ANURAG PACHAURI

R171218026

500069505

EXPERIMENT 3

Create the network and connect with the container

Create Network:-

Command:- docker network create backend-network

```
$ docker network create backend-network
de46c857f3ecc217dd4f06e840b34358afa48285d6620d9905d72a6ccf05982b
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
de46c857f3ec        backend-network     bridge              local
a648cec72646        bridge              bridge              local
0c288d2ed25a        host                host                local
4d3221fe852b        none                null                local
$
```

Connect To Network:-

Command:- docker run -d --name=redis --net=backend-network redis

```
$ docker run -d --name=redis --net=backend-network redis
da6a8a308f1d73193805f630528fd01dba43160079b5e4ff1f25d9583b2d0c2b
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
          NAMES
da6a8a308f1d        redis               "docker-entrypoint.s…"   25 seconds ago      Up 23 seconds
p           redis
$
```

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.

Command:- [1] $ docker run --net=backend-network alpine env
[2] $ docker run --net=backend-network alpine cat /etc/hosts

```
$ docker run --net=backend-network alpine env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=ab1e378e05e9
HOME=/root
$ 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
172.19.0.3      d24deea9a5f7
```

The way containers can communicate via an Embedded DNS Server in Docker. This DNS server is assigned to all containers via the IP 127.0.0.11 and set in the *resolv.conf* file.

```
$ docker run --net=backend-network alpine cat /etc/resolv.conf
nameserver 127.0.0.11
options ndots:0
```

When containers access other container. The DNS server will return the IP of the container . In this case the name of redis will be *redis.backend-network*.

```
$ docker run --net=backend-network alpine ping -c1 redis
PING redis (172.19.0.2): 56 data bytes
64 bytes from 172.19.0.2: seq=0 ttl=64 time=0.127 ms

--- redis ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.127/0.127/0.127 ms
```

Again we create a new network with the help of this command which name is *frontend-network.*

Command- $ docker network create frontend-network

```
$ docker network create frontend-network
b46b2277a2d00cb67daf1842d95ddf8828774e441800b7422ef361bd83ffbb3b
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
             NAMES
da6a8a308f1d        redis               "docker-entrypoint.s…"   6 minutes ago       Up 6 minut
p            redis
$
```

- Command:- $docker inspect “container name”

  Docker inspect provides detailed information on constructs controlled by Docker.

```
$ docker inspect redis
[
    {
        "Id": "da6a8a308f1d73193805f630528fd01dba43160079b5e4ff1f25d9583b2d0c2b",
        "Created": "2021-02-05T06:36:24.668473312",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
```

```
        "NetworkMode": "backend-network",
        "PortBindings": {},
        "RestartPolicy": {
            "Name": "no",
            "MaximumRetryCount": 0
        },
        "AutoRemove": false,
        "VolumeDriver": "",
        "VolumesFrom": null,
        "CapAdd": null,
        "CapDrop": null,
        "Dns": [],
        "DnsOptions": [],
        "DnsSearch": [],
        "ExtraHosts": null,
        "GroupAdd": null,
        "IpcMode": "shareable",
        "Cgroup": "",
```


Command:- $ docker network connect frontend-network redis

$ docker inspect redis

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

- Now we can see the backend and frontend network.


```
$ docker network connect frontend-network redis
$ docker inspect redis
[
    {
        "Id": "da6a8a308f1d73193805f630528fd01dba43160079b5e4ff1f25d9583b2d0c2b",
        "Created": "2021-02-05T06:36:24.668473312",
        "Path": "docker-entrypoint.sh",
        "Args": [
            "redis-server"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1019,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2021-02-05T06:36:25.278649295Z",
```

```
        "NetworkMode": "backend-network",
        "PortBindings": {},
        "RestartPolicy": {
            "Name": "no",
            "MaximumRetryCount": 0
        },
        "AutoRemove": false,
        "VolumeDriver": "",
        "VolumesFrom": null,
        "CapAdd": null,
        "CapDrop": null,
        "Dns": [],
        "DnsOptions": [],
        "DnsSearch": [],
        "ExtraHosts": null,
        "GroupAdd": null,
        "IpcMode": "shareable",
        "Cgroup": "",
```

```
"frontend-network": {
    "IPAMConfig": {},
    "Links": null,
    "Aliases": [
        "da6a8a308f1d"
    ],
    "NetworkID": "b46b2277a2d00cb67daf1842d95ddf8828774e441800b7422ef361bd83ff
    "EndpointID": "6769e24e8a7bbb1cbc26764ef4880f4ae5df5c1865280aa2924a39bc81e
    "Gateway": "172.20.0.1",
    "IPAddress": "172.20.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
```


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

$ curl docker:3000

Create Aliases:-

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

Connect container with Alias:-

Create a new network *frontend-network2*

Command:- $ docker network create frontend-network2

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

Command:- $ 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

```
$ docker network create frontend-network2
763208ae7e55750ddd0c7d0f32311930689af8f11d859087b43eb1767d48dcfe
$ docker network connect --alias db frontend-network2 redis
$ 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=0.168 ms

--- db ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.168/0.168/0.168 ms
$
```

To show the list of all the created network:-

Command:- $ docker network ls

```
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
de46c857f3ec        backend-network     bridge              local
a648cec72646        bridge              bridge              local
b46b2277a2d0        frontend-network    bridge              local
763208ae7e55        frontend-network2   bridge              local
0c288d2ed25a        host                host                local
4d3221fe852b        none                null                local
$
```

We can see all the information about frontend-network with the help of this command.

Command:- $ docker network inspect frontend-network

```
$ docker network inspect frontend-network
[
    {
        "Name": "frontend-network",
        "Id": "b46b2277a2d00cb67daf1842d95ddf8828774e441800b7422ef361bd83ffbb3b",
        "Created": "2021-02-05T06:43:08.134765956Z",
        "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,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
```

Disconnect the container:-

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

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