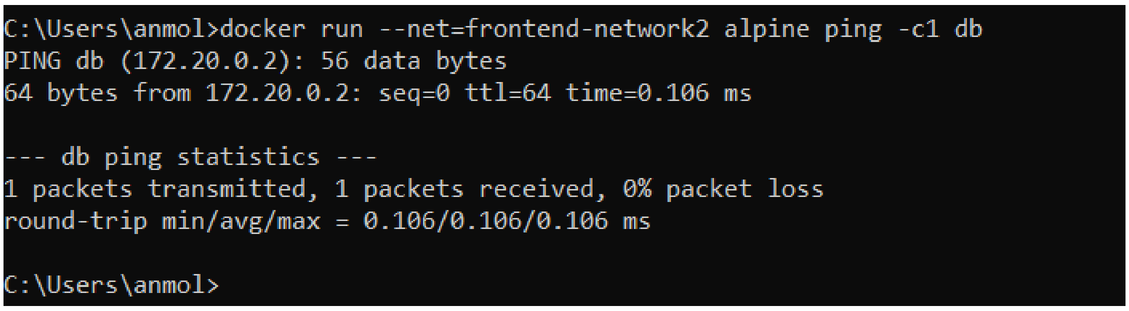


```
C:\Users\anmol>docker network connect --alias db frontend-network2 redis

C:\Users\anmol>
```

When containers attempt to access a service via the name

db, they will be given the IP address of our Redis container.

docker run --net=frontend-network2 alpine ping -c1 db



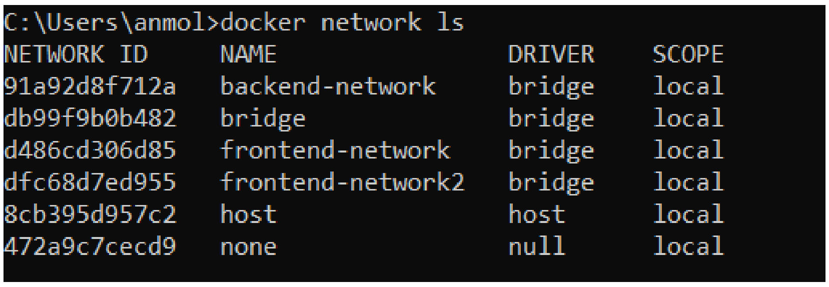
```
C:\Users\anmol>docker run --net=frontend-network2 alpine ping -c1 db
PING db (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.106 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.106/0.106/0.106 ms

C:\Users\anmol>
```

Step 5 - Disconnect Containers

With our networks created, we can use the CLI to explore the details.

The following command will list all the networks on our host.

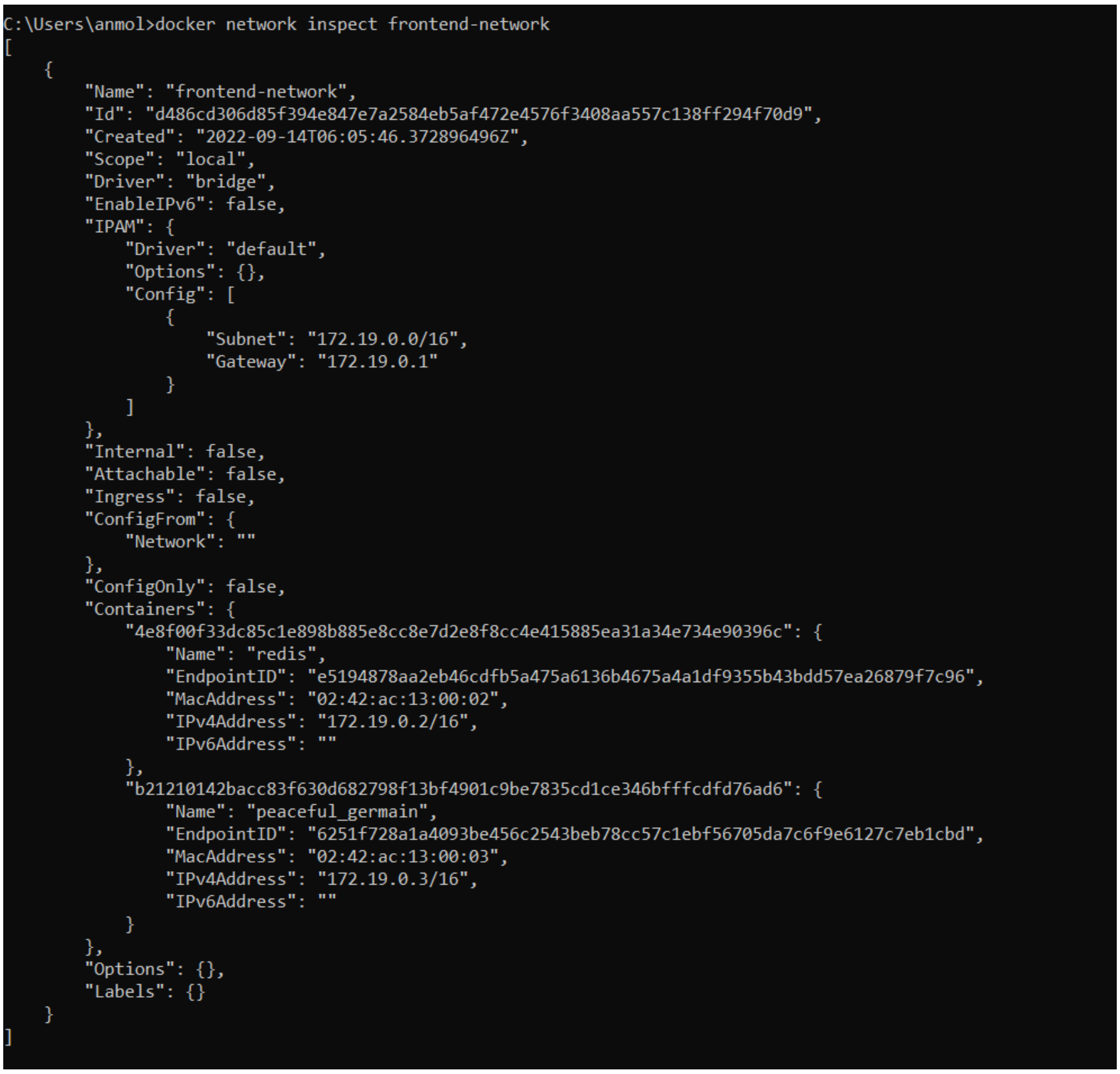
`docker network ls`



```
C:\Users\anmol>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
91a92d8f712a   backend-network     bridge    local
db99f9b0b482   bridge              bridge    local
d486cd306d85   frontend-network    bridge    local
dfc68d7ed955   frontend-network2   bridge    local
8cb395d957c2   host                host      local
472a9c7cecd9   none                null      local
```

We can then explore the network to see which containers are attached and their IP addresses.

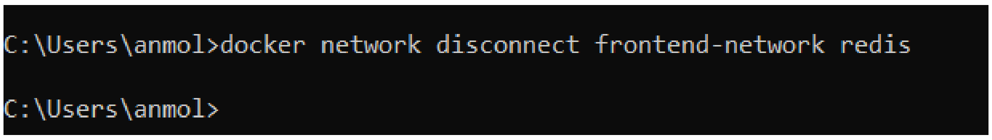
`docker network inspect frontend-network`



```
C:\Users\anmol>docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "d486cd306d85f394e847e7a2584eb5af472e4576f3408aa557c138ff294f70d9",
        "Created": "2022-09-14T06:05:46.372896496Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "4e8f00f33dc85c1e898b885e8cc8e7d2e8f8cc4e415885ea31a34e734e90396c": {
                "Name": "redis",
                "EndpointID": "e5194878aa2eb46cdfb5a475a6136b4675a4a1df9355b43bdd57ea26879f7c96",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            },
            "b21210142bacc83f630d682798f13bf4901c9be7835cd1ce346bfffcdfd76ad6": {
                "Name": "peaceful_germain",
                "EndpointID": "6251f728a1a4093be456c2543beb78cc57c1ebf56705da7c6f9e6127c7eb1cbd",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

The following command disconnects the redis container from the *frontend-network*.

docker network disconnect frontend-network redis



```
C:\Users\anmol>docker network disconnect frontend-network redis

C:\Users\anmol>
```