UPES

UNIVERSITY OF TOMORROW

Application Containerization And Orchestration Lab

Submitted By – Chitwan Singh

SAP ID – 500097009

Enrolment no. – R2142211291

Batch – DevOps B4

Submitted to

– Dr. Hitesh Kumar Sharma

EXPERIMENT 4

AIM: Working with Docker Network

Steps to Complete:

Step 1 - Create Network

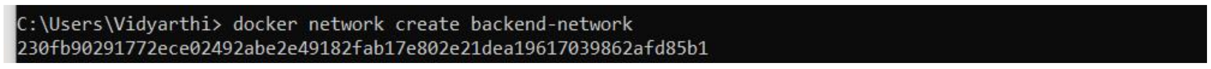
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

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
```

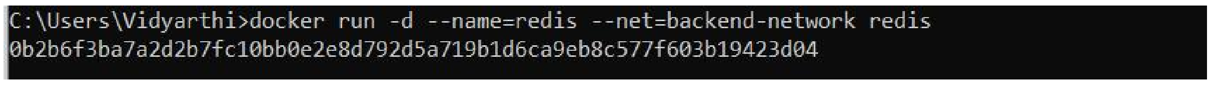


```
C:\Users\Vidyarthi> docker network create backend-network
230fb90291772ece02492abe2e49182fab17e802e21dea19617039862afd85b1
```

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\Vidyarthi>docker run -d --name=redis --net=backend-network redis
0b2b6f3ba7a2d2b7fc10bb0e2e8d792d5a719b1d6ca9eb8c577f603b19423d04
```

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

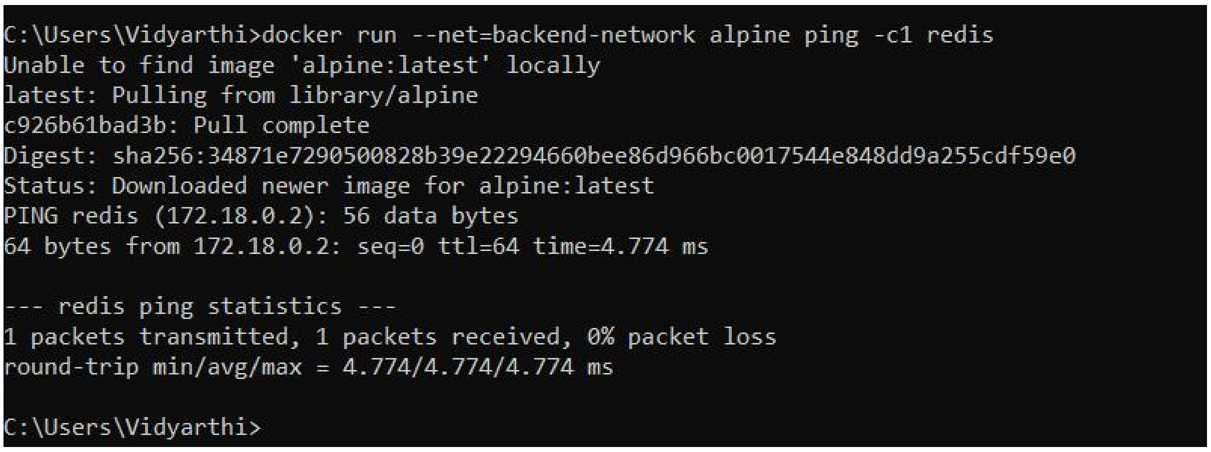
Step 2 - Network Communication

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\Vidyarthi>docker run --net=backend-network alpine ping -c1 redis
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
c926b61bad3b: 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=4.774 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 4.774/4.774/4.774 ms

C:\Users\Vidyarthi>
```

Step 3 - Connect Two Containers

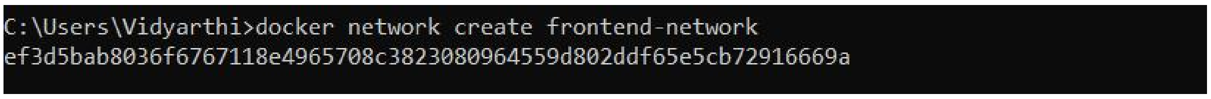
Docker supports multiple networks and containers being attached to more than one network at a time.

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.

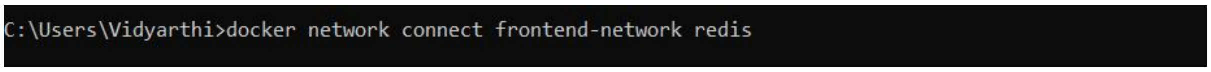
```
docker network create frontend-network
```



```
C:\Users\Vidyarthi>docker network create frontend-network
ef3d5bab8036f6767118e4965708c3823080964559d802ddf65e5cb72916669a
```

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

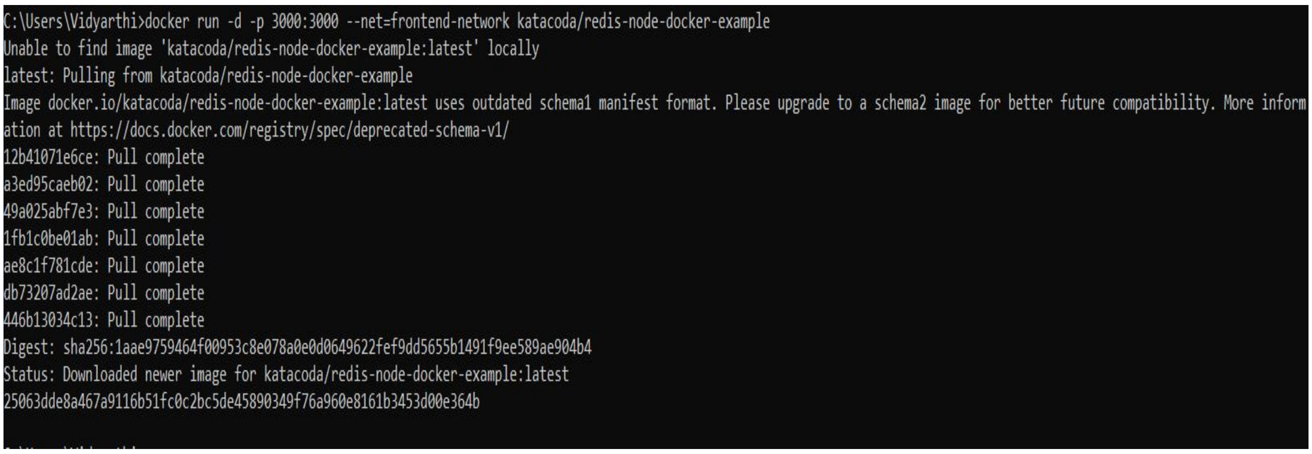
```
docker network connect frontend-network redis
```



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

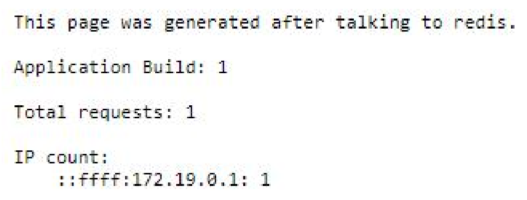
When we launch the web server, given it's attached to the same network it will be able to communicate with our Redis instance.

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



```
C:\Users\Vidyarthi>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 compatibility. More inform
ation 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
25063dde8a467a9116b51fc0c2bc5de45890349f76a960e8161b3453d00e364b
```

You can test it using `curl docker:3000`



```
This page was generated after talking to redis.

Application Build: 1

Total requests: 1

IP count:
    ::ffff:172.19.0.1: 1
```

Step 4 - Create Aliases

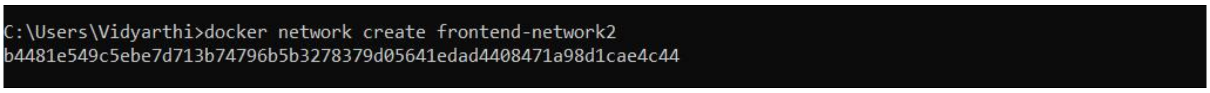
Links are still supported when using *docker network* and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where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

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

Connect Container with Alias

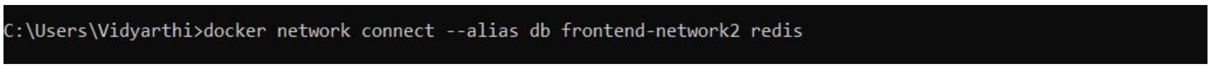
The following command will connect our Redis instance to the frontend-network with the alias of *db*.

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



```
C:\Users\Vidyarthi>docker network create frontend-network2
b4481e549c5ebe7d713b74796b5b3278379d05641edad4408471a98d1cae4c44
```

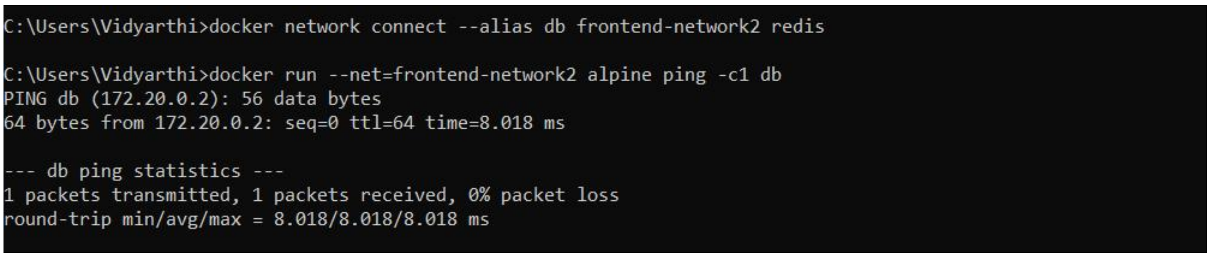
```
docker network connect --alias db frontend-network2 redis
```



```
C:\Users\Vidyarthi>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
```



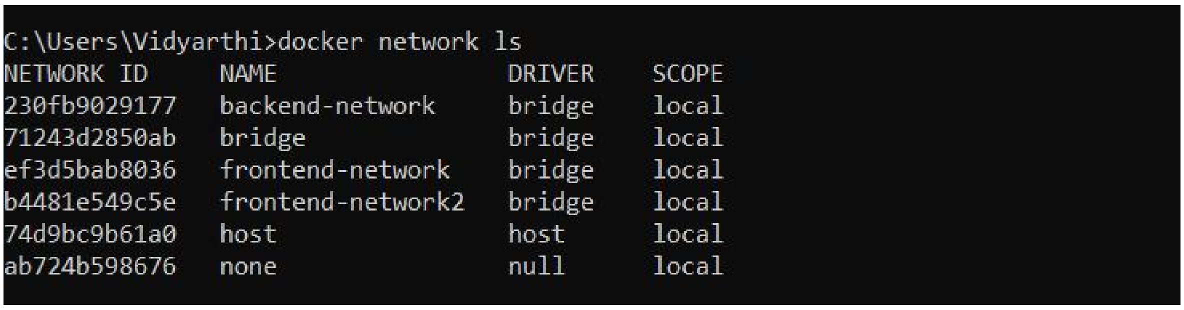
```
C:\Users\Vidyarthi>docker network connect --alias db frontend-network2 redis

C:\Users\Vidyarthi>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=8.018 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 8.018/8.018/8.018 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.

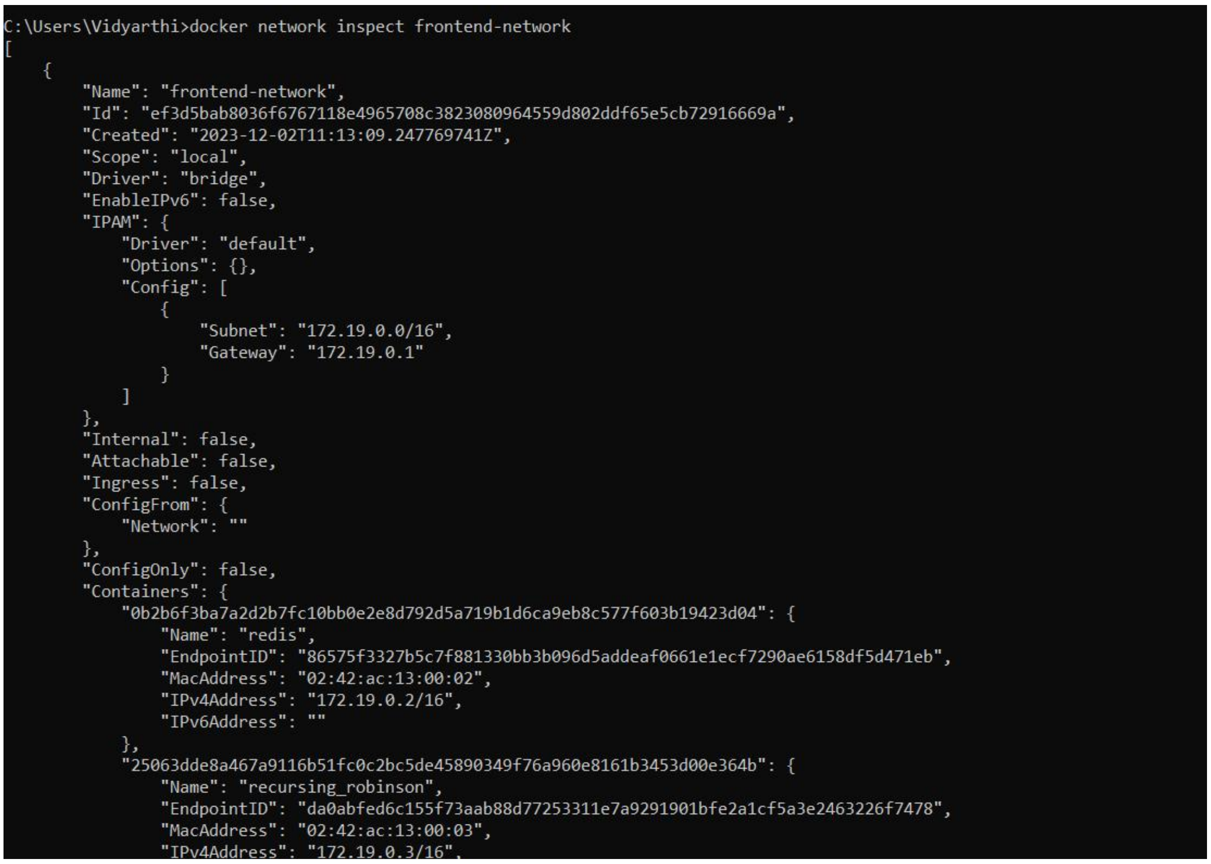
```
docker network ls
```



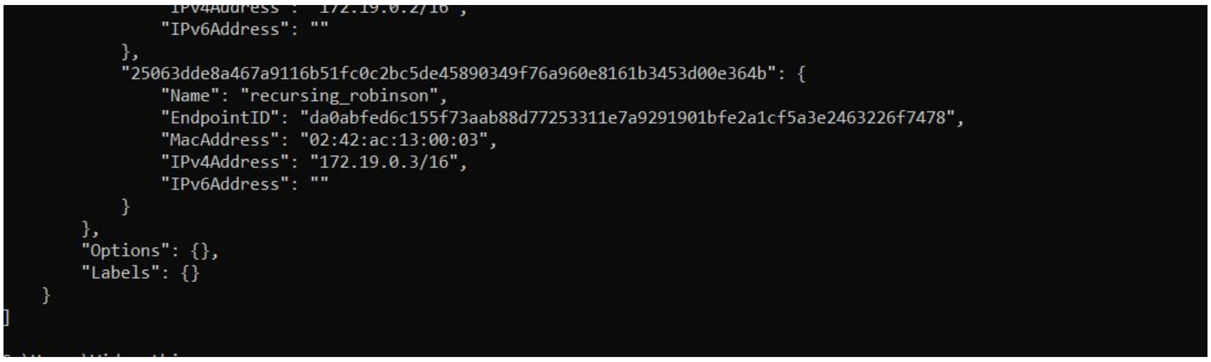
```
C:\Users\Vidyarthi>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
230fb9029177   backend-network     bridge    local
71243d2850ab   bridge              bridge    local
ef3d5bab8036   frontend-network    bridge    local
b4481e549c5e   frontend-network2   bridge    local
74d9bc9b61a0   host                host      local
ab724b598676   none                null      local
```

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

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



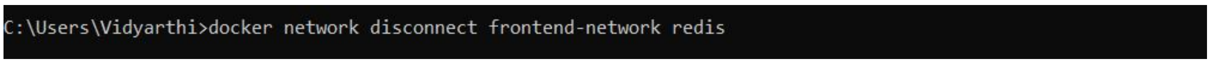
```
C:\Users\Vidyarthi>docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "ef3d5bab8036f6767118e4965708c3823080964559d802ddf65e5cb72916669a",
        "Created": "2023-12-02T11:13:09.247769741Z",
        "Scope": "local",
        "Driver": "bridge",
        "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": {
            "0b2b6f3ba7a2d2b7fc10bb0e2e8d792d5a719b1d6ca9eb8c577f603b19423d04": {
                "Name": "redis",
                "EndpointID": "86575f3327b5c7f881330bb3b096d5addeaf0661e1ecf7290ae6158df5d471eb",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            },
            "25063dde8a467a9116b51fc0c2bc5de45890349f76a960e8161b3453d00e364b": {
                "Name": "recursing_robinson",
                "EndpointID": "da0abfed6c155f73aab88d77253311e7a9291901bfe2a1cf5a3e2463226f7478",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
```



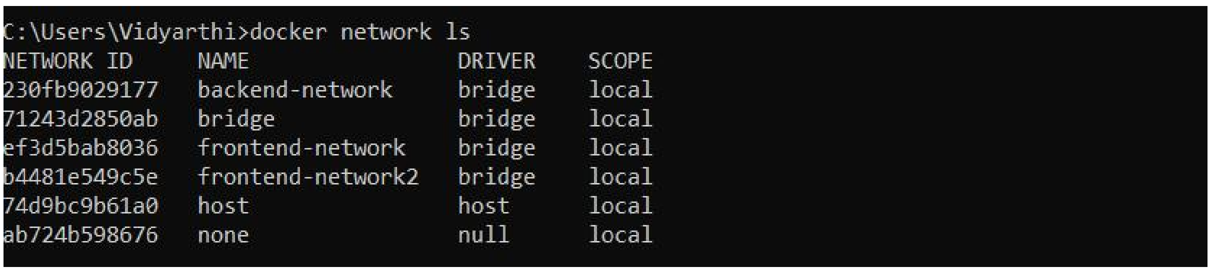
```
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            },
            "25063dde8a467a9116b51fc0c2bc5de45890349f76a960e8161b3453d00e364b": {
                "Name": "recursing_robinson",
                "EndpointID": "da0abfed6c155f73aab88d77253311e7a9291901bfe2a1cf5a3e2463226f7478",
                "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\Vidyarthi>docker network disconnect frontend-network redis
```



```
C:\Users\Vidyarthi>docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
230fb9029177   backend-network     bridge    local
71243d2850ab   bridge              bridge    local
ef3d5bab8036   frontend-network    bridge    local
b4481e549c5e   frontend-network2   bridge    local
74d9bc9b61a0   host                host      local
ab724b598676   none                null      local
```