Docker networking

Theory

Default Bridge Network

- Bridge Network: By default, Docker creates a network named bridge when it starts.
- Virtual Switch: This bridge acts as a virtual switch, allowing containers to connect and communicate if they are on the same bridge network.
- Default CIDR: Docker assigns the subnet 172.17.0.0/16 to the default bridge network.
    - Example: Container IPs might look like 172.17.1.6 (Docker Documentation)

IP Allocation

- DHCP Allocation: Each container receives an IP address from the bridge network's subnet when it starts, managed by Docker's built-in DHCP.

Custom Networks

- Custom CIDR: You can create custom networks with specified CIDRs for more control over IP address allocation.

Embedded DNS Server

- Service Discovery: Docker has an embedded DNS server that provides automatic service discovery for containers.
- Dynamic DNS Updates: The DNS server is automatically updated when a new container starts.

Hostnames and Container IDs

- Hostname Resolution: Containers can communicate using hostnames, which by default are the container IDs.

Summary:

- Docker's default bridge network uses 172.17.0.0/16 CIDR.
- Containers receive IPs via DHCP.
- Custom networks with specific CIDRs can be created.
- Docker's embedded DNS server provides automatic service discovery and hostname resolution.
- Containers can communicate using hostnames, typically the container IDs, within the same network.

Lab

- A repo to test our the functionality of networking in Docker

Basic test

- Create 2 containers in same net and ping them for a response.

```
docker network create my_network

docker run -d --name container1 --network my_network nginx
docker run -d --name container2 --network my_network alpine sleep 1000

docker exec container2 ping container1

## Opposite Test:

- Attempt to ping a container not in the same network, and it should fai
```

DNS stuff

- DNS reslution:

```
docker network create mo_net

docker run -d --name webserver --network mo_net nginx
docker run -d --name test_client --network mo_net alpine sleep 1000


docker exec test_client ping -c 4 webserver

PING webserver (172.22.0.2): 56 data bytes
64 bytes from 172.22.0.2: seq=0 ttl=64 time=0.070 ms
64 bytes from 172.22.0.2: seq=1 ttl=64 time=0.119 ms
64 bytes from 172.22.0.2: seq=2 ttl=64 time=0.110 ms
^C
```

HTTP/Curl request (client to web)

- You can also use curl to make an HTTP request to the webserver container from test_client.

```
docker exec test_client apk add --no-cache curl
docker exec test_client curl http://webserver
```

Reponse:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
                                 Dload  Upload   Total   Spent    Left
100   615  100   615    0     0   865k      0 --:--:-- --:--:-- --:--:--
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Advanced Docker networking

```
## You can inspect the network to see its configuration and connected co
docker network create mo_net # if cleaned up from earlier
docker network inspect mo_net


## Create and Use a Custom Network with Specific Subnet:
docker network create --subnet=192.168.1.0/24 my_custom_network ## a cus
docker inspect network my_custom_network # verify that your network has
docker run -d --name custom_webserver --network my_custom_network nginx
docker run -d --name custom_client --network my_custom_network alpine sl

## If you have multiple instances of a service, Docker's DNS server can
docker run -d --name webserver1 --network mo_net nginx
docker run -d --name webserver2 --network mo_net nginx
docker run -d --name webserver3 --network mo_net nginx

## use the test_client container to perform DNS resolution multiple time
docker exec test_client nslookup webserver

## Connecting Containers Across Multiple Networks - A container can be c

docker network create my_second_network
docker network connect my_second_network test_client

docker inspect test_client | grep "Networks" -A 20

## Check net connectivvity

docker exec test_client ping webserver
docker exec test_client curl -I http://webserver

# optinola - you can specify your custom DNS server
docker run -d --name dns_test --dns 8.8.8.8 --network mo_net alpine slee

# use custom DNS for one container
docker run --dns 8.8.8.8 nicolaka/netshoot nslookup google.com

## cleanup
docker rm -f webserver test_client webserver1 webserver2 webserver3 cust
docker network rm mo_net my_custom_network my_second_network

# or do :
docker network prune
docker rm -f $(docker ps -a -q)
docker image prune -a
```

Other lab

```
## Create the frontend network
## Use the docker network to create the frontend network:

docker network create frontend

## Create the localhost network
# User the docker network command to create the localhost network:

docker network create localhost --internal

##Create a MySQL container
##Create a MySQL container that is attached to the localhost network:

docker container run -d --name database --network localhost -e MYSQL_ROO

## Create an Nginx container
## Create an Nginx container that is attached to the localhost network:

docker container run -d --name frontend-app --network frontend nginx:lat

## Connect frontend-app to the internal network
## Connect frontend-app to the localhost network:

docker network connect localhost frontend-app
```

Creating a svc and logging (docker swarm)

```
docker service create --name log-svc --replicas 3 -p 8080:80 nginx ## us
curl localhost:8080

docker service logs log-svc
sudo journalctl -u docker
```