UPES

UNIVERSITY OF THE FUTURE

Appl Containerization and Orchestration

Experiment 4

AIM: Working with Docker Network

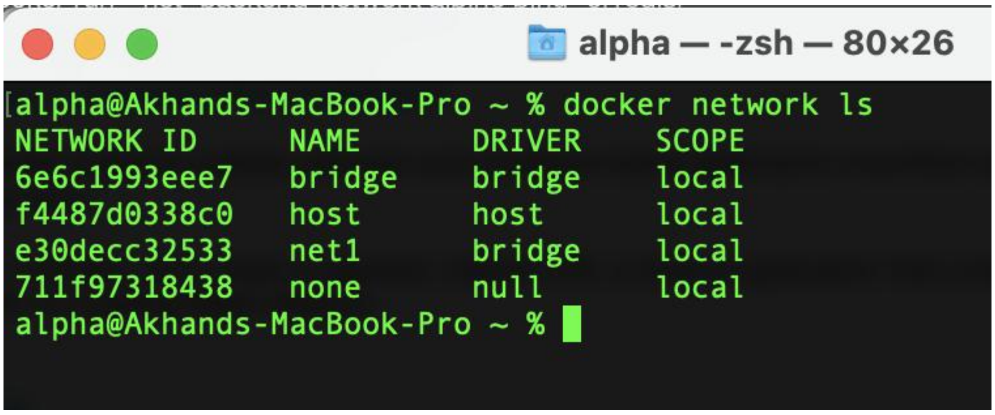
Submitted by-
Akhand Pratap Singh

Submitted to:
Mr. Hitesh Kumar Sharma

Steps to Complete:

Check list of available network

docker network ls



```
alpha@Akhands-MacBook-Pro ~ % docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
6e6c1993eee7   bridge    bridge    local
f4487d0338c0   host      host      local
e30decc32533   net1      bridge    local
711f97318438   none      null      local
alpha@Akhands-MacBook-Pro ~ %
```

Step 1 - Create Network

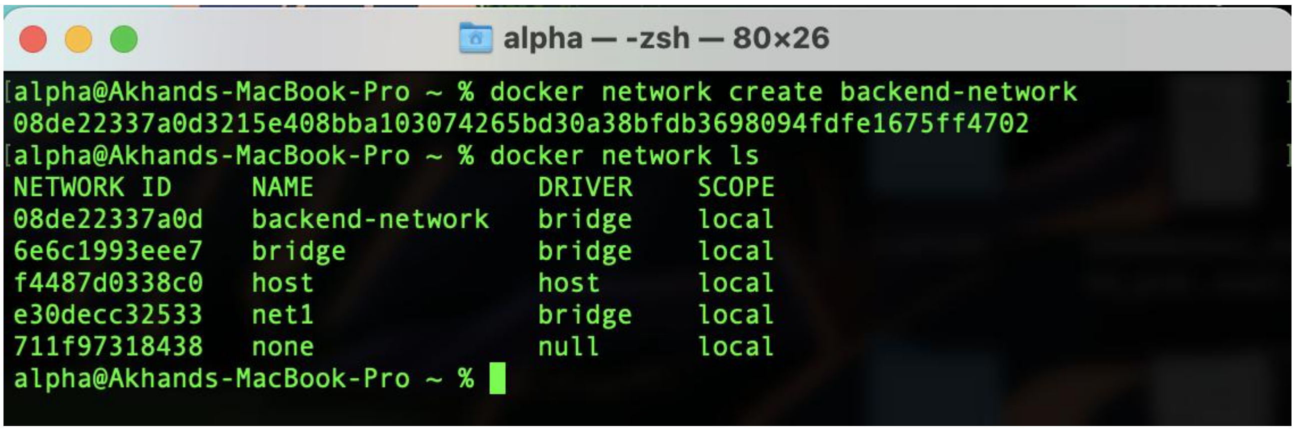
The first step is to create a network using the CLI. This network will allow us to attach multiple containers which will be able to discover each other.

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
```

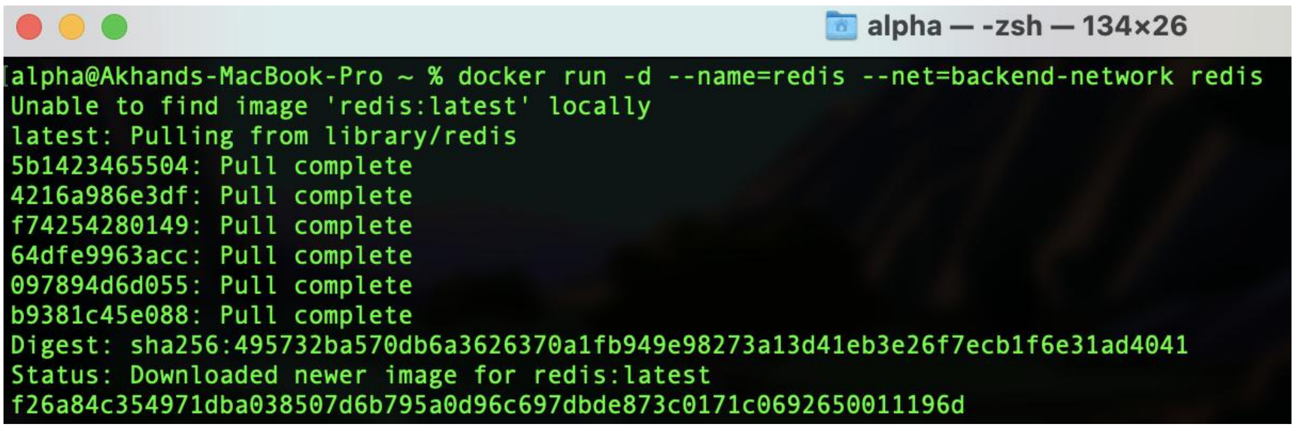


```
alpha@Akhands-MacBook-Pro ~ % docker network create backend-network
08de22337a0d3215e408bba103074265bd30a38bfdb3698094fdfe1675ff4702
alpha@Akhands-MacBook-Pro ~ % docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
08de22337a0d   backend-network   bridge    local
6e6c1993eee7   bridge            bridge    local
f4487d0338c0   host              host      local
e30decc32533   net1              bridge    local
711f97318438   none              null      local
alpha@Akhands-MacBook-Pro ~ %
```

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
```



```
alpha@Akhands-MacBook-Pro ~ % docker run -d --name=redis --net=backend-network redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
5b1423465504: Pull complete
4216a986e3df: Pull complete
f74254280149: Pull complete
64dfe9963acc: Pull complete
097894d6d055: Pull complete
b9381c45e088: Pull complete
Digest: sha256:495732ba570db6a3626370a1fb949e98273a13d41eb3e26f7ecb1f6e31ad4041
Status: Downloaded newer image for redis:latest
f26a84c354971dba038507d6b795a0d96c697dbde873c0171c0692650011196d
```

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

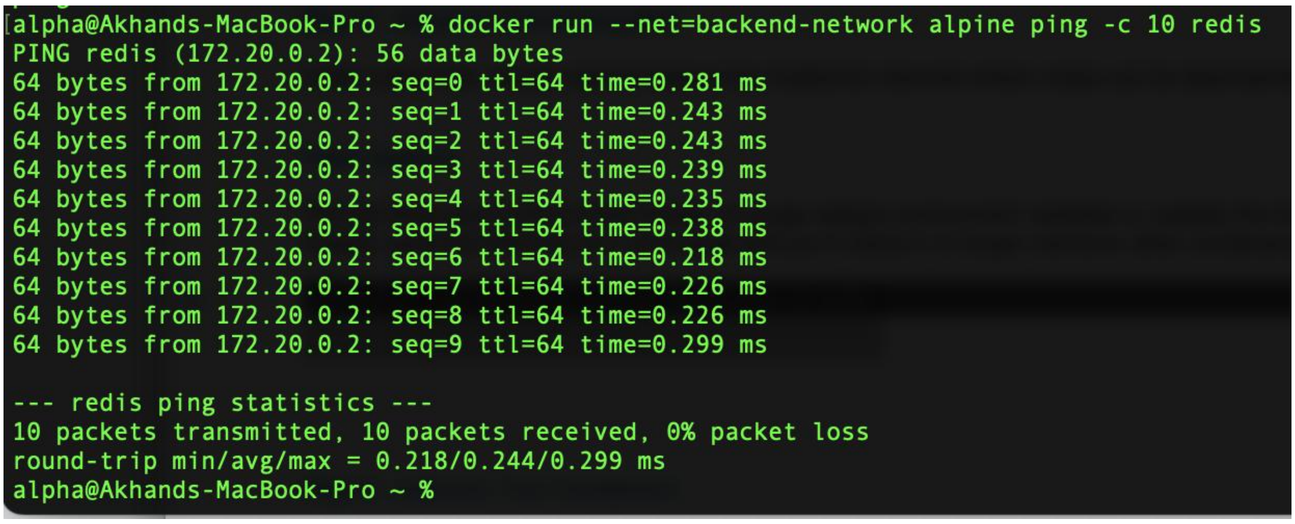
Step 2 - Network Communication

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 -cl redis|
```



```
alpha@Akhands-MacBook-Pro ~ % docker run --net=backend-network alpine ping -c 10 redis
PING redis (172.20.0.2): 56 data bytes
64 bytes from 172.20.0.2: seq=0 ttl=64 time=0.281 ms
64 bytes from 172.20.0.2: seq=1 ttl=64 time=0.243 ms
64 bytes from 172.20.0.2: seq=2 ttl=64 time=0.243 ms
64 bytes from 172.20.0.2: seq=3 ttl=64 time=0.239 ms
64 bytes from 172.20.0.2: seq=4 ttl=64 time=0.235 ms
64 bytes from 172.20.0.2: seq=5 ttl=64 time=0.238 ms
64 bytes from 172.20.0.2: seq=6 ttl=64 time=0.218 ms
64 bytes from 172.20.0.2: seq=7 ttl=64 time=0.226 ms
64 bytes from 172.20.0.2: seq=8 ttl=64 time=0.226 ms
64 bytes from 172.20.0.2: seq=9 ttl=64 time=0.299 ms

--- redis ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.218/0.244/0.299 ms
alpha@Akhands-MacBook-Pro ~ %
```

Step 3 - Connect Two Containers

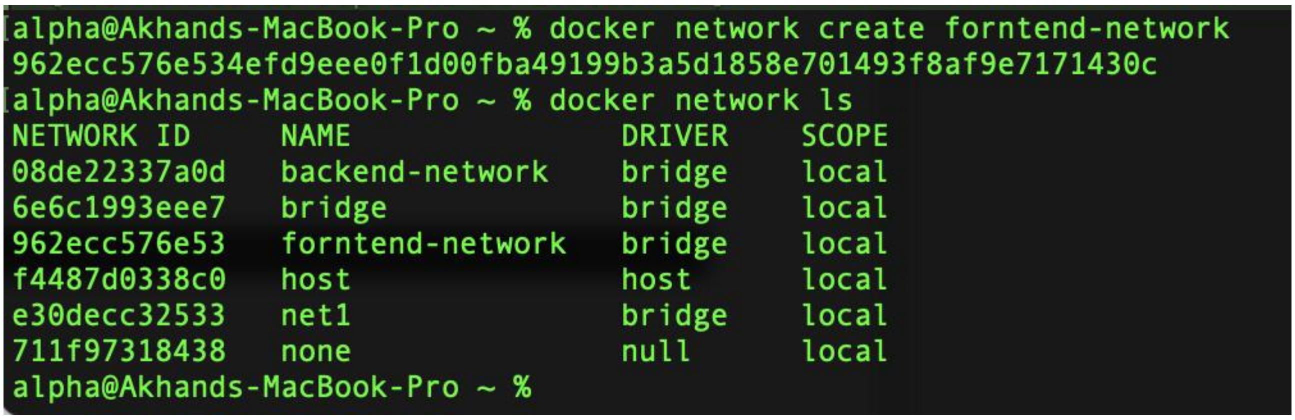
Docker supports multiple networks and containers being attached to more than one network at a time.

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.

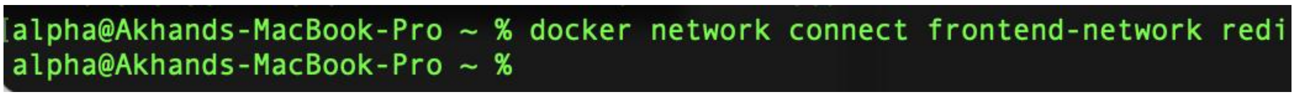
```
docker network create frontend-network
```



```
[alpha@Akhands-MacBook-Pro ~ % docker network create forntend-network
962ecc576e534efd9eee0f1d00fba49199b3a5d1858e701493f8af9e7171430c
[alpha@Akhands-MacBook-Pro ~ % docker network ls
NETWORK ID     NAME               DRIVER    SCOPE
08de22337a0d   backend-network    bridge    local
6e6c1993eee7   bridge             bridge    local
962ecc576e53   forntend-network   bridge    local
f4487d0338c0   host               host      local
e30decc32533   net1               bridge    local
711f97318438   none               null      local
alpha@Akhands-MacBook-Pro ~ %
```

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

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

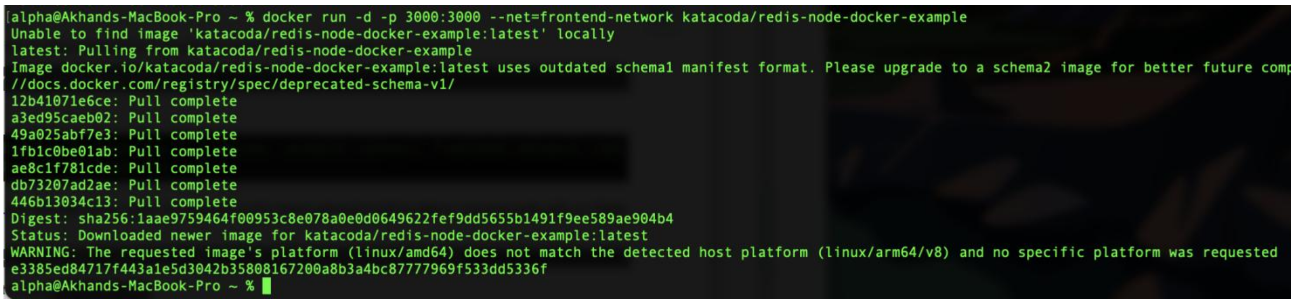


```
[alpha@Akhands-MacBook-Pro ~ % docker network connect frontend-network redi
alpha@Akhands-MacBook-Pro ~ %
```

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|
```

You can test it using `curl docker:3000`



```
[alpha@Akhands-MacBook-Pro ~ % 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 comp
//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
e3385ed84717f443a1e5d3042b35808167200a8b3a4bc87777969f533dd5336f
alpha@Akhands-MacBook-Pro ~ %
```

Step 4 - Create Aliases

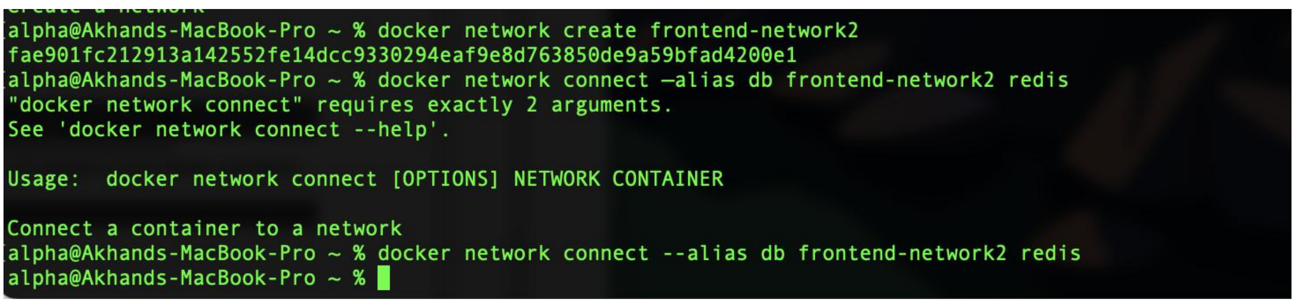
Links are still supported when using docker network and provide a way to define an Alias to the container name. This will give the container an extra DNS entry name and way to be discovered. When using --link the embedded DNS will guarantee that localised lookup result only on that container where the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

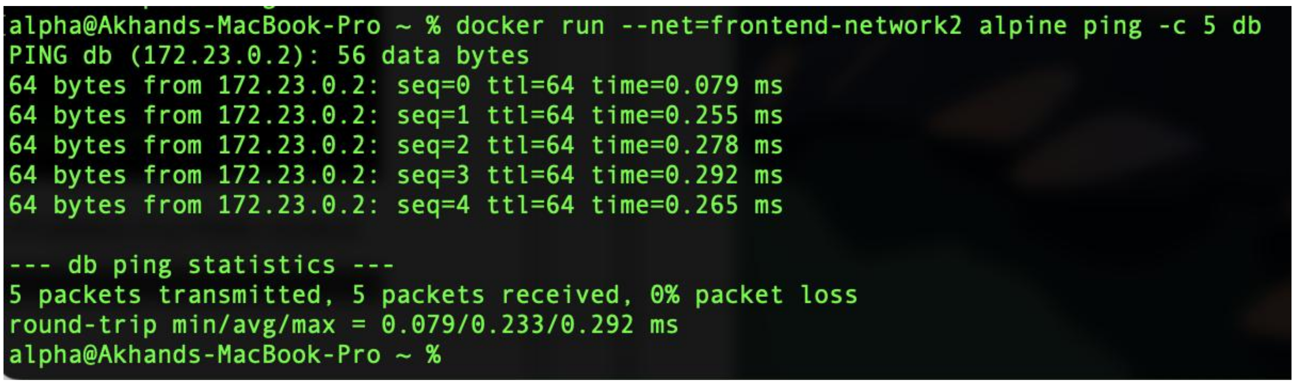
```
docker network create frontend-network2

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



```
alpha@Akhands-MacBook-Pro ~ % docker network create frontend-network2
fae901fc212913a142552fe14dcc9330294eaf9e8d763850de9a59bfad4200e1
alpha@Akhands-MacBook-Pro ~ % docker network connect —alias db frontend-network2 redis
"docker network connect" requires exactly 2 arguments.
See 'docker network connect --help'.

Usage:  docker network connect [OPTIONS] NETWORK CONTAINER

Connect a container to a network
alpha@Akhands-MacBook-Pro ~ % docker network connect --alias db frontend-network2 redis
alpha@Akhands-MacBook-Pro ~ %
```

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 -cl db|
```



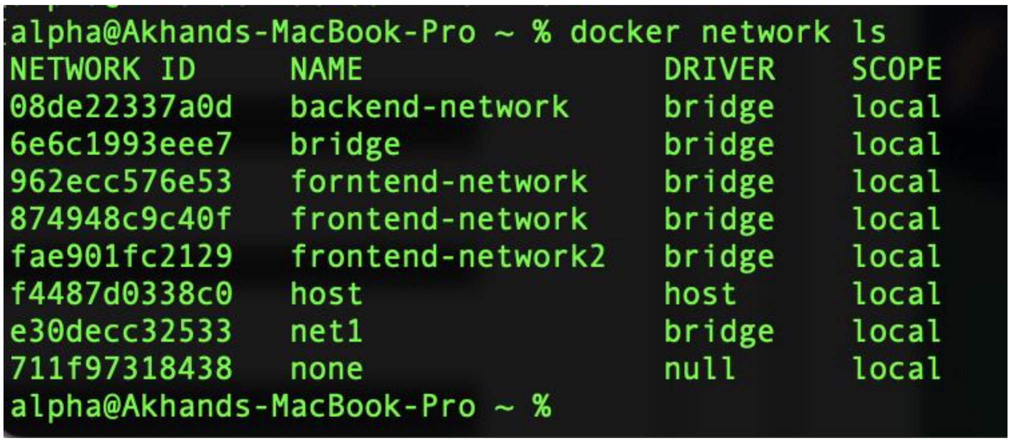
```
alpha@Akhands-MacBook-Pro ~ % docker run --net=frontend-network2 alpine ping -c 5 db
PING db (172.23.0.2): 56 data bytes
64 bytes from 172.23.0.2: seq=0 ttl=64 time=0.079 ms
64 bytes from 172.23.0.2: seq=1 ttl=64 time=0.255 ms
64 bytes from 172.23.0.2: seq=2 ttl=64 time=0.278 ms
64 bytes from 172.23.0.2: seq=3 ttl=64 time=0.292 ms
64 bytes from 172.23.0.2: seq=4 ttl=64 time=0.265 ms

--- db ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.079/0.233/0.292 ms
alpha@Akhands-MacBook-Pro ~ %
```

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.

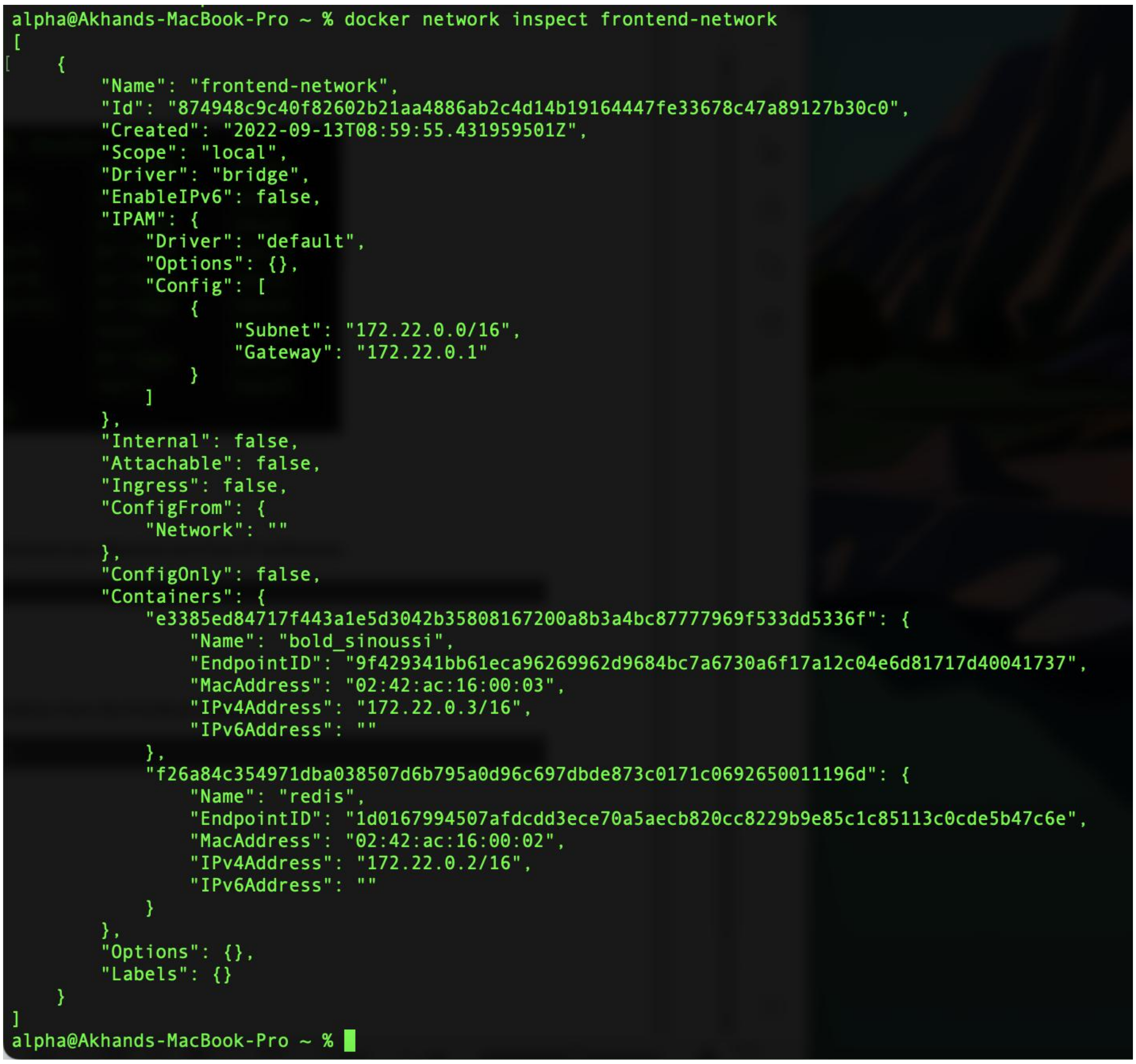
```
docker network ls
```



```
alpha@Akhands-MacBook-Pro ~ % docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
08de22337a0d   backend-network     bridge    local
6e6c1993eee7   bridge              bridge    local
962ecc576e53   forntend-network    bridge    local
874948c9c40f   frontend-network    bridge    local
fae901fc2129   frontend-network2   bridge    local
f4487d0338c0   host                host      local
e30decc32533   net1                bridge    local
711f97318438   none                null      local
alpha@Akhands-MacBook-Pro ~ %
```

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

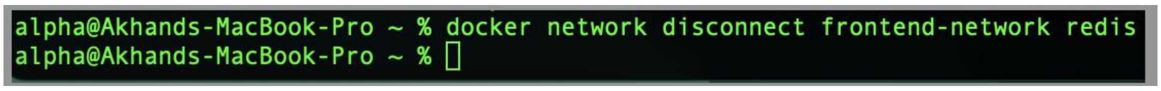
```
|docker network inspect frontend-network|
```



```
alpha@Akhands-MacBook-Pro ~ % docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "874948c9c40f82602b21aa4886ab2c4d14b19164447fe33678c47a89127b30c0",
        "Created": "2022-09-13T08:59:55.431959501Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.22.0.0/16",
                    "Gateway": "172.22.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "e3385ed84717f443a1e5d3042b35808167200a8b3a4bc87777969f533dd5336f": {
                "Name": "bold_sinoussi",
                "EndpointID": "9f429341bb61eca96269962d9684bc7a6730a6f17a12c04e6d81717d40041737",
                "MacAddress": "02:42:ac:16:00:03",
                "IPv4Address": "172.22.0.3/16",
                "IPv6Address": ""
            },
            "f26a84c354971dba038507d6b795a0d96c697dbde873c0171c0692650011196d": {
                "Name": "redis",
                "EndpointID": "1d0167994507afdcdd3ece70a5aecb820cc8229b9e85c1c85113c0cde5b47c6e",
                "MacAddress": "02:42:ac:16:00:02",
                "IPv4Address": "172.22.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
alpha@Akhands-MacBook-Pro ~ %
```

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

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



```
alpha@Akhands-MacBook-Pro ~ % docker network disconnect frontend-network redis
alpha@Akhands-MacBook-Pro ~ %
```