EXPERIMENT 4

AIM: Working with Docker Network

Steps to Complete:

Step 1 - Create Network

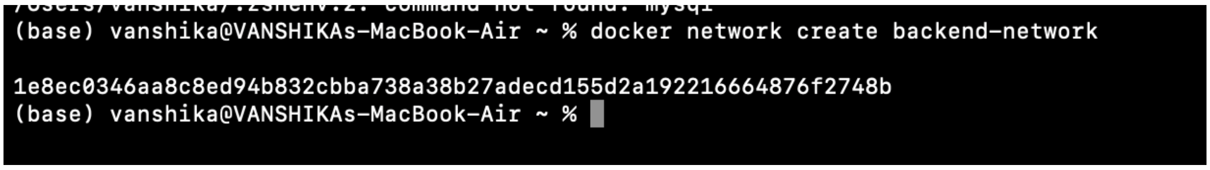
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

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
```

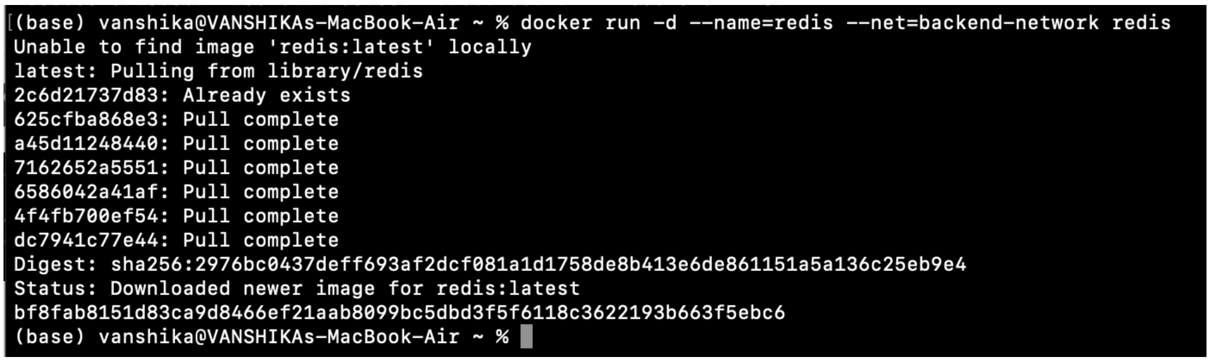


```
/Users/vanshika/.zshenv:2: command not found: mysql
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network create backend-network

1e8ec0346aa8c8ed94b832cbba738a38b27adecd155d2a192216664876f2748b
(base) vanshika@VANSHIKAs-MacBook-Air ~ %
```

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
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker run -d --name=redis --net=backend-network redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
2c6d21737d83: Already exists
625cfba868e3: Pull complete
a45d11248440: Pull complete
7162652a5551: Pull complete
6586042a41af: Pull complete
4f4fb700ef54: Pull complete
dc7941c77e44: Pull complete
Digest: sha256:2976bc0437deff693af2dcf081a1d1758de8b413e6de861151a5a136c25eb9e4
Status: Downloaded newer image for redis:latest
bf8fab8151d83ca9d8466ef21aab8099bc5dbd3f5f6118c3622193b663f5ebc6
(base) vanshika@VANSHIKAs-MacBook-Air ~ %
```

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

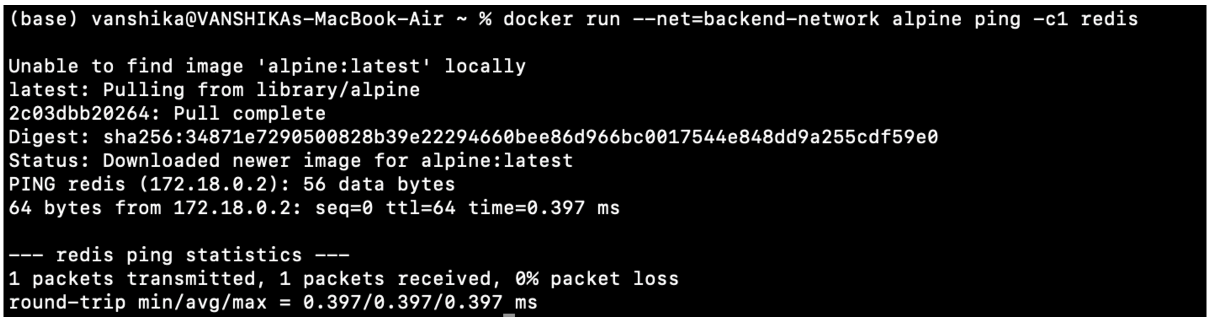
Step 2 - Network Communication

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
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker run --net=backend-network alpine ping -c1 redis

Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
2c03dbb20264: Pull complete
Digest: sha256:34871e7290500828b39e22294660bee86d966bc0017544e848dd9a255cdf59e0
Status: Downloaded newer image for alpine:latest
PING redis (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.397 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.397/0.397/0.397 ms
```

Step 3 - Connect Two Containers

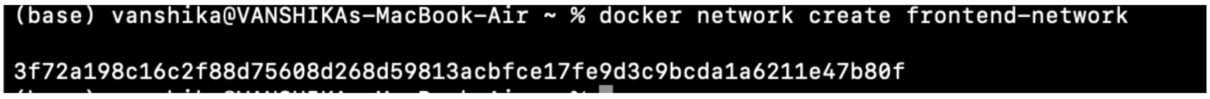
Docker supports multiple networks and containers being attached to more than one network at a time.

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.

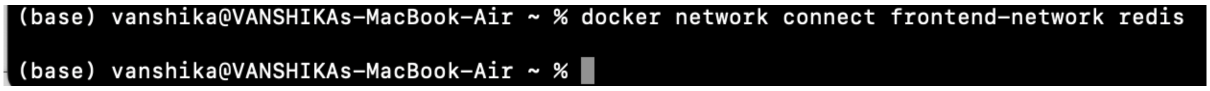
```
docker network create frontend-network
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network create frontend-network

3f72a198c16c2f88d75608d268d59813acbfce17fe9d3c9bcda1a6211e47b80f
```

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

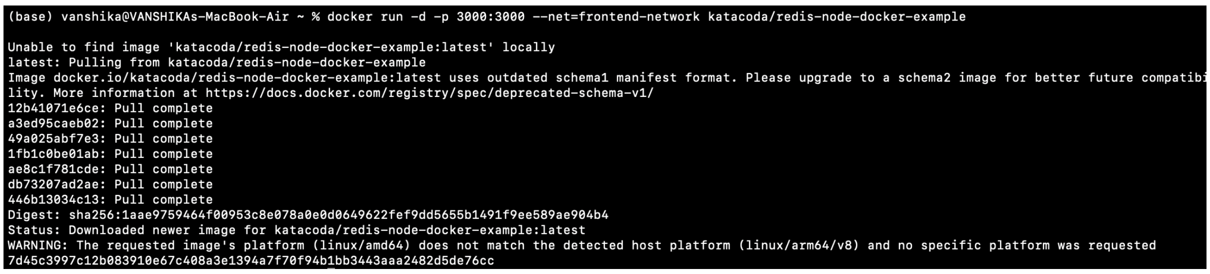
```
docker network connect frontend-network redis
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network connect frontend-network redis

(base) vanshika@VANSHIKAs-MacBook-Air ~ %
```

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
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % 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 upgrade to a schema2 image for better future compatibi
lity. More information at https://docs.docker.com/registry/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
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no specific platform was requested
7d45c3997c12b083910e67c408a3e1394a7f70f94b1bb3443aaa2482d5de76cc
```

You can test it using `curl docker:3000`

Step 4 - Create Aliases

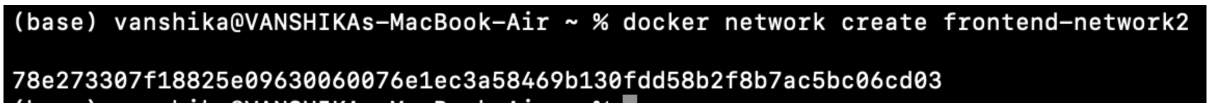
Links are still supported when using *docker network* and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where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

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

Connect Container with Alias

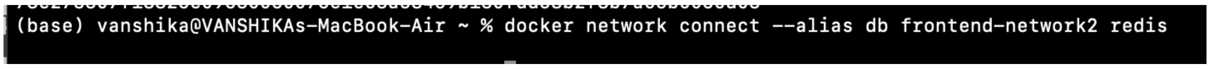
The following command will connect our Redis instance to the frontend-network with the alias of *db*.

```
docker network create frontend-network2
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network create frontend-network2

78e273307f18825e09630060076e1ec3a58469b130fdd58b2f8b7ac5bc06cd03
```

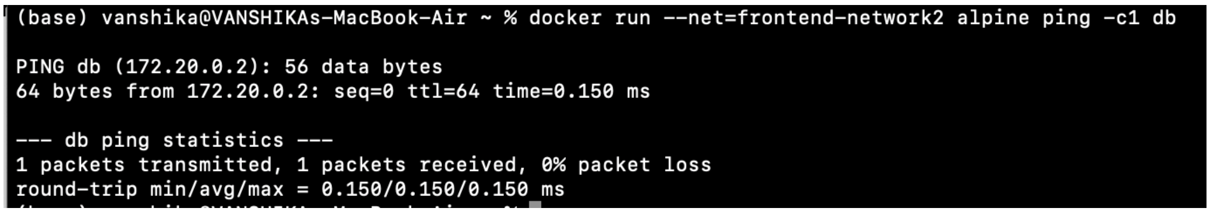
```
docker network connect --alias db frontend-network2 redis
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network connect --alias db frontend-network2 redis
```

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
```



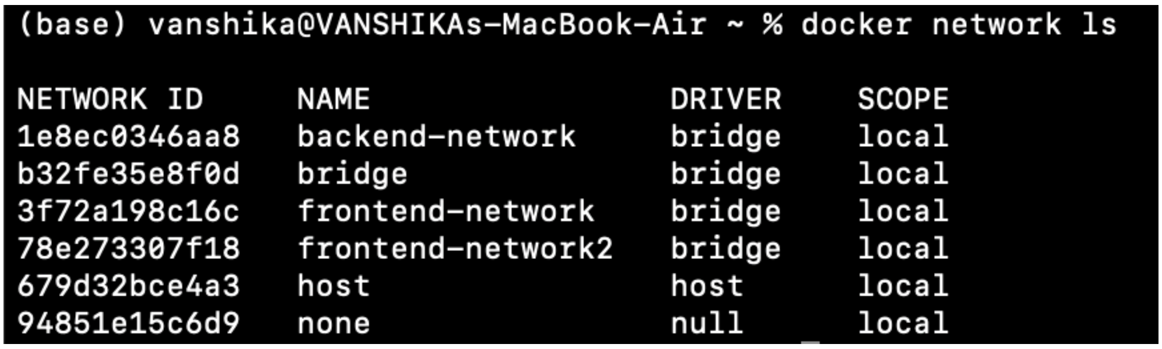
```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker run --net=frontend-network2 alpine ping -c1 db

PING db (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.150 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.150/0.150/0.150 ms
```

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.

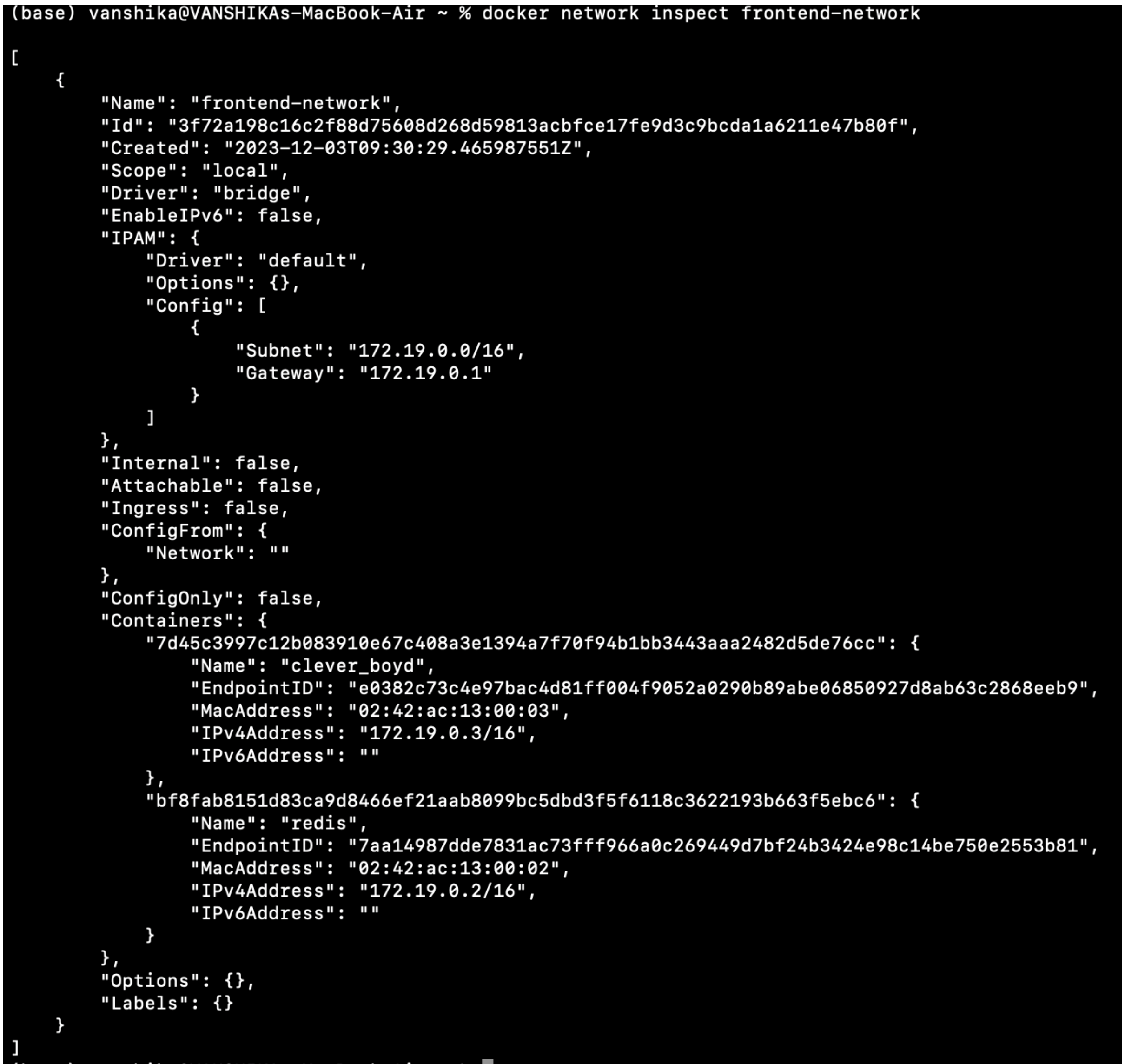
```
docker network ls
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network ls

NETWORK ID     NAME                DRIVER    SCOPE
1e8ec0346aa8   backend-network     bridge    local
b32fe35e8f0d   bridge              bridge    local
3f72a198c16c   frontend-network    bridge    local
78e273307f18   frontend-network2   bridge    local
679d32bce4a3   host                host      local
94851e15c6d9   none                null      local
```

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

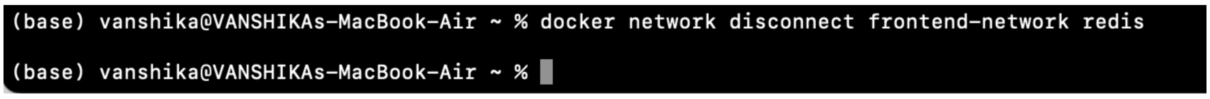
```
docker network inspect frontend-network
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network inspect frontend-network

[
    {
        "Name": "frontend-network",
        "Id": "3f72a198c16c2f88d75608d268d59813acbfce17fe9d3c9bcda1a6211e47b80f",
        "Created": "2023-12-03T09:30:29.465987551Z",
        "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": {
            "7d45c3997c12b083910e67c408a3e1394a7f70f94b1bb3443aaa2482d5de76cc": {
                "Name": "clever_boyd",
                "EndpointID": "e0382c73c4e97bac4d81ff004f9052a0290b89abe06850927d8ab63c2868eeb9",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "bf8fab8151d83ca9d8466ef21aab8099bc5dbd3f5f6118c3622193b663f5ebc6": {
                "Name": "redis",
                "EndpointID": "7aa14987dde7831ac73fff966a0c269449d7bf24b3424e98c14be750e2553b81",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

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

```
docker network disconnect frontend-network redis
```



```
(base) vanshika@VANSHIKAs-MacBook-Air ~ % docker network disconnect frontend-network redis

(base) vanshika@VANSHIKAs-MacBook-Air ~ %
```