

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.

```
$ docker network

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

Usage: docker network COMMAND

Manage networks

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.
(node1) (local) root@192.168.0.43 ~
$ []

#####
# WARNING!!!!                                     #
# This is a sandbox environment. Using personal credentials #
# is HIGHLY! discouraged. Any consequences of doing so are #
# completely the user's responsibilities.                 #
#                                                         #
#####
```

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
```

```
Run 'docker network COMMAND --help' for more information on a command.
(node1) (local) root@192.168.0.43 ~
$ docker network ls

NETWORK ID        NAME                DRIVER              SCOPE
935ac525e1cc     bridge             bridge             local
a7b3d79a417d     host               host               local
2240264e2073     none               null               local
(node1) (local) root@192.168.0.43 ~
$ []

#####
```

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": "3430ad6f20bf486df2e5f6add93c4ff95d81f596b6aea510ad500ff2e57",
    "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": {}
  }
]
```

```
$ docker network inspect bridge

[
  {
    "Name": "bridge",
    "ID": "935ac525e1ccf6defc019252bb4cc98e0e4e174b311647574a5300eb7fd8b58",
    "Created": "2018-12-04T09:50:34.671621583Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16"
        }
      ]
    },
    "Internal": false,
  }
]

#####
# WARNING!!!!                                     #
# This is a sandbox environment. Using personal credentials #
# is HIGHLY! discouraged. Any consequences of doing so are #
# completely the user's responsibilities.                 #
#                                                         #
# The FWD team.                                           #
#####
(node2) (local) root@192.168.0.42 ~
$ []
```

```

    "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 browser.

```

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

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`.

```
docker network ls
```

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

```

WARNING: Bridge nftables is disabled
[node1] (local) root@192.168.0.43 ~
$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
935ac525e1cc        bridge              bridge              local
a7b3d79a417d        host                host                local
2240264e2073        none                null                local
[node1] (local) root@192.168.0.43 ~
$

```

```

#####
#                               WARNING!!!!                               #
# This is a sandbox environment. Using personal credentials                #
#####

```

```
"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:/#
```

```
[root@1 (local)] root@192.168.0.43 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
2d0df0c6d518   ubuntu   "sleep infinity"        9 minutes ago Up 9 minutes
[noel] (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 [89.2 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.github.com (184.239.220.248): 56(84) bytes of data:
64 bytes from 184.239.220.248: icmp_seq=1 ttl=45 time=38.1 ms
64 bytes from 184.239.220.248: icmp_seq=2 ttl=45 time=37.3 ms
64 bytes from 184.239.220.248: icmp_seq=3 ttl=45 time=37.3 ms
64 bytes from 184.239.220.248: icmp_seq=4 ttl=45 time=37.5 ms
64 bytes from 184.239.220.248: icmp_seq=5 ttl=45 time=37.5 ms

--- www.github.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 400ms
rtt min/avg/max/ndev = 37.372/37.641/38.143/0.314 ms
```

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

```
exit
```

```
Setting up libc-bin (2.29-1ubuntu1) ...
Processing triggers for libc-bin (2.29-1ubuntu1) ...
root@2d0df0c6d518:/# ping -c5 www.github.com
PING www.github.com (192.30.253.112): 56(84) bytes of data:
64 bytes from 1b-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=1 ttl=50 time=1.50 ms
64 bytes from 1b-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=2 ttl=50 time=1.51 ms
64 bytes from 1b-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=3 ttl=50 time=1.62 ms
64 bytes from 1b-192-30-253-112-iad.github.com (192.30.253.112): icmp_seq=4 ttl=50 time=1.01 ms
64 bytes from 1b-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 400ms
rtt min/avg/max/ndev = 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 responsibilities. #
# The FWD team. #
```

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
6d827a3ef350: Pull complete
b556b18c7952: Pull complete
8355b0976e24: Pull complete
9abce7a1ef9d: Pull complete
Digest: sha256:52f84ace6ea43f2f58937e5f9f5c62e99ad8876e02b99d1719161ccc587c188
Status: Downloaded newer image for nginx:latest
4e6da4300f169f18b61e1e7b777950bc0ef756402ca8021d55565f162742b3e
```

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

```
docker ps
```

```
[noel] (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
a5a622f73cd8: Pull complete
1ba02017c4b2: Pull complete
23b176c004de: Pull complete
Digest: sha256:5432f60db294b5deb55d078cd4feb410ad88e6fe77500c87d3970eca97f54dba
Status: Downloaded newer image for nginx:latest
7634ffae25922a8a08d00678c031ead0e91cfc8e157173dc8700541dd5fcff57
[noel] (local) root@192.168.0.43 ~
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS
7634ffae2592   nginx    "nginx -g 'daemon off..." 10 seconds ago Up 9 second
a 0.0.0.0:8080->80/tcp
2d0df0c6d518   ubuntu   "sleep infinity"        14 minutes ago Up 14 minut
na          angry_swirles
[noel] (local) root@192.168.0.43 ~
# is HIGHLY! discouraged. Any consequences of doing so are #
# completely the user's responsibilities. #
# #
# The FWD team. #
```

```
curl 127.0.0.1:8080

<!DOCTYPE html>
<html>
<snip>
<head>
<title>Welcome to nginx!</title>
<snip>
<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

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).

```
$ curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
  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 and
working. Further configuration is required.</p>
```

Section #3 - Overlay Networking

Section #3

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)`.

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

Swarm initialized: current node (r5y572arj0o2w0j82zvjk6u) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
      --token SWMTKN-1-69b2x3u2tjdmotboqjxjr2d27f0lbnfhv7j83chl16e5-37ykdpu8vylenefe2439cpgf \
      10.0.0.5:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

In the first terminal copy the entire docker swarm join ... command that is displayed as part of the output from your terminal output. Then, paste the copied command into the second terminal.

    docker swarm join \
      --token SWMTKN-1-69b2x3u2tjdmotboqjxjr2d27f0lbnfhv7j83chl16e5-37ykdpu8vylenefe2439cpgf \
      10.0.0.5:2377
    This node joined a swarm as a worker.

Run a docker node ls to verify that both nodes are part of the Swarm.

    docker node ls

ID                HOSTNAME        STATUS    AVAILABILITY    MANAGER STATUS
ijjqthkdyas4shirjzyngdn48   node2    Ready     Active
r5y572arj0o2w0j82zvjk6u *    node1    Ready     Active           Leader
```

```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
$ docker swarm init --advertise-addr $(hostname -i)
Swarm initialized: current node (m101tyhmouuncfagbitkv8jhs) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2ga3falw04k8rvb78aulnakeuuqwmrqpacqw5e6bokiru627h-1xq14gtqyt970
7020vx8pyppjuw2 192.168.0.43:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

(node1) (local) root@192.168.0.43 ~
$ docker node ls
ID                ENGINE VERSION    HOSTNAME        STATUS    AVAILABILITY    MANAGER STATUS
m101tyhmouuncfagbitkv8jhs *    node1    Ready           Active           Leader
yvgw5xfkx4seze5gyu715tnx      node2    Ready           Active
18.06.1-ce
(node1) (local) root@192.168.0.43 ~

(node2) (local) root@192.168.0.42 ~
$ docker swarm join --token SWMTKN-1-2ga3falw04k8rvb78aulnakeuuqwmrqpacqw5e6bokiru627h-1xq14gtqyt970
20wa8pyppjuw2 192.168.0.43:2377
This node joined a swarm as a worker.
(node2) (local) root@192.168.0.42 ~
$ []
```

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 network called "overnet" by running `docker network create -d overlay overnet`.

```
docker network create -d overlay overnet

wlqnvajmzskn84bqbd1ytuy

Use the docker network ls command to verify the network was created successfully.

    docker network ls

NETWORK ID        NAME                DRIVER            SCOPE
343ba60f7208f    bridge             bridge            local
a4d584310f099    docker_gwbridge    bridge            local
a7449465c379     host               host              local
8hq1n8nat54x     ingress            overlay           swarm
66c34f08cc77     none              null              local
wlqnvajmzskn84bqbd1ytuy    overnet            overlay           swarm

The new "overnet" network is shown on the last line of the output above. Notice how it is associated with the overlay driver and is scoped to the entire Swarm.

NOTE: The other new networks (ingress and docker_gwbridge) were created automatically when the Swarm cluster was created.

Run the same docker network ls command from the second terminal.

    docker network ls

NETWORK ID        NAME                DRIVER            SCOPE
35f18b370e6d     bridge             bridge            local
b7b30433a639     docker_gwbridge    bridge            local
a7449465c379     host               host              local
8hq1n8nat54x     ingress            overlay           swarm
66c34f08cc77     none              null              local
```

```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
TUS
ENGINE VERSION
m101tyhmouuncfagbitkv8jhs *    node1    Ready           Active           Leader
18.06.1-ce
yvgw5xfkx4seze5gyu715tnx      node2    Ready           Active
18.06.1-ce
(node1) (local) root@192.168.0.43 ~
$ docker network create -d overlay overnet
jxq9p9p95ayxcrthhw08ry9kg
(node1) (local) root@192.168.0.43 ~
$ docker network ls
NETWORK ID        NAME                DRIVER            SCOPE
935ac525e1cc      bridge             bridge            local
18d295aa4c10      docker_gwbridge    bridge            local
a7b3d79a417d      host               host              local
hg2vk7crxpat      ingress            overlay           swarm
224024e2073       none              null              local
jxq9p9p95ayxcrthhw08ry9kg    overnet            overlay           swarm
(node1) (local) root@192.168.0.43 ~
$ []

(node2) (local) root@192.168.0.42 ~
$ docker swarm join --token SWMTKN-1-2ga3falw04k8rvb78aulnakeuuqwmrqpacqw5e6bokiru627h-1xq14gtqyt970
20wa8pyppjuw2 192.168.0.43:2377
This node joined a swarm as a worker.
(node2) (local) root@192.168.0.42 ~
$ docker network ls
NETWORK ID        NAME                DRIVER            SCOPE
63ea93b0386b      bridge             bridge            local
32b11bc4acfa      docker_gwbridge    bridge            local
633926684ed7      host               host              local
hg2vk7crxpat      ingress            overlay           swarm
f2e95d4e1735      none              null              local
(node2) (local) root@192.168.0.42 ~
$ []
```

```
docker network inspect overnet

[
  {
    "Name": "overnet",
    "Id": "wlqnvajmzskn84bqbd1ytuy",
    "Created": "0001-01-01T00:00:00Z",
    "Scope": "swarm",
    "Driver": "overlay",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": []
    },
    "Internal": false,
    "Attachable": false,
    "Containers": null,
    "Options": {
      "com.docker.network.driver.overlay.vxlanid_list": "4097"
    },
    "Labels": null
  }
]
```

```
If the commandline doesn't appear in the terminal, make sure popups are enabled or try resizing the browser window.
{
  {
    "Name": "overnet",
    "Id": "jxq9p9p95ayxcrthhw08ry9kg",
    "Created": "2018-12-04T10:38:34.758930125Z",
    "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,
```



```

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

```

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 `myservice` on the `overnet` network 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`.

```
docker service ls
```

```

ID                NAME                MODE                REPLICAS            IMAGE
ov381tv6t2n7     myservice           replicated          2/2                 ubuntu:latest

```

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

Verify that a single task (replica) is running on each of the two nodes in the Swarm by running `docker service ps myservice`.

```
docker service ps myservice
```

```

ID                NAME                IMAGE                NODE                DESIRED STATE        CURRENT STATE        ERROR                PORTS
r1icgg3tute       myservicel           ubuntu:latest        node1               Running               Running about a minute ago
n1oon8zusttv       myservicel2          ubuntu:latest        node1               Running               Running about a minute ago

```

Now that the second node is running a task on the 'overnet' network it will be able to see the 'overnet' network. Let's run `docker network ls` from the second terminal to verify this.

```
docker network ls
```

```

NETWORK ID        NAME                DRIVER              SCOPE
55f18b3f88ed     bridge              bridge              local
b7b38433a639     docker_gwbridge     bridge              local
a7449465c379     host                host                local
8hq1d8nuk54e     ingress             overlay             swarm
06c349b9cc77     none                null                local
wlqva9mzmzk      overnet             overlay             swarm

```

We can also run `docker network inspect overnet` on the second terminal to get more detailed information about the 'overnet' network and obtain the IP address of the task running on the second terminal.

If the commandline doesn't appear in the terminal, make sure popups are enabled or by resizing the browser window.

```
overall progress: 2 out of 2 tasks
```

```
1/2: running
```

```
2/2: running
```

```
Verify: Service converged
```

```
(node1) (local) root@192.168.0.43 ~
```

```
$ docker service ls
```

```

ID                NAME                MODE                REPLICAS            IMAGE
yx4kk6m37uw      myservice           replicated          2/2                 ubuntu:latest
(node1) (local) root@192.168.0.43 ~
$ docker service ps myservice

```

```

ID                NAME                IMAGE                NODE                DESIRED STATE        CURRENT STATE        ERROR                PORTS
uh2nint5dr7k     myservicel         ubuntu:latest        node1               Running               Running 25 seconds ago

```

```
running 23 seconds ago
```

```
(node1) (local) root@192.168.0.43 ~
```

```
$
```

```
(node2) (local) root@192.168.0.42 ~
```

```
$ docker swarm join --token SWMTKN-1-2gz3faw04k8rvb78aulmakeuuqemrgpqcq5a6bokiru627h-1xq14gtyqt970
```

```
20vx8pypjw2 192.168.0.43:2377
```

```
This node joined a swarm as a worker.
```

```
(node2) (local) root@192.168.0.42 ~
```

```
$ docker network ls
```

```

NETWORK ID        NAME                DRIVER              SCOPE
63ea93b0386b     bridge              bridge              local
52b11bc4acfa     docker_gwbridge     bridge              local
633926684ed7     host                host                local
8q2vk7rxspst     ingress             overlay             swarm
f2e95da1e735     none                null                local
jxqh3rp95ayx     overnet             overlay             swarm

```

```
(node2) (local) root@192.168.0.42 ~
```

```
$ docker network inspect overnet
```

```
{
```

```
$ 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": "ulqmv5jmczskn84bqbdllytuy",
    "Created": "2019-04-04T09:35:47.362263887Z",
    "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": "jxqhb9p95ayacrtbhw08xy9kg",
    "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"
        }
      ]
    },

```

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

```

"Internal": false,
"Attachable": false,
"Ingress": false,
"ConfigFrom": {
  "Network": ""
},
"ConfigOnly": false,
"Containers": {
  "21947303655d7ec1f32ffdc01aab294ce33499524f9e980957f74929ad33376": {
    "Name": "myservice.1.uh2nint5dr7krh8whka8cccd5g",
    "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",

```

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

```

    "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": [

```

```

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

```

docker ps						\$ docker ps					
CONTAINER ID	IMAGE	PORTS	STATUS	NAME	CMD	CONTAINER ID	IMAGE	STATUS	COMMAND	CREATED	STATUS
ATD	ubuntu:latest		Up 10 minutes	myservice.1.uh2nint5dr7krh8whka8cccd5g	sleep infinity	21947303655d	ubuntu:latest	Up 5 minutes	sleep infinity	5 minutes ago	Up 5 minutes
d676496d18f7	ubuntu:latest		Up 10 minutes	myservice.2.nlozn82usttv75c9vs3ju7vs	sleep infinity	7634f9ae2592	nginx	Up 21 minutes	nginx -g 'daemon off;'	21 minutes ago	Up 21 minutes
<Snip>						0.0.0.0:8080->80/tcp	web1				
						2d0df0c6d518	ubuntu	Up 35 minutes	sleep infinity	35 minutes ago	Up 35 minutes
							angry_swirls				

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

```

$ docker exec -it 21947303655d /bin/bash
root@192.168.0.43:~#

```

Now, let's ping 10.0.0.3.

```
root@d676496d18f7:/# ping -c 5 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

```
Processing triggers for libc-bin (2.27-3ubuntu1) ...
root@21947303655d:/# ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.228 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.108 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.091 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.223 ms
^C
--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.091/0.162/0.228/0.064 ms
```

```
root@21947303655d:/# cat /etc/resolv.conf
search 51ur3jppi0eupdptvsj42kdvgc.bx.internal.cloudapp.net
nameserver 127.0.0.11
options ndots:0
```

```
PING myservice (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4 (10.0.0.4): icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 10.0.0.4 (10.0.0.4): icmp_seq=2 ttl=64 time=0.050 ms
64 bytes from 10.0.0.4 (10.0.0.4): icmp_seq=3 ttl=64 time=0.046 ms
64 bytes from 10.0.0.4 (10.0.0.4): icmp_seq=4 ttl=64 time=0.072 ms
64 bytes from 10.0.0.4 (10.0.0.4): icmp_seq=5 ttl=64 time=0.076 ms

--- myservice ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 400ms
rtt min/avg/max/mdev = 0.046/0.067/0.092/0.017 ms
```

docker service inspect myservice

```
{
  "ID": "ov38itvdz2n7xy2gnbfqtse",
  "Version": {
    "Index": 19
  },
  "CreatedAt": "2017-04-04T09:35:47.009730798Z",
  "UpdatedAt": "2017-04-04T09:35:47.054730962Z",
  "Spec": {
    "Name": "myservice",
    "TaskTemplate": {
      "ContainerSpec": {
        "Image": "ubuntu:latest@sha256:dd7808d792c9041d0b408122fac6a2dd1f56404f8d1e5629004885e45535",
        "Args": [
          "sleep",
          "infinity"
        ],
        "Labels": {}
      },
      "Endpoint": {
        "Spec": {
          "Mode": "vip"
        }
      }
    }
  }
}
```

```
root@21947303655d:/# docker service inspect myservice
[{"ID": "ov38itvdz2n7xy2gnbfqtse",
  "Version": {
    "Index": 19
  },
  "CreatedAt": "2017-04-04T09:35:47.009730798Z",
  "UpdatedAt": "2017-04-04T09:35:47.054730962Z",
  "Spec": {
    "Name": "myservice",
    "TaskTemplate": {
      "ContainerSpec": {
        "Image": "ubuntu:latest@sha256:dd7808d792c9041d0b408122fac6a2dd1f56404f8d1e5629004885e45535",
        "Args": [
          "sleep",
          "infinity"
        ],
        "Labels": {}
      },
      "Endpoint": {
        "Spec": {
          "Mode": "vip"
        }
      }
    }
  }
}]
```

```
    "Args": [
        "sleep",
        "infinity"
    ],
    "Init": false,
    "StopGracePeriod": 100000000000,
    "DNSConfig": {},
    "Isolation": "default"
},
"Resources": {
    "Limits": {},
    "Reservations": {}
},
"RestartPolicy": {
    "Condition": "any",
```

```
    "RestartPolicy": {
        "Condition": "any",
        "Delay": 50000000000,
        "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,

```

If the commandline doesn't appear in the terminal, make

```

        "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
names	ubuntu	"sleep infinity"	17 minutes ago	Up 17 minutes	
846af9a79544	ubuntu	"sleep infinity"	17 minutes ago	Up 17 minutes	
heuristic_boyd	ubuntu	"sleep infinity"	17 minutes ago	Up 17 minutes	
4e8da45b8f16	nginx	"nginx -g 'daemon ...'"	12 minutes ago	Up 12 minutes	443/tcp, 0.0.0.0:8080->80/tcp
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
21947303655d   ubuntu:latest   "sleep infinity"        14 minutes ago   Up 14 minute
#
7634ffae2552   nginx      "nginx -g 'daemon of.'" 30 minutes ago   Up 30 minute
#
2d8df0c6d518   ubuntu     "sleep infinity"        44 minutes ago   Up 44 minute
#
angry_wirles
[node1] (local) root@192.168.0.43 ~
$

```

```

[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.
[node2] (local) root@192.168.0.42 ~

```