APPLICATION CONTAINERIZATION

LAB EXPERIMENT 3

Create docker network allowing containers to communicate

In this experiment, we are going to create a network in Docker. This network will allow us to attach multiple containers with one another and each container will be able to discover each other and can communicate.

The steps that needs to be followed are:

1. Create a network

Command syntax: docker network create <network-name>
Here we will create a network with the name backend-network

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network create backend-network
97a2997ea48814fe108e4cc988a7824a9f3342c6fe69176d1b7eaf1822bd25f5
atishay@atishay-HP-15-Notebook-PC:~$
```

2. Now we will create a new redis container, named redis only, and we will assign backend -network to this container.

Command syntax: docker run --net=<network-name> <container-image>

```
atishay@atishay-HP-15-Notebook-PC:~$ docker run -d --name=redis --net=backend-ne
twork redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
45b42c59be33: Pull complete
5ce2e937bf62: Pull complete
2a031498ff58: Pull complete
78f77a50d3fe: Pull complete
60553ab7cb08: Pull complete
68240431d2fc: Pull complete
Digest: sha256:f29bcfb891678a0c6a0fc5da0b32ce1ac685af87c0f3aa9327e562da8d3f3b88
Status: Downloaded newer image for redis:latest
ae88c88fd92247c16ee769f3601e72b29410ef49244d7d224449c14cc70a339c
atishay@atishay-HP-15-Notebook-PC:~$
```

We can see our newly created container is running using the command docker ps.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS         PORTS      NAMES
ae88c88fd922   redis     "docker-entrypoint.s…"   15 hours ago   Up 3 seconds   6379/tcp   redis
atishay@atishay-HP-15-Notebook-PC:~$
```

3. The networks in Docker do not use environment variables to discover other containers. We can verify this by creating a new container and assigning it the same network.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker run --net=backend-network alpine env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=2659231dad59
HOME=/root
atishay@atishay-HP-15-Notebook-PC:~$
```

The above output verifies that there is no entry of the redis container which is also connected to the same network in the environment variables of this alpine container.

Also, Docker does not add any entries in /etc/host for the other containers. This can be verified as follows.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker run --net=backend-network alpine cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.3      a02f9f2480e4
atishay@atishay-HP-15-Notebook-PC:~$
```

The above output has no entry for any other container assigned to the same network as this container.

4. Neither environment variables, nor the hosts file is updated. This is because the containers communicate via DNS-based entries.

The DNS server is assigned to all containers and it is set in resolv.conf file. This can be verified as follows:

```
atishay@atishay-HP-15-Notebook-PC:~$ docker run --net=backend-network alpine cat /etc/resolv.conf
search domain.name
nameserver 127.0.0.11
options edns0 trust-ad ndots:0
atishay@atishay-HP-15-Notebook-PC:~$
```

5. Now let us see how the containers connected to the same network can Communicate with each other.

When a container wants to access another container by its name, the DNS return the IP address of that container, and thus making the communication work. Let us try to ping the redis container which is attached to the same network.

```
atishay@atishay-HP-15-Notebook-PC:~$ 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=0.210 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.210/0.210/0.210 ms
atishay@atishay-HP-15-Notebook-PC:~$
```

6. We can attach a container with multiple networks. For this, let us create a new network named frontend-network.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network create frontend-network
3d8c67918abc895d12b9d6b7990d7339ef928683a28772354eae5977c69bc2ec
atishay@atishay-HP-15-Notebook-PC:~$
```

7. We can connect the already created containers to a network using the following command.

Command syntax: docker network connect <network-name> <container-name>

Now we will connect the previously made redis container with this frontend-network. This redis container is already connected with the backend-network.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network connect frontend-network red
is
atishay@atishay-HP-15-Notebook-PC:~$
```

8. Now we will use a customized redis image created by katacoda to communicate over frontend-network.

We will bind this customized image to port 3000.

```
atishay@atishay-HP-15-Notebook-PC:~$ 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 schema
1 manifest format. Please upgrade to a schema2 image for better future compati
bility. More information at https://docs.docker.com/registry/spec/deprecated-s
chema-v1/
12b41071e6ce: Pull complete

a3ed95caeb02: Pull complete

49a025abf7e3: Pull complete

1fb1c0be01ab: Pull complete

ae8c1f781cde: Pull complete

db73207ad2ae: Pull complete

446b13034c13: Pull complete
```

We can verify that it is able to communicate as follows:

```
atishay@atishay-HP-15-Notebook-PC:~$ curl localhost:3000
This page was generated after talking to redis.

Application Build: 1

Total requests: 1

IP count:
    ::ffff:172.20.0.1: 1
```

9. We can also create aliases for containers. This will create a new entry for the container in the DNS.

Let us see how this works. Let us first create a new network **frontend-network2** and then we will connect our **redis** container with this network and we will provide an alias **db.**

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network create frontend-network2
065952efbb0204f8de814aa4dfc71b8bb0851d41492fadba811537160da78ceb
atishay@atishay-HP-15-Notebook-PC:~$ docker network connect --alias db fronten
d-network2 redis
```

When other containers try to communicate with db, they will be provided with the IP of redis. We can verify that other containers can communicate to our container by using aliases.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker run --net=frontend-network2 alpine
 ping -c1 db
PING db (172.21.0.2): 56 data bytes
64 bytes from 172.21.0.2: seq=0 ttl=64 time=0.303 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.303/0.303/0.303 ms
atishay@atishay-HP-15-Notebook-PC:~$
```

10. You can verify the creation of the network by listing the available networks.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
97a2997ea488   backend-network     bridge    local
d8bc4b35eb4b   bridge              bridge    local
3d8c67918abc   frontend-network    bridge    local
065952efbb02   frontend-network2   bridge    local
1a06f9918c3c   host                host      local
fea9810c6857   none                null      local
```

11. We can get the details of any network by using the following command.

Command syntax: docker network inspect <network-name>

Let us inspect frontend-network to see the connected containers.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "3d8c67918abc895d12b9d6b7990d7339ef928683a28772354eae5977c69bc2e
c",
        "Created": "2021-02-10T18:46:05.813629906+05:30",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.20.0.0/16",
                    "Gateway": "172.20.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
```

12. We can disconnect a container from a network by disconnect command.

Command syntax: docker network disconnect <network-name> <container-name>

Let us use this to disconnect the redis container from the frontend-network.

```
atishay@atishay-HP-15-Notebook-PC:~$ docker network disconnect frontend-networ
k redis
atishay@atishay-HP-15-Notebook-PC:~$
```