Section #1

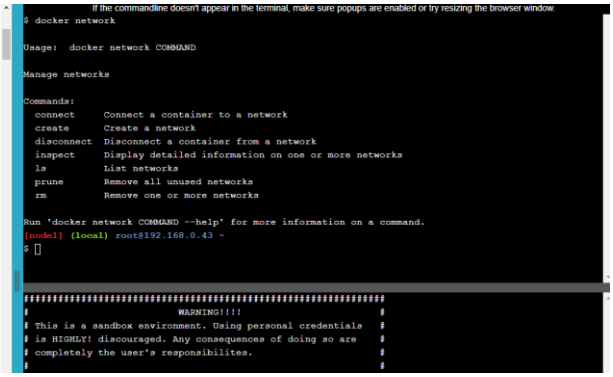
Step 1: The Docker Network Command

The `docker network` command is the main command for configuring and managing container networks. Run the `docker network` command from the first terminal.

```
docker network
```

```
Usage:  docker network COMMAND

Manage networks

Options:
      --help   Print usage

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

The command output shows how to use the command as well as all of the `docker network` sub-commands. As you can see from the output, the `docker network` command allows you to create new networks, list existing networks, inspect networks, and remove networks. It also allows you to connect and disconnect containers from networks.

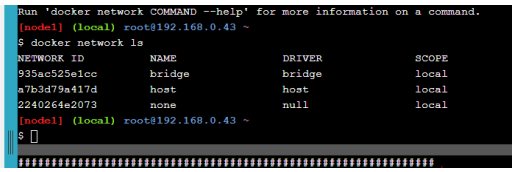


Step 2: List networks

Run a `docker network ls` command to view existing container networks on the current Docker host.

```
docker network ls
```

```
NETWORK ID          NAME                DRIVER              SCOPE
3430ad6f20bf        bridge              bridge              local
a7449465c379        host                host                local
06c349b9cc77        none                null                local
```



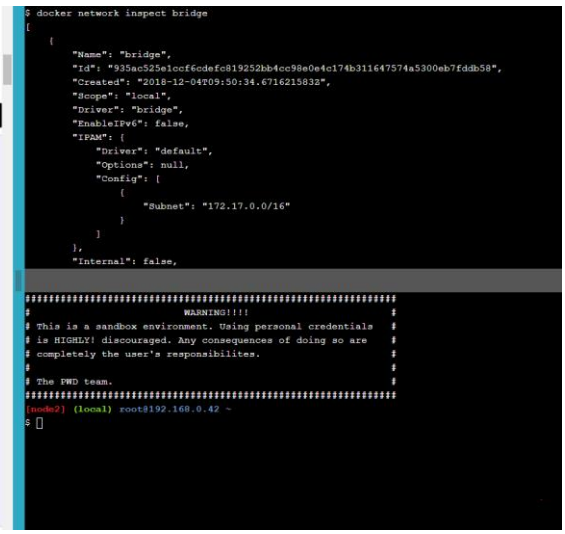
Step 3: Inspect a network

The `docker network inspect` command is used to view network configuration details. These details include; name, ID, driver, IPAM driver, subnet info, connected containers, and more.

Use `docker network inspect <network>` to view configuration details of the container networks on your Docker host. The command below shows the details of the network called `bridge`.

```
docker network inspect bridge
```

```
[
    {
        "Name": "bridge",
        "Id": "3430ad6f20bf1486df2e5f64ddc93cc4ff95d81f59b6baea8a510ad500df2e57",
        "Created": "2017-04-03T16:49:58.6536278Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```



```
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {},
    "Options": {
        "com.docker.network.bridge.default_bridge": "true",
        "com.docker.network.bridge.enable_icc": "true",
        "com.docker.network.bridge.enable_ip_masquerade": "true",
        "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
        "com.docker.network.bridge.name": "docker0",
        "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
}
```


Step 4: List network driver plugins

The `docker info` command shows a lot of interesting information about a Docker installation.

Run the `docker info` command and locate the list of network plugins.

```
docker info
```

```
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 17.03.1-ee-3
Storage Driver: aufs
<Snip>
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: runc
<Snip>
```

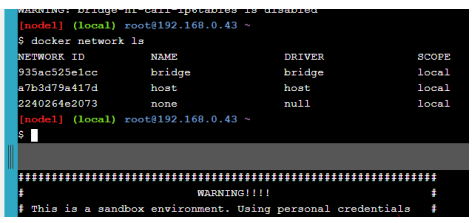
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the brow

```
[node1] (local) root@192.168.0.43 ~
$ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.06.1-ce
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
```

```
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 468a545b9edcd5932818eb9de8e72413e616e86e
runc version: 69663f0bd4b60df09991c08812a60108003fa340
init version: fec3683
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.4.0-96-generic
Operating System: Alpine Linux v3.8 (containerized)
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 31.4GiB
Name: node1
ID: SUXR:D25S:53OI:7HT4:DVCG:O7XB:EQQR:HAMU:GE7P:RNPA:Z7RM:7NTZ
```

```
ID: SUXR:D25S:53OI:7HT4:DVCG:O7XB:EQQR:HAMU:GE7P:RNPA:Z7RM:7NTZ
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): true
 File Descriptors: 23
 Goroutines: 45
 System Time: 2018-12-04T09:59:39.910646645Z
 EventsListeners: 0
Registry: https://index.docker.io/v1/
Labels:
Experimental: true
Insecure Registries:
 127.0.0.1
 127.0.0.0/8
Live Restore Enabled: false
```

Section #2



```
brctl show
```

```
bridge name     bridge id               STP enabled     interfaces
docker0         8000.024252ed52f7       no
```

The output above shows a single Linux bridge called **docker0**. This is the bridge that was automatically created for the **bridge** network. You can see that it has no interfaces currently connected to it.

You can also use the `ip a` command to view details of the **docker0** bridge.

```
ip a
```

```
<Snip>
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:52:ed:52:f7 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
```

Step 2: Connect a container

```
v3.8.1-112-g45bdd0edfb [http://dl-cdn.alpinelinux.org/alpine/v3.8/community]
OK: 9555 distinct packages available
[node1] (local) root@192.168.0.43 ~
$ apk add bridge
OK: 302 MiB in 111 packages
[node1] (local) root@192.168.0.43 ~
$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242225438aa       no
[node1] (local) root@192.168.0.43 ~
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:22:54:38:aa brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
```

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 be connected to the **bridge** network.

Create a new container by running `docker run -dt ubuntu sleep infinity`.

```
docker run -dt ubuntu sleep infinity
```

```
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
d54efb8db41d: Pull complete
f8b845f45a87: Pull complete
e8db7bf7c39f: Pull complete
9654c40e9079: Pull complete
6d9ef359eaaa: Pull complete
Digest: sha256:dd7808d8792c9841d0b460122f1acf0a2dd1f56404f8d1e56298048885e45535
Status: Downloaded newer image for ubuntu:latest
846af8479944d406843c90a39cba68373c619d1feaa932719260a5f5afddbf71
```

```
    inet 192.168.0.43/23 scope global eth0
       valid_lft forever preferred_lft forever
1822: eth1@if1823: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:ac:12:00:25 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.37/16 scope global eth1
       valid_lft forever preferred_lft forever
[node1] (local) root@192.168.0.43 ~
$ docker run -dt ubuntu sleep infinity
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
da1315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db9489744ecc9b4f50c7fa671f23c49
Status: Downloaded newer image for ubuntu:latest
2d0df0c6d518d03cdaa71703eb37d2314dd82b73bfb7131a05ae4829d7c39430
```

```
docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
846af8479944        ubuntu              "sleep infinity"    55 seconds ago      Up 54 seconds                           heuristic_boyd
```

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

Run the `brctl show` command again.

```
brctl show
```

```
bridge name     bridge id               STP enabled     interfaces
docker0         8000.024252ed52f7       no              vethd630437
```

```
[node1] (local) root@192.168.0.43 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
2d0df0c6d518        ubuntu              "sleep infinity"    36 seconds ago      Up 35 seconds                           angry_swirles
[node1] (local) root@192.168.0.43 ~
$ brctl show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.0242225438aa       no              vethb2dfad4
[node1] (local) root@192.168.0.43 ~
$
```

```
docker network inspect bridge
```

```
<Snip>
        "Containers": {
            "846af8479944d406843c90a39cba68373c619d1feaa932719260a5f5afddbf71": {
                "Name": "heuristic_boyd",
                "EndpointID": "1265c418f0b812004d80336bafdc4437eda976f166c11dbcc97d365b2bfa91e5",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
<Snip>
```

Step 3: Test network connectivity

The output to the previous `docker network inspect` command shows the IP address of the new container. In the previous example it is `172.17.0.2` but yours might be different.

Ping the IP address of the container from the shell prompt of your Docker host by running `ping -c5 <IPv4 Address>`. Remember to use the IP of the container

```
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "935ac525e1ccf6cdefc819252bb4cc98e0e4c174b311647574a5300eb7fddb58",
        "Created": "2018-12-04T09:50:34.6716215832",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16"
                }
            ]
        },
        "Internal": false,
```

```
"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
    "Network": ""
},
"ConfigOnly": false,
"Containers": {
    "2d0df0c6d518d03cdaa71703eb37d2314dd82b73bfb7131a05ae4829d7c39430": {
        "Name": "angry_swirles",
        "EndpointID": "788cb78559724bdb3413476f65a030bb265f54e423d555e528cfe3ddc04c181f"

        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    }
},
"Options": {
    "com.docker.network.bridge.default_bridge": "true",
```

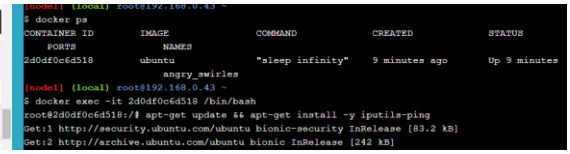
```
    "Options": {
        "com.docker.network.bridge.default_bridge": "true",
        "com.docker.network.bridge.enable_icc": "true",
        "com.docker.network.bridge.enable_ip_masquerade": "true",
        "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
        "com.docker.network.bridge.name": "docker0",
        "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
}
```

```
$ 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=0.154 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.123 ms
64 bytes from 172.17.0.2: seq=2 ttl=64 time=0.100 ms
64 bytes from 172.17.0.2: seq=3 ttl=64 time=0.177 ms
64 bytes from 172.17.0.2: seq=4 ttl=64 time=0.107 ms

--- 172.17.0.2 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.100/0.132/0.177 ms
```

```
docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
846af8479944        ubuntu              "sleep infinity"    7 minutes ago       Up 7 minutes                            heuristic_boyd
```

Next, lets run a shell inside that ubuntu container, by running `docker exec -it <CONTAINER ID> /bin/bash`.

```
docker exec -it yourcontainerid /bin/bash
root@846af8479944:/#
```



```
[node1] (local) root@192.168.0.43 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
2d0df0c6d518        ubuntu              "sleep infinity"    9 minutes ago       Up 9 minutes                            angry_swirles
[node1] (local) root@192.168.0.43 ~
$ docker exec -it 2d0df0c6d518 /bin/bash
root@2d0df0c6d518:/# apt-get update && apt-get install -y iputils-ping
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [83.2 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
```

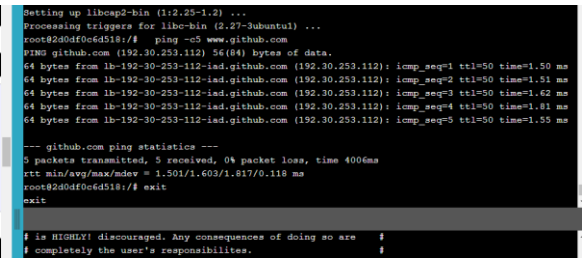
```
apt-get update && apt-get install -y iputils-ping
```

Lets ping www.github.com by running `ping -c5 www.github.com`

```
ping -c5 www.github.com

PING www.docker.com (104.239.220.248) 56(84) bytes of data.
64 bytes from 104.239.220.248: icmp_seq=1 ttl=45 time=38.1 ms
64 bytes from 104.239.220.248: icmp_seq=2 ttl=45 time=37.3 ms
64 bytes from 104.239.220.248: icmp_seq=3 ttl=45 time=37.5 ms
64 bytes from 104.239.220.248: icmp_seq=4 ttl=45 time=37.5 ms
64 bytes from 104.239.220.248: icmp_seq=5 ttl=45 time=37.5 ms

--- www.docker.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
rtt min/avg/max/mdev = 37.372/37.641/38.143/0.314 ms
```

Finally, lets disconnect our shell from the container, by running `exit`.

```
exit
```



```
Setting up libcap2-bin (1:2.25-1.2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
root@2d0df0c6d518:/#   ping -c5 www.github.com
PING github.com (192.30.253.112) 56(84) bytes of data.
64 bytes from lb-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=1 ttl=50 time=1.50 ms
64 bytes from lb-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=2 ttl=50 time=1.51 ms
64 bytes from lb-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=3 ttl=50 time=1.62 ms
64 bytes from lb-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=4 ttl=50 time=1.81 ms
64 bytes from lb-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=5 ttl=50 time=1.55 ms

--- github.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 1.501/1.603/1.817/0.118 ms
root@2d0df0c6d518:/# exit
exit

# is HIGHLY! discouraged. Any consequences of doing so are    #
# completely the user's responsibilites.                      #
```

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.

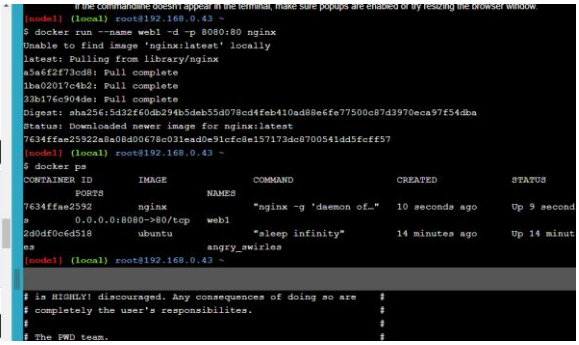
> **NOTE:** If you start a new container from the official NGINX image without specifying a command to run, the container will run a basic web server on port 80.

Start a new container based off the official NGINX image by running `docker run --name web1 -d -p 8080:80 nginx`.

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

Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6d827a3ef358: Pull complete
b5568b1c7952: Pull complete
03558b976e24: Pull complete
9abee7e1ef9d: Pull complete
Digest: sha256:52f84ace6ea43f2f58937e5f9fc562e99ad6876e82b99d171916c1ece587c188
Status: Downloaded newer image for nginx:latest
4e0da45b0f169f18b0e1ee9bf779500cb0f756402c0a0821d55565f162741b3e
```

Review the container status and port mappings by running `docker ps`.

```
docker ps
```

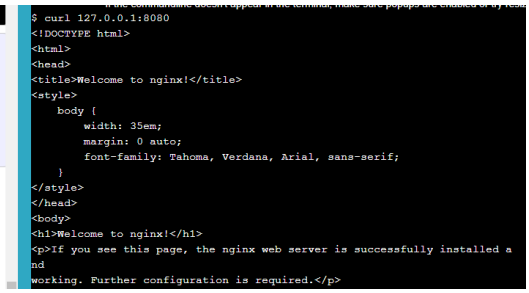


```
[node1] (local) root@192.168.0.43 ~
$ docker run --name web1 -d -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
a5a6f2f73cd8: Pull complete
1ba02017c4b2: Pull complete
33b176c904de: Pull complete
Digest: sha256:5d32f60db294b5deb55d078cd4feb410ad88e6fe77500c87d3970eca97f54dba
Status: Downloaded newer image for nginx:latest
7634ffae25922a8a08d00678c031ead0e91cfc8e157173dc8700541dd5fcff57
[node1] (local) root@192.168.0.43 ~
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                  NAMES
7634ffae2592        nginx               "nginx -g 'daemon of…"   10 seconds ago      Up 9 seconds        0.0.0.0:8080->80/tcp   web1
2d0df0c6d518        ubuntu              "sleep infinity"         14 minutes ago      Up 14 minutes                              angry_swirles
[node1] (local) root@192.168.0.43 ~

# is HIGHLY! discouraged. Any consequences of doing so are    #
# completely the user's responsibilites.                      #
#                                                             #
# The BWD team.                                               #
```

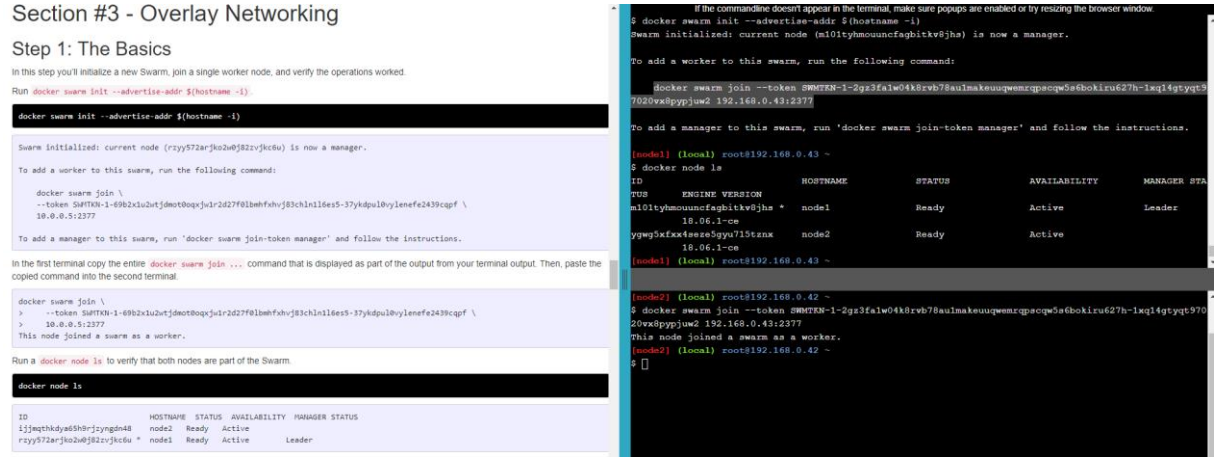
If you try and curl the IP address on a different port number it will fail.

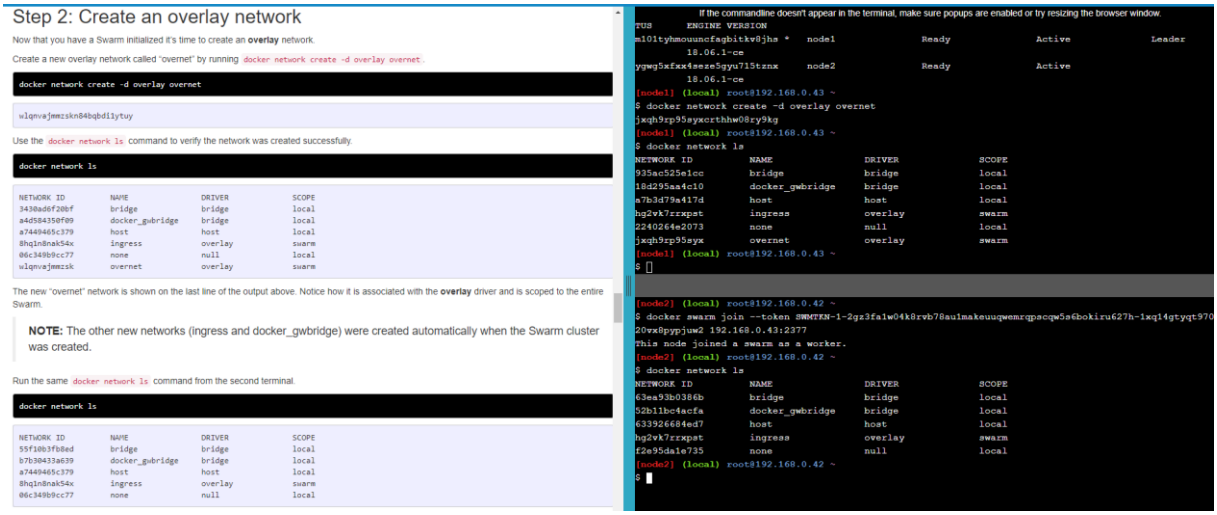
> **NOTE:** The port mapping is actually port address translation (PAT).

Section #3 - Overlay Networking

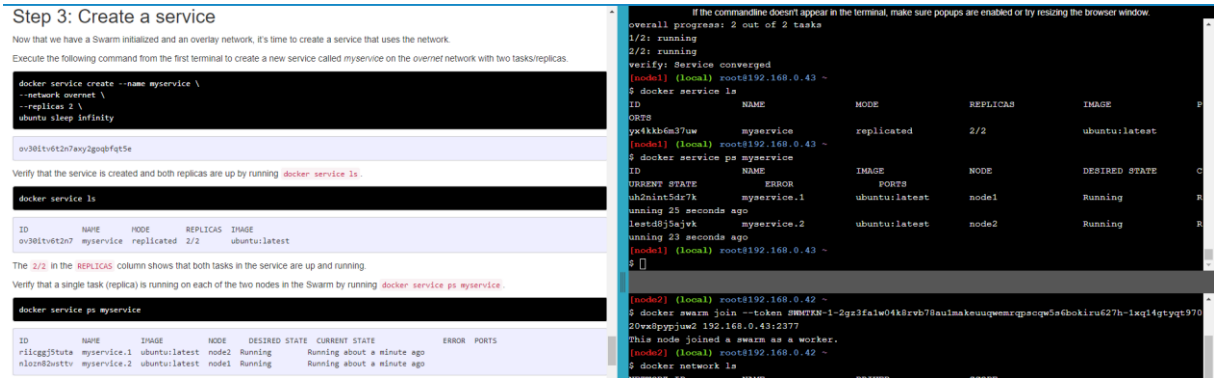


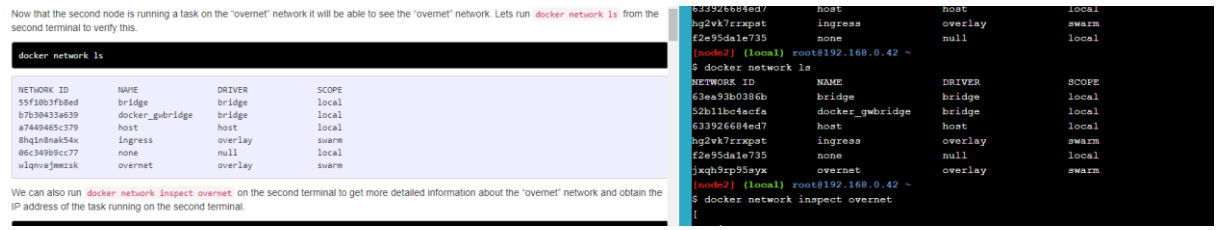
Section #3





```
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": null,
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4097"
        },
        "Labels": null
    }
```



```
$ docker network inspect overnet
[
    {
        "Name": "overnet",
        "Id": "jxqh9rp95syxcrthhw08ry9kg",
        "Created": "2018-12-04T10:41:08.022472593Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
```

```
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "96b24868fb8186dfbdf842668072e9373300fcf7bb5dea2173e518e50702776e": {
            "Name": "myservice.2.lestd8j5ajvkpg2o40ku8v3hn",
            "EndpointID": "533dabcd2703ae49af804a259c3b19bc7b9e3fa44ee5f3cbe5b383b22f094f90",
            "MacAddress": "02:42:0a:00:00:06",
            "IPv4Address": "10.0.0.6/24",
            "IPv6Address": ""
        },
        "lb-overnet": {
            "Name": "overnet-endpoint",
            "EndpointID": "2b61c5204e2e0be91006716ecdac4ceec9a22216a5e5f3628edd0be4f394fcb9",
            "MacAddress": "02:42:0a:00:00:03",
```

```
            "IPv4Address": "10.0.0.6/24",
            "IPv6Address": ""
        },
        "lb-overnet": {
            "Name": "overnet-endpoint",
            "EndpointID": "2b61c5204e2e0be91006716ecdac4ceec9a22216a5e5f3628edd0be4f394fcb9",
            "MacAddress": "02:42:0a:00:00:03",
            "IPv4Address": "10.0.0.3/24",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.driver.overlay.vxlanid_list": "4097"
    },
    "Labels": {},
    "Peers": [
        {
```



```
        "Peers": [
            {
                "Name": "f068a23e3b24",
                "IP": "192.168.0.42"
            },
            {
                "Name": "0f2958ee1888",
                "IP": "192.168.0.43"
            }
        ]
    }
```


Step 4: Test the network

To complete this step you will need the IP address of the service task running on **node2** that you saw in the previous step (`10.0.0.3`).

Execute the following commands from the first terminal.

```
docker network inspect overnet
```


```
[
    {
        "Name": "overnet",
        "Id": "wlqnvajmmzskn84bqbdi1ytuy",
        "Created": "2017-04-04T09:35:47.362263887Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
```

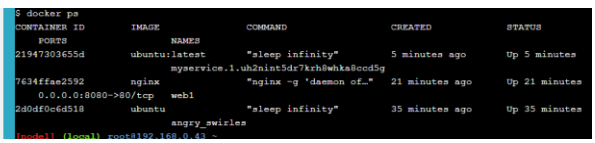


```
$ docker network inspect overnet
[
    {
        "Name": "overnet",
        "Id": "jxqh9rp95syxcrthhw08ry9kg",
        "Created": "2018-12-04T10:41:08.022138896Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
```

```
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
        "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
        "21947303655d7ec1f32ffdcb01aab294ce33499524f9e980957f74929ad33376": {
            "Name": "myservice.1.uh2nint5dr7krh8whka8ccd5g",
            "EndpointID": "799dddaea096246060dabf01003e0c7cf4018ff7f46c4cea1b82b4e606d29182",
            "MacAddress": "02:42:0a:00:00:05",
            "IPv4Address": "10.0.0.5/24",
            "IPv6Address": ""
        },
        "lb-overnet": {
            "Name": "overnet-endpoint",
            "EndpointID": "07f1ee4a031f1d82b2091ae3999ae7cc162804a69521acc2074341d1771df87e",
            "MacAddress": "02:42:0a:00:00:02",
```

```
            "IPv4Address": "10.0.0.5/24",
            "IPv6Address": ""
        },
        "lb-overnet": {
            "Name": "overnet-endpoint",
            "EndpointID": "07f1ee4a031f1d82b2091ae3999ae7cc162804a69521acc2074341d1771df87e",
            "MacAddress": "02:42:0a:00:00:02",
            "IPv4Address": "10.0.0.2/24",
            "IPv6Address": ""
        }
    },
    "Options": {
        "com.docker.network.driver.overlay.vxlanid_list": "4097"
    },
    "Labels": {},
    "Peers": [
```

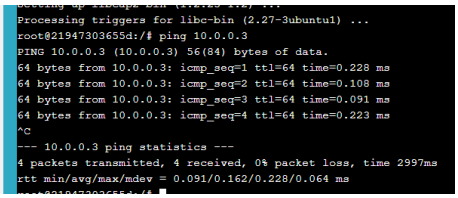
```
    "Peers": [
        {
            "Name": "0f2958ee1888",
            "IP": "192.168.0.43"
        },
        {
            "Name": "f068a23e3b24",
            "IP": "192.168.0.42"
        }
    ]
}
```

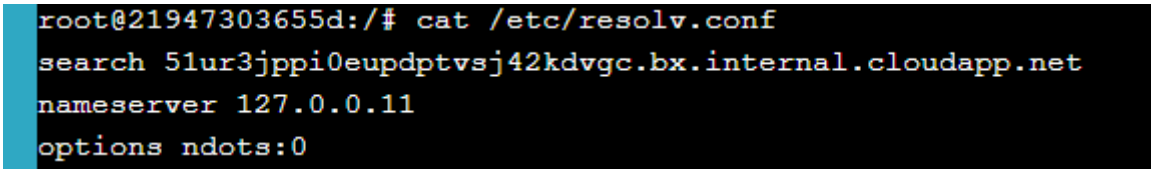
Now, lets ping `10.0.0.3`.

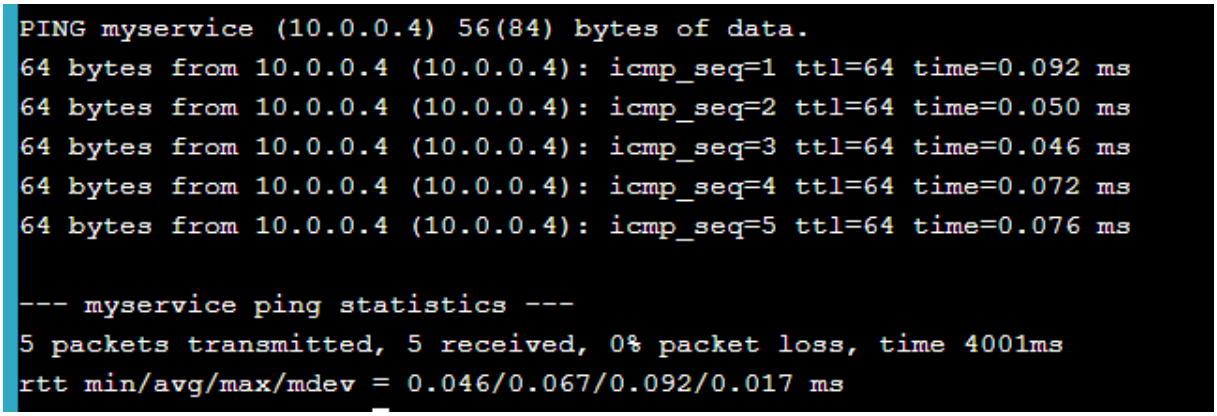
```
root@d676496d18f7:/# ping -c5 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 2998ms
```

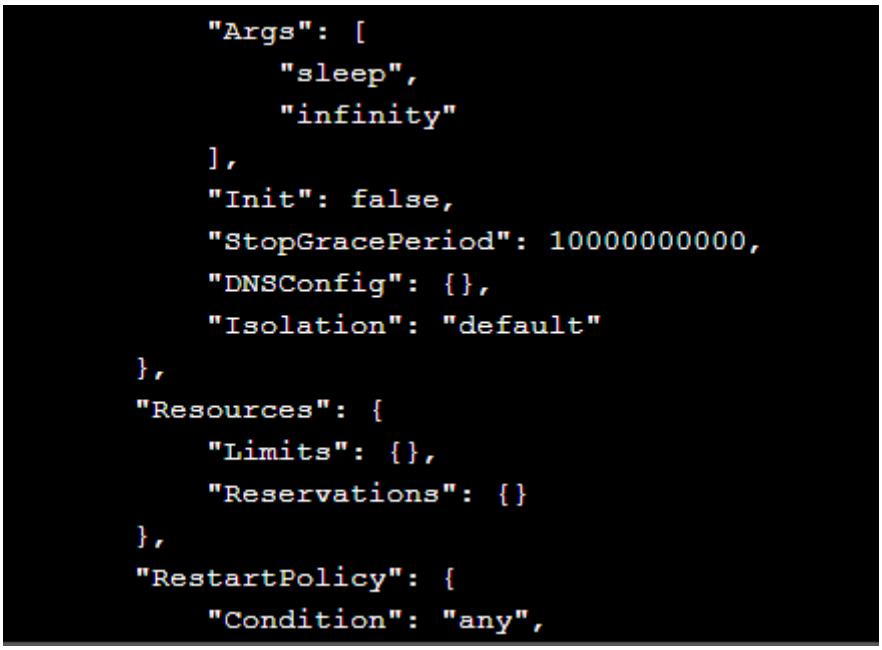
The output above shows that both tasks from the **myservice** service are on the same overlay network spanning both nodes and that they can use this network to communicate.



Step 5: Test service discovery

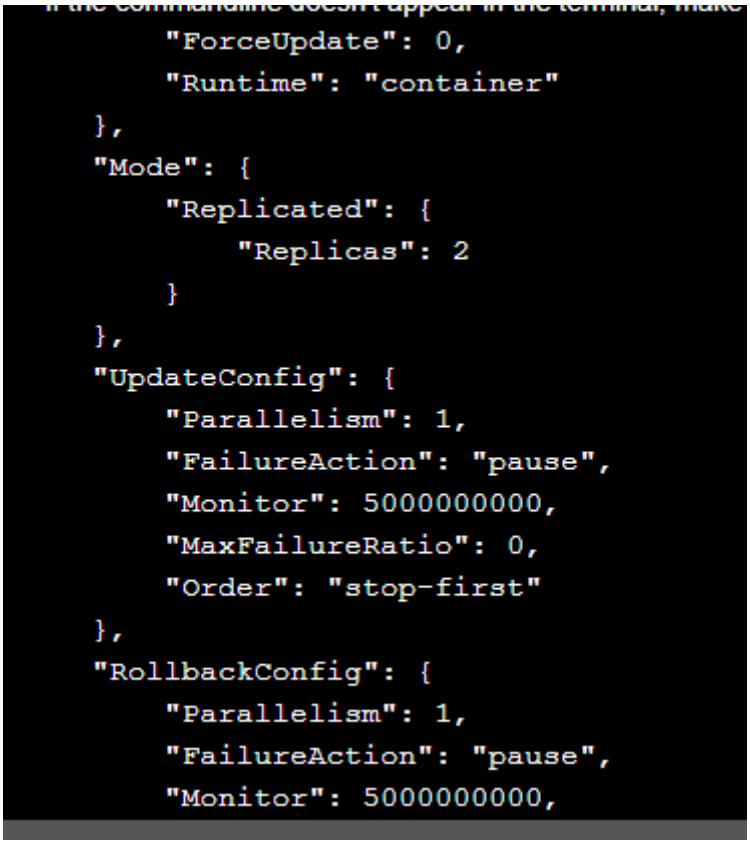





"Args": [
"sleep",
"infinity"
],
"Init": false,
"StopGracePeriod": 10000000000,
"DNSConfig": {},
"Isolation": "default"
},
"Resources": {
"Limits": {},
"Reservations": {}
},
"RestartPolicy": {
"Condition": "any",


"RestartPolicy": {
"Condition": "any",
"Delay": 5000000000,
"MaxAttempts": 0
},
"Placement": {
"Platforms": [
{
"Architecture": "amd64",
"OS": "linux"
},
{
"OS": "linux"
},
{
"Architecture": "arm64",
"OS": "linux"
},
{

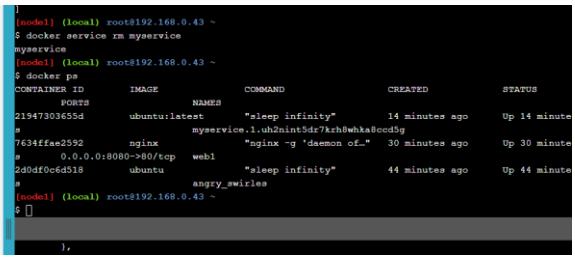

"Architecture": "386",
"OS": "linux"
},
{
"Architecture": "ppc64le",
"OS": "linux"
},
{
"Architecture": "s390x",
"OS": "linux"
}
]
},
"Networks": [
{
"Target": "jxqh9rp95syxcrthhw08ry9kg"
}
],
"ForceUpdate": 0,

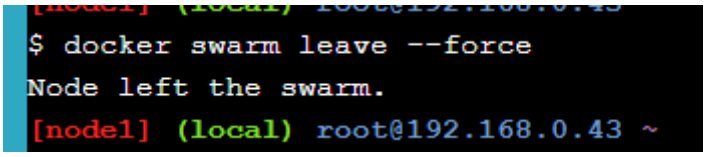

"ForceUpdate": 0,
"Runtime": "container"
},
"Mode": {
"Replicated": {
"Replicas": 2
}
},
"UpdateConfig": {
"Parallelism": 1,
"FailureAction": "pause",
"Monitor": 5000000000,
"MaxFailureRatio": 0,
"Order": "stop-first"
},
"RollbackConfig": {
"Parallelism": 1,
"FailureAction": "pause",
"Monitor": 5000000000,

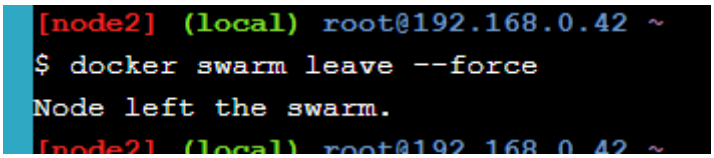

"MaxFailureRatio": 0,
"Order": "stop-first"
},
"EndpointSpec": {
"Mode": "vip"
}
},
"Endpoint": {
"Spec": {
"Mode": "vip"
},
"VirtualIPs": [
{
"NetworkID": "jxqh9rp95syxcrthhw08ry9kg",
"Addr": "10.0.0.4/24"
}
]

Cleanig Up


Cleaning Up
Hopefully you were able to learn a little about how Docker Networking works during this lab. Lets clean up the service we created, the containers we started, and finally disable Swarm mode.
Execute the docker service rm myservice command to remove the service called myservice.
docker service rm myservice
Execute the docker ps command to get a list of running containers.
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
846af8479944 ubuntu "sleep infinity" 17 minutes ago Up 17 minutes
heuristic_boyd
4e0da45b0f16 nginx "nginx -g 'daemon ..." 12 minutes ago Up 12 minutes 443/tcp, 0.0.0.0:8080->80/tc
p web1


[node1] (local) root@192.168.0.43 ~
$ docker service rm myservice
myservice
[node1] (local) root@192.168.0.43 ~
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
21947303655d ubuntu:latest "sleep infinity" 14 minutes ago Up 14 minute
s myservice.1.uh2nint5dr7kzh0whka8ccd5g
7634ffae2592 nginx "nginx -g 'daemon of…" 30 minutes ago Up 30 minute
s 0.0.0.0:8080->80/tcp web1
2d0df0c6d518 ubuntu "sleep infinity" 44 minutes ago Up 44 minute
s angry_swirles
[node1] (local) root@192.168.0.43 ~
$


$ docker swarm leave --force
Node left the swarm.
[node1] (local) root@192.168.0.43 ~


[node2] (local) root@192.168.0.42 ~
$ docker swarm leave --force
Node left the swarm.