Docker Networking Hands-on Lab

Section 1 - Networking Basics

Step 1: The Docker Network Command

The main command to configuring and managing container network is

docker network

Step 2: List networks

View existing container network on the current Docker host:

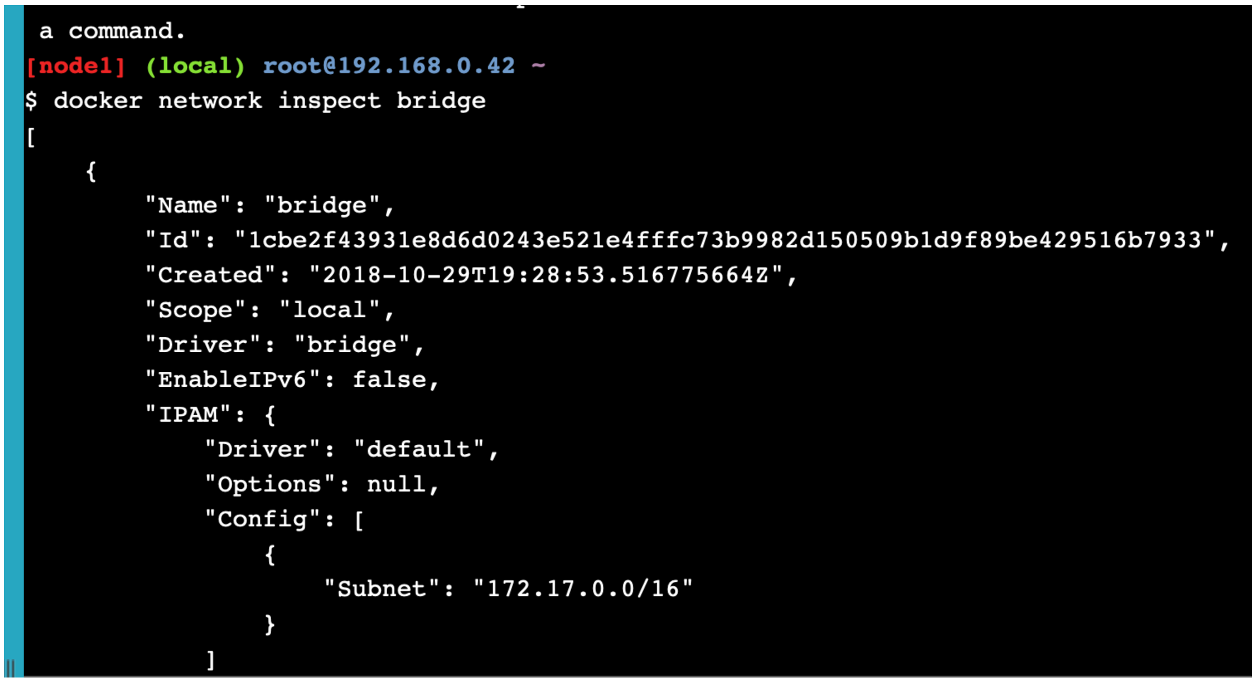
docker network ls

Step 3: Inspect a network

The command to inspect network configuration detail:

docker network inspect bridge
docker network inspect <network>

<network> can be either network name or network ID.



```
 a command.
[node1] (local) root@192.168.0.42 ~
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "1cbe2f43931e8d6d0243e521e4fffc73b9982d150509b1d9f89be429516b7933",
        "Created": "2018-10-29T19:28:53.516775664Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
```

Step 4: List network driver plugins

Use command shows a lot of interesting information about a Docker installation:

docker info

Section 2: Bridge Networking

Step 1: The Basics

Every clean installation of Docker comes with a pre-built network called bridge. Verify this with the:

docker network ls

All networks created with the bridge driver are based on a Linux bridge.

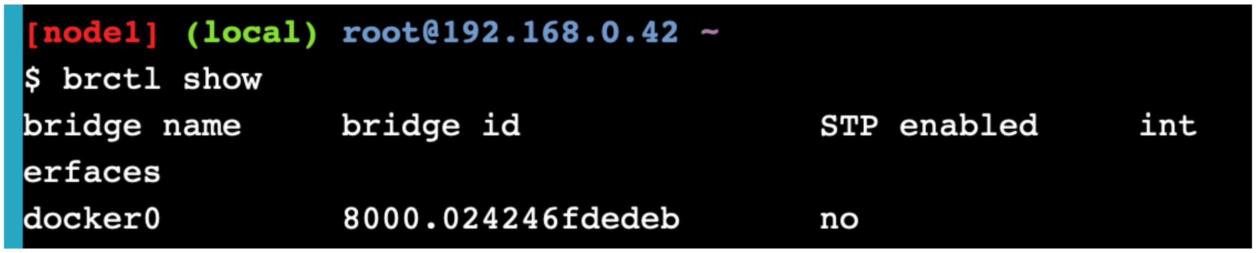
Install the brctl command and use it to list the Linux bridges on your Docker host.

sudo apt-get install bridge-utils

apk update
apk add bridge

Then, list the bridges on your Docker host, by running brctl show.

brctl show



```
[node1] (local) root@192.168.0.42 ~
$ brctl show
bridge name     bridge id               STP enabled     int
erfaces
docker0         8000.024246fdedeb       no
```

Step 2: Connect a container

The bridge network is the default network for new containers. This means that unless you specify a different network, all new containers will bee connected to the bridge network.

Create a new container by running:

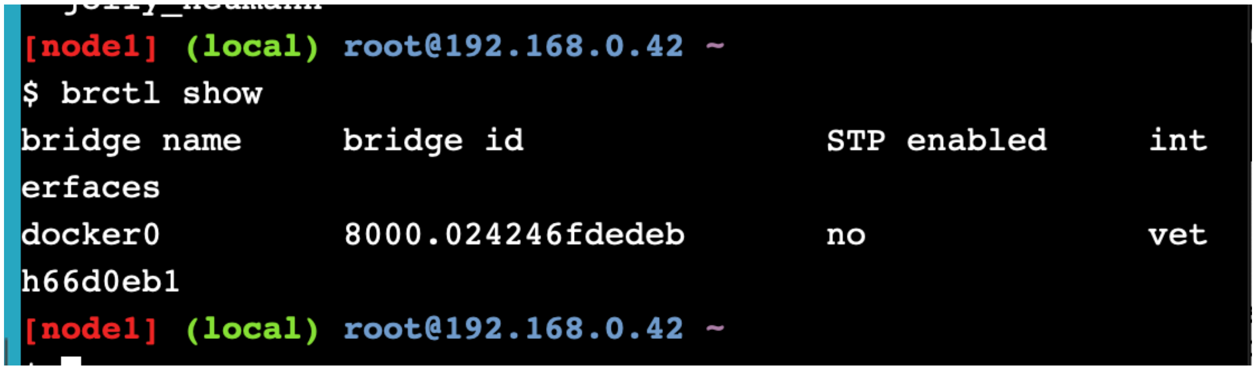
docker run -dt ubuntu sleep infinity

To verify example container is up by running:

docker ps

As no network was specified on the docker run command, the container will be added to the bridge network

brctl show



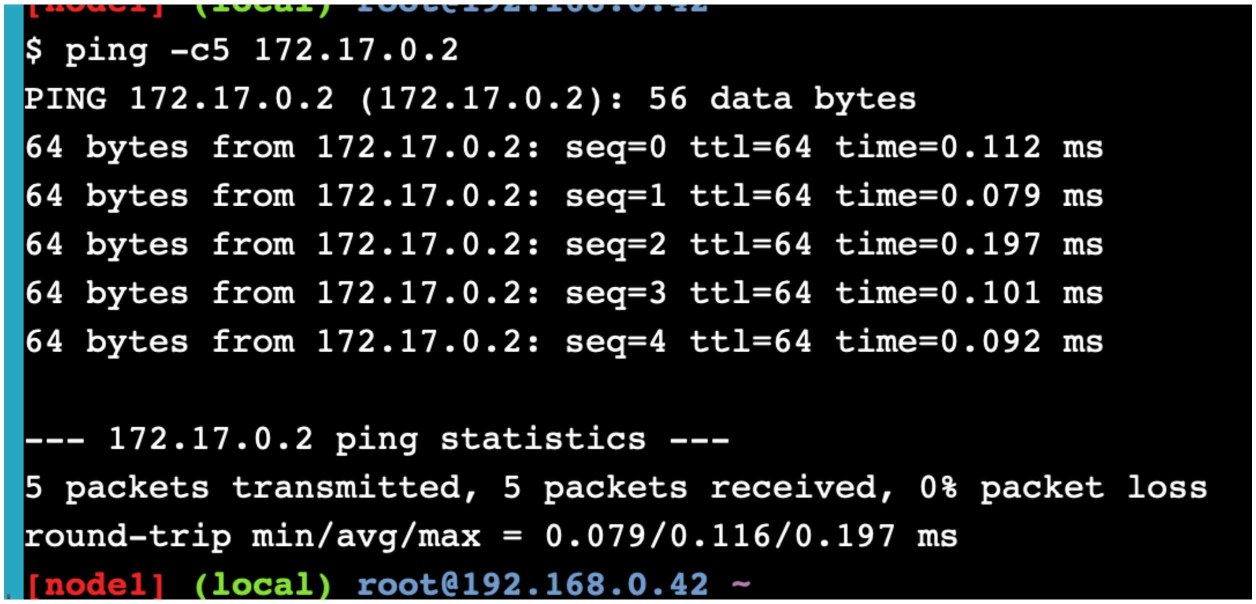
```
[node1] (local) root@192.168.0.42 ~
$ brctl show
bridge name     bridge id               STP enabled     int
erfaces
docker0         8000.024246fdedeb       no              vet
h66d0eb1
[node1] (local) root@192.168.0.42 ~
```

Notice this time docker0 bridge now has an interface connected. This interface connects the docker0 bridge to the new container just created.

Step 3: Test network connectivity

Ping the IP address of the container from the shell prompt of your Docker host by running:

ping -c5 172.17.0.2



```
$ ping -c5 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.112 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.079 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.197 ms
64 bytes from 172.17.0.2: seq=3 ttl=64 time=0.101 ms
64 bytes from 172.17.0.2: seq=4 ttl=64 time=0.092 ms

--- 172.17.0.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.079/0.116/0.197 ms
[node1] (local) root@192.168.0.42 ~
```

The replies above that show that the Docker host can ping the container over the bridge network.

But we can also verify the container can connect to the outside world too. To get the ID of the container started in the previous step:

docker ps

Next, run a steal inside that ubuntu container, by running:

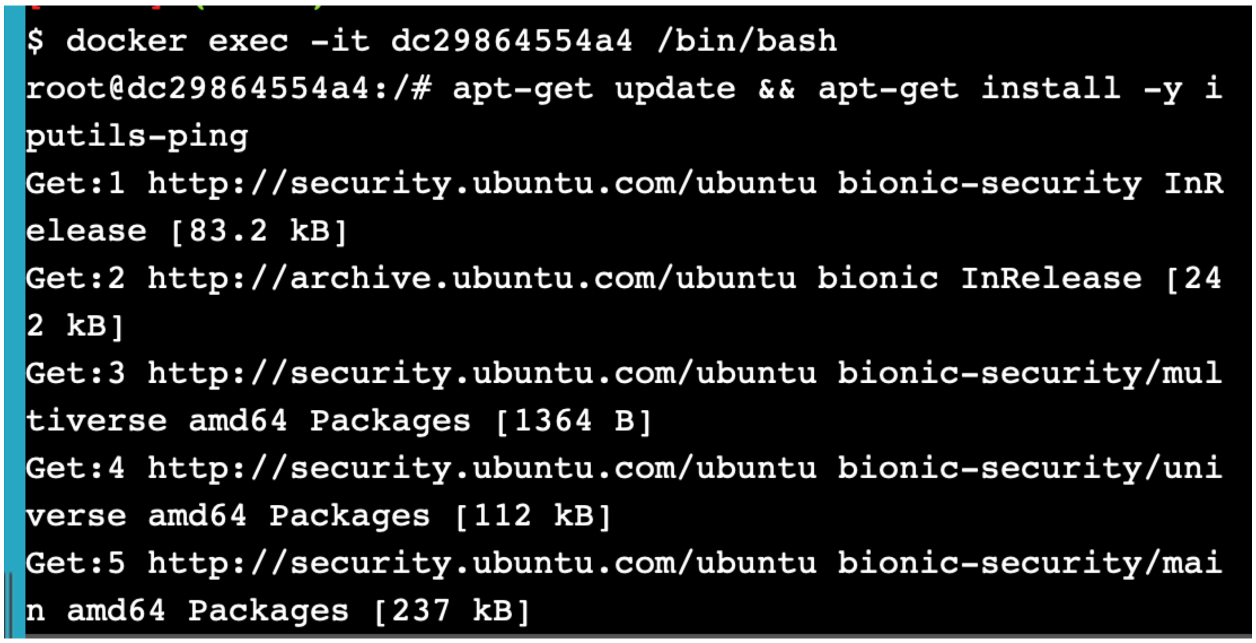
docker exec -it dc29864554a4 /bin/bash

Now we have root authority, then install the ping program:

apt-get update && apt-get install -y iputils-ping

Lets ping www.github.com by running:

ping -c5 www.github.com



```
$ docker exec -it dc29864554a4 /bin/bash
root@dc29864554a4:/# apt-get update && apt-get install -y i
putils-ping
Get:1 http://security.ubuntu.com/ubuntu bionic-security InR
elease [83.2 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [24
2 kB]
Get:3 http://security.ubuntu.com/ubuntu bionic-security/mul
tiverse amd64 Packages [1364 B]
Get:4 http://security.ubuntu.com/ubuntu bionic-security/uni
verse amd64 Packages [112 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security/mai
n amd64 Packages [237 kB]
```

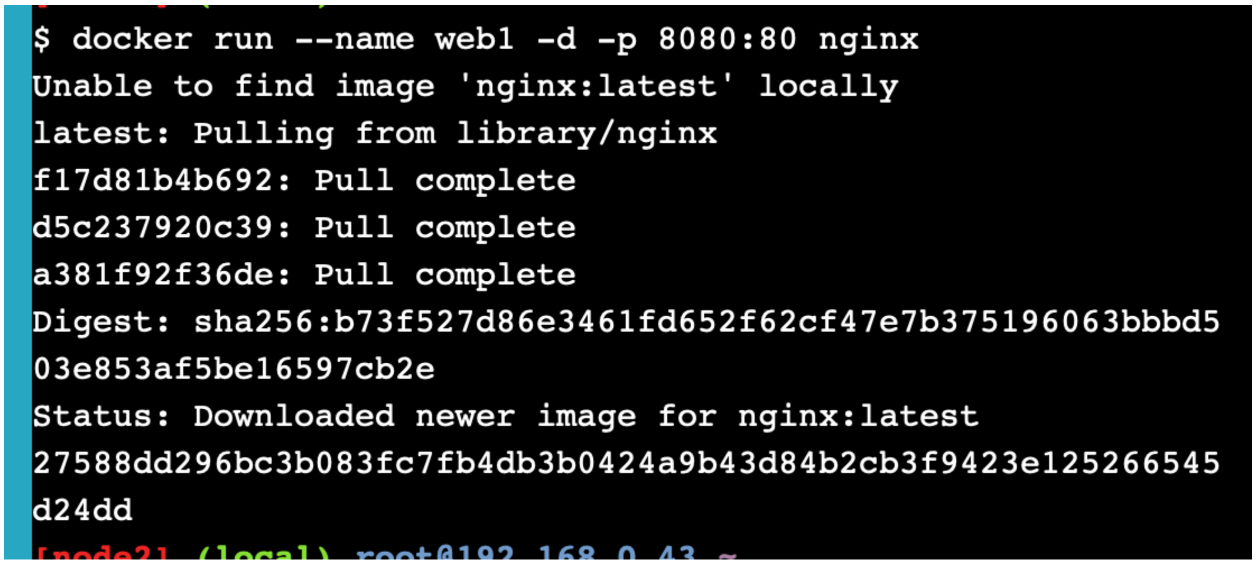
Step 4: Configure NAT for external connectivity

In this step we'll start a new NGINX container and map port 8080 on the Docker host to port 80 inside of the container. This means that traffic that hits the Docker host on port 8080 will be passed on to port 80 inside the container.
Start a new container based off the official NGINX image by running:

docker run — name web1 -d -p 8080:80 nginx

Review the container status and port mappings by running:

docker ps



```
$ docker run --name web1 -d -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
f17d81b4b692: Pull complete
d5c237920c39: Pull complete
a381f92f36de: Pull complete
Digest: sha256:b73f527d86e3461fd652f62cf47e7b375196063bbbd5
03e853af5be16597cb2e
Status: Downloaded newer image for nginx:latest
27588dd296bc3b083fc7fb4db3b0424a9b43d84b2cb3f9423e125266545
d24dd
[node2] (local) root@192.168.0.43 ~
```

The top line shows the new web1 container running NGINX. Take note of the command the container is running as well as the port mapping - 0.0.0.0:8080->80/tcp maps port 8080 on all host interfaces to port 80 inside the web1 container. This port mapping is what effectively makes the containers web service accessible from external sources.

Now that the container is running and mapped to a port on a host interface you can test connectivity to the NGINX Webb server.

To complete the following task you will need the IP address of your Docker host. This will need to bbbe an IP address that you can reach.

Section #3 - Overlay Networking

Step 1: The Basics

In this step you'll initialize a new Swarm, join a single worker node, and verify the operations worked, Run:

docker swarm init —advertise-addr $(hostname -i)

Run command to verify that both nodes are part of the Swarm:

docker node ls

Step 2: Create an overlay network

Now that you have a Swarm initialized it's time to create an overlay network

Create a new overlay called "overnet" by running:

```
docker network create -d overlay overset
```

Use the command to verify the network was created successfully:

```
docker network ls
```

The new "overnet" network is shown on the last line of the output above.

Run the same command from the second terminal:

```
docker network ls
```

Notice that the "overnet" does not appear in the list. This is because Docker only extends overlay networks to hosts when they are needed. This is usually when a host runs a task from the service that is created on the network.

Use the command to view more detailed information about the "overnet" network:

```
docker network inspect overset
```

Step 3: Create a service

Now that we have a Swarm initialized and an overlay network, it's time to create a service that uses the network.

Execute the following command from the first terminal to create a new service called my service on the overset with two tasks/replicas.

```
docker service create —name myservice \
—network overnet \
—replicas 2\
ubuntu sleep infinity
```

Verify that the service is created and both replicas are up by running:

```
docker service ls
```

The 2/2 in the REPLICAS column shows that both tasks in the service are up and running.

Verify that a single task is running on each of the two nodes in the Swarm by running:

```
docker service ps myservice
```

Step 4: Test the network

To complete this step you will need to IP address of the service task running on node2 that you saw in the previous step.

Execute the following commands from the first terminal:

```
docker network inspect overset
```

Notice that the IP address for the service task running is different to the IP address for the service task running on the second node. Note also that they are on the same “overnet” network.

To get the ID of the service task so that you can log in to it in the next step:

```
docker ps
```

Log on to the service task. Be sure to use the container ID from your environment as it will be different from the example shown below. We can do this by running:

```
docker exec -it d676494d18f7 /bin/bash
```

Install the ping command and ping the service task running on the second node where it had a IP address of 10.0.0.3 from the command:

```
apt-get update && apt-get install -y iputils-ping
```

Now ping 10.0.0.3

Step 5: Test service discovery

Run:

```
cat /etc/resolv.conf
search ivaf2i2atqouppoxund0tvddsa.jx.internal.cloudapp.net
nameserver 127.0.0.11
options nodes:0
```

They and ping the “myservice” name from within the container by running:

```
ping -c5 myservice
```

The output clearly shows that the container can ping the myservice service by name. Notice that the IP address returned is 10.0.0.2. In the next few steps we’ll verify that this address is the virtual IP assign to the myservice service.