RX-M

Docker In Practice

Lab – Swarm

Docker Engine 1.12 includes swarm mode for natively managing a cluster of Docker Engines. This cluster is called a swarm. The Docker CLI can be used to create a swarm, deploy application services to a swarm, and manage swarm behavior.

In this lab we will convert our lab system into a single node swarm and explore swarm functionality.

1. Initialize a Swarm

To convert your lab system into a swarm node you will need to tell it which IP address to advertise to the world. Display the host IPs:

```
user@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default 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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d3:48:0f brd ff:ff:ff:ff:ff:ff
    inet 192.168.131.203/24 brd 192.168.131.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fed3:480f/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:ab:2b:a8:97 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:abff:fe2b:a897/64 scope link
       valid_lft forever preferred_lft forever
user@ubuntu:~$
```

Now we can initialize a new swarm cluster or join an existing cluster (a single Docker Engine can only be in one cluster at a time.) Let's initialize a new cluster using our ens33 address:

```
user@ubuntu:~$ docker swarm init --advertise-addr=192.168.131.203
Swarm initialized: current node (eq59zkamgylgjezs1gyopvkvj) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-1u482mfjnikwnncbjzekk69sfm8xxyqihtmm2gt75qzfo4wlpx-axk1omy0zc6pn6u6loitdrulu \
    192.168.131.203:2377

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

user@ubuntu:~$
```

That's it, we now have a swarm running. Run the `docker info` subcommand to gather cluster information associated with your node:

```
user@ubuntu:~$ docker info
Containers: 8
 Running: 0
 Paused: 0
 Stopped: 8
Images: 31
Server Version: 1.12.1
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 58
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: host bridge overlay null
Swarm: active
 NodeID: eq59zkamgylgjezs1gyopvkvj
 Is Manager: true
 ClusterID: 37f12zkc772a1i7v9vckxhood
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
 Node Address: 192.168.131.203
Runtimes: runc
Default Runtime: runc
Security Options: apparmor seccomp
Kernel Version: 4.4.0-34-generic
Operating System: Ubuntu 16.04.1 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 1.937 GiB
Name: ubuntu
ID: 222O:USAP:JG2M:YRNA:G26Q:XQKM:XQB3:MORI:RHXJ:X4TO:H44T:QSXL
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Username: randyabernethy
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
Insecure Registries:
 127.0.0.0/8
user@ubuntu:~$
```

In the listing above you will see the top level key 'Swarm' with the new value of 'Active'. The swarm cluster has an ID and each node in the Swarm has a Node ID. Node's are either managers or a workers. Managers perform scheduling and other cluster orchestration jobs. Workers execute tasks (containers in cluster speak). Managers are also worker by default.

Nodes in a Swarm can be inspected and configured using the `docker node` subcommand. List the nodes in the cluster:

```
user@ubuntu:~$ docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
eq59zkamgylgjezs1gyopvkvj *  ubuntu    Ready   Active        Leader
```

Inspect the node to gather more node information:

```
user@ubuntu:~$ docker node inspect eq59zkamgylgjezs1gyopvkvj
[
    {
        "ID": "eq59zkamgylgjezs1gyopvkvj",
        "Version": {
            "Index": 11
        },
        "CreatedAt": "2016-08-24T06:54:14.866899365Z",
        "UpdatedAt": "2016-08-24T06:54:14.89459048Z",
        "Spec": {
            "Role": "manager",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "ubuntu",
            "Platform": {
                "Architecture": "x86_64",
                "OS": "linux"
            },
            "Resources": {
                "NanoCPUs": 2000000000,
                "MemoryBytes": 2079354880
            },
            "Engine": {
                "EngineVersion": "1.12.1",
                "Plugins": [
                    {
                        "Type": "Network",
                        "Name": "bridge"
                    },
                    {
                        "Type": "Network",
                        "Name": "host"
                    },
                    {
                        "Type": "Network",
                        "Name": "null"
                    },
                    {
                        "Type": "Network",
                        "Name": "overlay"
                    },
                    {
                        "Type": "Volume",
                        "Name": "local"
                    }
                ]
            }
        },
        "Status": {
            "State": "ready"
        },
        "ManagerStatus": {
            "Leader": true,
            "Reachability": "reachable",
            "Addr": "192.168.131.203:2377"
        }
    }
]
user@ubuntu:~$
```

Just like most things in Docker, Nodes are described by metadata which can be displayed with the inspect command. You can refer to a node by name (hostname) or Node Id.

Note that Docker Swarm knows how much memory and CPU the Node has available. This will be useful when deciding where to run containers in the swarm. Also notice that Swarm has automatically enabled the overlay multi-host networking driver. This will allow networks to be created that span the Swarm, so that containers on the same network do not need to run on the same host.

2. Run a service

Swarm clusters do not run containers directly, they run services. Services are implemented by tasks, and task are in fact implemented by containers. The semantics are important. A service is the implementation of a piece of application functionality. A given service is usually implemented by a Docker image but we run multiple copies of that image to gain scale and reliability in a microservices deployment. So in Swarm the service endpoint abstracts away the notion of connecting to a particular container, clients instead connect to the service. Docker then arranges service connections to be routed to one of the tasks (containers) implementing the service.

If one container (task) dies we just start another to take its place and the service perseveres. Docker Swarm implements services via load balanced Virtual IPs or Round Robin DNS.

Try running a simple web service:

```
user@ubuntu:~$ docker service create --replicas=2 --name website -p 80 nginx
5yrlrimm9fwu8clxq9tdln2jx
```

List the services running using docker service ls:

```
user@ubuntu:~$ docker service ls
ID            NAME     REPLICAS  IMAGE  COMMAND
5yrlrimm9fwu  website  2/2       nginx
```

Our service has two out of two replicas running.

Run `docker ps` to see the containers Docker has launched to support your new service:

```
user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
cf3ce19366de        nginx:latest        "nginx -g 'daemon off"   8 minutes ago       Up 8 minutes        80/tcp, 443/tcp
website.2.3qpuoylq4vehbbo6jnfcr0agh
483be9275c61        nginx:latest        "nginx -g 'daemon off"   8 minutes ago       Up 8 minutes        80/tcp, 443/tcp
website.1.5qs5ty2pyj3zg016fncq25yu8
```

Inspect your new service:

```
user@ubuntu:~$ docker service inspect website
[
    {
        "ID": "5yrlrimm9fwu8clxq9tdln2jx",
        "Version": {
            "Index": 14
        },
        "CreatedAt": "2016-08-24T07:14:50.973689275Z",
        "UpdatedAt": "2016-08-24T07:14:50.975210467Z",
        "Spec": {
            "Name": "website",
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "nginx"
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                    "Condition": "any",
                    "MaxAttempts": 0
                },
                "Placement": {}
            },
            "Mode": {
                "Replicated": {
                    "Replicas": 2
                }
            },
            "UpdateConfig": {
                "Parallelism": 1,
                "FailureAction": "pause"
            },
            "EndpointSpec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80
                    }
                ]
            }
        },
        "Endpoint": {
            "Spec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80
                    }
                ]
            },
            "Ports": [
                {
                    "Protocol": "tcp",
                    "TargetPort": 80,
                    "PublishedPort": 30000
                }
            ],
            "VirtualIPs": [
                {
                    "NetworkID": "bjsxbeqzqh13be503c6zkgxy0",
                    "Addr": "10.255.0.5/16"
                }
            ]
        },
        "UpdateStatus": {
            "StartedAt": "0001-01-01T00:00:00Z",
            "CompletedAt": "0001-01-01T00:00:00Z"
        }
    }
]
user@ubuntu:~$
```

You can get a more compact view with the --pretty switch:

```
user@ubuntu:~$ docker service inspect --pretty website
ID:        5yrlrimm9fwu8clxq9tdln2jx
Name:        website
Mode:        Replicated
 Replicas:    2
Placement:
UpdateConfig:
 Parallelism:    1
 On failure:    pause
ContainerSpec:
 Image:        nginx
Resources:
Ports:
 Protocol = tcp
 TargetPort = 80
 PublishedPort = 30000
```

The inspect metadata shows that our service was given a virtual IP address (10.255.0.5 in the example) on a network with the ID bjsxbeqzqh13be503c6zkgxy0. List the networks available:

```
user@ubuntu:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
1b1a61f6359a        bridge              bridge              local
e43fc4316a4e        docker_gwbridge     bridge              local
3e6e431b74a8        host                host                local
bjsxbeqzqh13        ingress             overlay             swarm
1f18223bf787        none                null                local
```

Swarm has created an overlay network (multihost VXLAN) for our swarm and it is the network our virtual IP is on. Let's inspect the network:

```
user@ubuntu:~$ docker network inspect bjsxbeqzqh13
[
    {
        "Name": "ingress",
        "Id": "bjsxbeqzqh13be503c6zkgxy0",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.255.0.0/16",
                    "Gateway": "10.255.0.1"
                }
            ]
        },
        "Internal": false,
        "Containers": {
            "483be9275c615d640ef3b261f9f7971f31d427d2f1321e5ef312f08afab3ef4a": {
                "Name": "website.1.5qs5ty2pyj3zg016fncq25yu8",
                "EndpointID": "7720a3b65bc66f1a33609de0b055ddf47d7138f0c4bfdc1f57f93b1109e0c2ad",
                "MacAddress": "02:42:0a:ff:00:06",
                "IPv4Address": "10.255.0.6/16",
                "IPv6Address": ""
            },
            "cf3ce19366de6f411863a9831e2eee1136f82873eb272732a15d64dfc8a37206": {
                "Name": "website.2.3qpuoylq4vehbbo6jnfcr0agh",
                "EndpointID": "c838736a73f993a1f102334ff4cb72da317747e0d04e18b5559fa0543df08142",
                "MacAddress": "02:42:0a:ff:00:07",
                "IPv4Address": "10.255.0.7/16",
                "IPv6Address": ""
            },
            "ingress-sbox": {
                "Name": "ingress-endpoint",
                "EndpointID": "c4d7dd8ba6985fb9121249e9a0c3a4fe579376af5fd00938dd2cffa75d66f16c",
                "MacAddress": "02:42:0a:ff:00:04",
                "IPv4Address": "10.255.0.4/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "256"
        },
        "Labels": {}
    }
]
user@ubuntu:~$
```

Note that the network's name is ingress. This network is designed to allow traffic into the services of the swarm. List the routes known to your lab host:

```
user@ubuntu:~$ ip route
default via 192.168.131.2 dev ens33
172.17.0.0/16 dev docker0  proto kernel  scope link  src 172.17.0.1 linkdown
172.18.0.0/16 dev docker_gwbridge  proto kernel  scope link  src 172.18.0.1
192.168.131.0/24 dev ens33  proto kernel  scope link  src 192.168.131.203
```

Our host has no knowledge of the swarm network. It is a software defined network used to expose services within the swarm. Open a shell inside one of your service's task containers and display the network interfaces:

```
user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
cf3ce19366de        nginx:latest        "nginx -g 'daemon off"   About an hour ago   Up About an hour    80/tcp, 443/tcp
website.2.3qpuoylq4vehbbo6jnfcr0agh
483be9275c61        nginx:latest        "nginx -g 'daemon off"   About an hour ago   Up About an hour    80/tcp, 443/tcp
website.1.5qs5ty2pyj3zg016fncq25yu8

user@ubuntu:~$ docker exec -it cf3ce19366de bash
root@cf3ce19366de:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default 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
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
115: eth0@if116: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default
    link/ether 02:42:0a:ff:00:07 brd ff:ff:ff:ff:ff:ff
    inet 10.255.0.7/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet 10.255.0.5/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:aff:feff:7/64 scope link
       valid_lft forever preferred_lft forever
119: eth1@if120: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.4/16 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe12:4/64 scope link
       valid_lft forever preferred_lft forever

root@cf3ce19366de:/# exit
exit
user@ubuntu:~$
```

Our task containers have a loopback interface, a 172 address interface and an interface on the ingress network. Traffic to/from the host transits the 172 interface and the ingress interface is used by the swarm load balancer.

The 172 interface of our container is connected to the gateway network on the host that it is running on. Display the subnet for the docker_gwbridge:

```
user@ubuntu:~$ docker network inspect docker_gwbridge | grep -i subnet
                    "Subnet": "172.18.0.0/16",
```

So our host can access the container via 172.18.0.4/16 and other services can access our container via the ingress network on 10.255.0.7,

Finally let's dispay the nodes our serivce tasks are running on:

```
user@ubuntu:~$ docker service ps website
ID                         NAME       IMAGE  NODE    DESIRED STATE  CURRENT STATE        ERROR
5qs5ty2pyj3zg016fncq25yu8  website.1  nginx  ubuntu  Running        Running 5 hours ago
3qpuoylq4vehbbo6jnfcr0agh  website.2  nginx  ubuntu  Running        Running 5 hours ago
user@ubuntu:~$
```

3. Use your service

Our service is assigned a port on every node in the cluster, even nodes not running a task container for the service.

The published port for our service was 30,000:

```
user@ubuntu:~$ docker service inspect website | grep -i PublishedPort
                    "PublishedPort": 30000
```

Try `curl` 'ing your lab host IP on port 30000.

```
user@ubuntu:~$ curl 192.168.131.203:30000
<!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>

<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>
```

This works because Docker has setup a DNAT rule to forward traffic coming in on port 30000 to our service. Display any iptables NAT rules associated with port 30000:

```
user@ubuntu:~$ sudo iptables -nvL -t nat | grep -i 30000
    1    60 DNAT     tcp  --  *      *     0.0.0.0/0     0.0.0.0/0       tcp dpt:30000 to:172.18.0.2:30000
user@ubuntu:~$
```

Note that the DNAT forwards traffic to 172.18.0.2. This is a Docker service proxy that directs traffic to the one or several task containers implementing the service. Because our lab host has a route to this Linux Bridge network we can curl the proxy directly:

```
user@ubuntu:~$ curl 172.18.0.2:30000
<!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>

<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>
user@ubuntu:~$
```

If you display the metadata for the gateway network you will see three devices, our two nginx containers and the sbox proxy:

```
user@ubuntu:~$ docker network inspect docker_gwbridge
[
    {
        "Name": "docker_gwbridge",
        "Id": "e43fc4316a4ec64bb522c452da0f74fca76b335a2c75195ab21ce96053e9a928",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1/16"
                }
            ]
        },
        "Internal": false,
        "Containers": {
            "483be9275c615d640ef3b261f9f7971f31d427d2f1321e5ef312f08afab3ef4a": {
                "Name": "gateway_483be9275c61",
                "EndpointID": "07867ec75793e08c5915a15189f9e6d27a8b5f098ff05a8a49985ef2e755e1e2",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "cf3ce19366de6f411863a9831e2eee1136f82873eb272732a15d64dfc8a37206": {
                "Name": "gateway_cf3ce19366de",
                "EndpointID": "e20be375e16eaff95d87fb98a14e98225136a8abd39379da04563a8c2252ea7d",
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
            },
            "ingress-sbox": {
                "Name": "gateway_ingress-sbox",
                "EndpointID": "333ffc2b0e7cbf1d4b0bd5bd0ec0738dfc81e7de72804af5dbb4d437ae7808c7",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.enable_icc": "false",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.name": "docker_gwbridge"
        },
        "Labels": {}
    }
]
user@ubuntu:~$
```

We can of course also `curl` the two containers directly:

```
user@ubuntu:~$ curl -s 172.18.0.3:80 | head -1
<!DOCTYPE html>
user@ubuntu:~$ curl -s 172.18.0.4:80 | head -1
<!DOCTYPE html>
```

To see the proxy load balancing between our tasks we can tail the log of one (or both) of the nginx containers. In a separate terminal tail the log for one of your nginx containers:

```
user@ubuntu:~$ docker logs --tail 0 -f cf3ce19366de

10.255.0.4 - - [24/Aug/2016:10:53:42 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.47.0" "-"
10.255.0.4 - - [24/Aug/2016:10:53:55 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.47.0" "-"
```

Now curl the service proxy 4 times:

```
user@ubuntu:~$ curl -s 172.18.0.2:30000 | head -1
<!DOCTYPE html>
user@ubuntu:~$ curl -s 172.18.0.2:30000 | head -1
<!DOCTYPE html>
user@ubuntu:~$ curl -s 172.18.0.2:30000 | head -1
<!DOCTYPE html>
user@ubuntu:~$ curl -s 172.18.0.2:30000 | head -1
<!DOCTYPE html>
user@ubuntu:~$
```

Docker sends every other request to the container we are tailing.

4. Scale the service

Imagine we need to increase the number of tasks supporting our service. We can easily scale our service up and down using the swarm service scale command. Try it:

```
user@ubuntu:~$ docker service ps website
ID                          NAME       IMAGE  NODE    DESIRED STATE  CURRENT STATE        ERROR
```

```
5qs5ty2pyj3zg016fncq25yu8  website.1  nginx  ubuntu  Running        Running 5 hours ago
3qpuoylq4vehbbo6jnfcr0agh  website.2  nginx  ubuntu  Running        Running 5 hours ago

user@ubuntu:~$ docker service scale website=3
website scaled to 3

user@ubuntu:~$ docker service ps website
ID                         NAME       IMAGE  NODE    DESIRED STATE  CURRENT STATE          ERROR
5qs5ty2pyj3zg016fncq25yu8  website.1  nginx  ubuntu  Running        Running 5 hours ago
3qpuoylq4vehbbo6jnfcr0agh  website.2  nginx  ubuntu  Running        Running 5 hours ago
8hr7x2xzhrcy9omepgfv23u49  website.3  nginx  ubuntu  Running        Running 1 seconds ago

user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
1b338e68f471        nginx:latest        "nginx -g 'daemon off"   10 seconds ago      Up 9 seconds        80/tcp, 443/tcp
website.3.8hr7x2xzhrcy9omepgfv23u49
cf3ce19366de        nginx:latest        "nginx -g 'daemon off"   5 hours ago         Up 5 hours          80/tcp, 443/tcp
website.2.3qpuoylq4vehbbo6jnfcr0agh
483be9275c61        nginx:latest        "nginx -g 'daemon off"   5 hours ago         Up 5 hours          80/tcp, 443/tcp
website.1.5qs5ty2pyj3zg016fncq25yu8
user@ubuntu:~$
```

The scale command automatically adds new task containers to the service load balancer.

5. Tear down the service

Shutting down a service is easy in Swarm using the `docker service rm` subcommand. Remove your website service:

```
user@ubuntu:~$ docker service rm website
website

user@ubuntu:~$ docker service ls
ID  NAME  REPLICAS  IMAGE  COMMAND
user@ubuntu:~$
```

6. Tear down the cluster

You can demote a swarm master to a worker and you can remove a worker node from a cluster using the `docker node` subcommand. Our lab system is the only master however so we can not demote it. To leave the swarm (and terminate the swarm) we need to use the `docker swarm leave` subcommand. Leave and terminate your Swarm:

```
user@ubuntu:~$ docker swarm leave
Error response from daemon: You are attempting to leave the swarm on a node that is participating as a manager. Removing the
last manager erases all current state of the swarm. Use `--force` to ignore this message.

user@ubuntu:~$ docker swarm leave --force
Node left the swarm.

user@ubuntu:~$ docker service ls
Error response from daemon: This node is not a swarm manager. Use "docker swarm init" or "docker swarm join" to connect this
node to swarm and try again.
user@ubuntu:~$
```

[OPTIONAL] Run a networked service

In typical use we will want to run services on multi-host networks. When swarm is enabled we have access to the Docker multi-host overlay driver by default. In this optional step we'll create a swarm rerun our nginx service on a multi-host network and then access it by Virtual IP over the network.

First create a swarm:

```
user@ubuntu:~$ docker swarm init --advertise-addr=192.168.131.203
Swarm initialized: current node (0a9psetfehtone1s7qbsshke2) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-1aiv3ngmqrie7pmhi6pj9r9l1xcyfnrj1moaxi1ycdon7wdniy-0nzfahk2kjk7i6b5snb2x28gn \
    192.168.131.203:2377

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

Now create an overlay network for our webservice and its clients:

```
user@ubuntu:~$ docker network create --driver overlay --subnet 10.0.9.0/24 --opt encrypted webnet
6uu971pse5wrrxk4e4v68455m
```

Now create the website service on the subnet network:

```
user@ubuntu:~$ docker service create --replicas=2 --name website --network webnet -p 80 nginx
04zgjspd0rnwri4ku5a98dhcb
```

To test our service we can create a client service on the same network using the cirros image. We'll exec into the service container to experiment:

```
user@ubuntu:~$ docker service create --name client --network webnet cirros
20jgpzjj7ud0csw4wrddqxkb3

user@ubuntu:~$ docker service ls
ID            NAME     REPLICAS  IMAGE   COMMAND
04zgjspd0rnw  website  2/2       nginx
20jgpzjj7ud0  client   0/1       cirros

user@ubuntu:~$ docker service ls
ID            NAME     REPLICAS  IMAGE   COMMAND
04zgjspd0rnw  website  2/2       nginx
20jgpzjj7ud0  client   1/1       cirros

user@ubuntu:~$ docker service ps website
ID                         NAME       IMAGE  NODE    DESIRED STATE  CURRENT STATE          ERROR
1kddy0zpcbej0747ul1muby7d  website.1  nginx  ubuntu  Running        Running 4 minutes ago
8mwtjjttnqv25mqau17vxz8sf  website.2  nginx  ubuntu  Running        Running 4 minutes ago

user@ubuntu:~$ docker service ps client
ID                         NAME      IMAGE   NODE    DESIRED STATE  CURRENT STATE           ERROR
c0oj11w51udcta40boijb2skd  client.1  cirros  ubuntu  Running        Running 25 seconds ago
```

Now lookup the container Id of your Cirros service container and `docker exec` into it:

```
user@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
NAMES
fbbbd162db94        cirros:latest       "/sbin/init"             About a minute ago  Up About a minute
client.1.c0oj11w51udcta40boijb2skd
cc2740ff2cde        nginx:latest        "nginx -g 'daemon off"   5 minutes ago       Up 5 minutes        80/tcp, 443/tcp
website.2.8mwtjjttnqv25mqau17vxz8sf
5778636029c2        nginx:latest        "nginx -g 'daemon off"   5 minutes ago       Up 5 minutes        80/tcp, 443/tcp
website.1.1kddy0zpcbej0747ul1muby7d

user@ubuntu:~$ docker exec -it fbbbd162db94 sh

/ #
```

Now we can lookup the website service by name:

```
/ # nslookup website
Server:    127.0.0.11
Address 1: 127.0.0.11

Name:      website
Address 1: 10.0.9.2
```

Try `curl` 'ing the service by name and IP:

```
/ # curl website
<!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>

<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>

/ # curl 10.0.9.2

<!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>

<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>
```

Remove all services, networks and exit the Swarm when you are finished exploring.

Congratulations, you have completed the Docker Swarm lab!