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. First lets clean up.

```
user@ubuntu:~$ docker container stop $(docker container ls -qa)
...
```

Display the host IPs:

```
user@ubuntu:~$ ip a show

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:fe:f9:fb brd ff:ff:ff:ff:ff:ff
    inet 172.16.151.148/24 brd 172.16.151.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fefe:f9fb/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 02:42:3a:cf:df:f7 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:3aff:fecf:dff7/64 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=172.16.151.148

Swarm initialized: current node (buepgr23z3x9wkmf7jy3fobzi) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-3k60bi74flc36ju1s2i7616la87ibt5ty4okjo1n1crc6jzn9c-
a2qjqstwwtu3x0wgwd5hjeawq \
    172.16.151.148:2377

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 `info` subcommand to gather cluster information associated with your node:

```
user@ubuntu:~$ docker info

Containers: 2
 Running: 0
 Paused: 0
 Stopped: 2
Images: 26
Server Version: 17.06.0-ce
Storage Driver: aufs
 Root Dir: /var/lib/docker/aufs
 Backing Filesystem: extfs
 Dirs: 60
 Dirperm1 Supported: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: active
 NodeID: buepgr23z3x9wkmf7jy3fobzi
 Is Manager: true
 ClusterID: kfmib0u7mcai1drjmtjuh7zle
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 3
 Dispatcher:
  Heartbeat Period: 5 seconds
 CA Configuration:
  Expiry Duration: 3 months
```

```
 Node Address: 172.16.151.148
 Manager Addresses:
  172.16.151.148:2377
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 977c511eda0925a723debdc94d09459af49d082a
runc version: a01dafd48bc1c7cc12bdb01206f9fea7dd6feb70
init version: 949e6fa
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 4.4.0-66-generic
Operating System: Ubuntu 16.04.1 LTS
OSType: linux
Architecture: x86_64
CPUs: 2
Total Memory: 3.842 GiB
Name: ubuntu
ID: P2KT:SFQG:SAL2:APPW:E63N:BN2P:NG7U:LVBN:NCRL:ZDPA:PM2G:7SRX
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
WARNING: No swap limit support
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false

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 `node` subcommand. List the nodes in the cluster:

```
user@ubuntu:~$ docker node ls

ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
buepgr23z3x9wkmf7jy3fobzi *  ubuntu    Ready   Active        Leader

user@ubuntu:~$
```

Inspect the node to gather more node information:

```
user@ubuntu:~$ docker node inspect self

[
    {
```

```
        "ID": "buepgr23z3x9wkmf7jy3fobzi",
        "Version": {
            "Index": 9
        },
        "CreatedAt": "2017-03-08T07:04:47.785042296Z",
        "UpdatedAt": "2017-03-08T07:04:48.388981317Z",
        "Spec": {
            "Role": "manager",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "ubuntu",
            "Platform": {
                "Architecture": "x86_64",
                "OS": "linux"
            },
            "Resources": {
                "NanoCPUs": 2000000000,
                "MemoryBytes": 4125437952
            },
            "Engine": {
                "EngineVersion": "17.06.0-ce",
                "Plugins": [
                    {
                        "Type": "Network",
                        "Name": "bridge"
                    },
                    {
                        "Type": "Network",
                        "Name": "host"
                    },
                    {
                        "Type": "Network",
                        "Name": "macvlan"
                    },
                    {
                        "Type": "Network",
                        "Name": "null"
                    },
                    {
                        "Type": "Network",
                        "Name": "overlay"
                    },
                    {
                        "Type": "Volume",
                        "Name": "local"
                    }
                ]
            }
        },
        "Status": {
            "State": "ready",
            "Addr": "127.0.0.1"
        },
        "ManagerStatus": {
            "Leader": true,
            "Reachability": "reachable",
            "Addr": "172.16.151.148:2377"
```

```
        }
    }
]

user@ubuntu:~$
```

Notice we used *self* (local alias), if we need to specify a node (especially remote ones) use the ID.

```
user@ubuntu:~$ docker node inspect self | jq -r .[].ID

buepgr23z3x9wkmf7jy3fobzi

user@ubuntu:~$ docker node inspect buepgr23z3x9wkmf7jy3fobzi | head -3

[
    {
        "ID": "buepgr23z3x9wkmf7jy3fobzi",

user@ubuntu:~$
```

Just like most things in Docker, Nodes are described by metadata which can be displayed with the `inspect` subcommand. 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:1.11

ivx2dm15wfbvfkee10ww0v02c

user@ubuntu:~$
```

List the services running using `docker service ls`:

```
user@ubuntu:~$ docker service ls

ID            NAME     MODE        REPLICAS  IMAGE
ivx2dm15wfbv  website  replicated  2/2       nginx:1.11

user@ubuntu:~$
```

Our service has two out of two replicas running.

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

```
user@ubuntu:~$ docker container ls

CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS              PORTS
NAMES
65cdd00293c8
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   34 seconds ago      Up 32 seconds       80/tcp, 443/tcp
website.1.n6r23600qp4s3xb0uelaxnc1x
0c76de7d0183
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   34 seconds ago      Up 33 seconds       80/tcp, 443/tcp
website.2.pv64kr6ulh1m2jplqodm7zcfj

user@ubuntu:~$
```

Inspect your new service:

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

[
    {
        "ID": "ivx2dm15wfbvfkee10ww0v02c",
        "Version": {
            "Index": 12
        },
        "CreatedAt": "2017-03-08T07:09:13.33299973Z",
        "UpdatedAt": "2017-03-08T07:09:13.333535516Z",
        "Spec": {
            "Name": "website",
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image":
"nginx:1.11@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe3
35",
                    "DNSConfig": {}
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
```

```
                "RestartPolicy": {
                    "Condition": "any",
                    "MaxAttempts": 0
                },
                "Placement": {},
                "ForceUpdate": 0
            },
            "Mode": {
                "Replicated": {
                    "Replicas": 2
                }
            },
            "UpdateConfig": {
                "Parallelism": 1,
                "FailureAction": "pause",
                "MaxFailureRatio": 0
            },
            "EndpointSpec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            }
        },
        "Endpoint": {
            "Spec": {
                "Mode": "vip",
                "Ports": [
                    {
                        "Protocol": "tcp",
                        "TargetPort": 80,
                        "PublishMode": "ingress"
                    }
                ]
            },
            "Ports": [
                {
                    "Protocol": "tcp",
                    "TargetPort": 80,
                    "PublishedPort": 30000,
                    "PublishMode": "ingress"
                }
            ],
            "VirtualIPs": [
                {
                    "NetworkID": "lpvkdljar2difu0dx32odzd65",
                    "Addr": "10.255.0.4/16"
                }
            ]
        },
        "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 website --pretty

ID:             ivx2dm15wfbvfkee10ww0v02c
Name:           website
Service Mode:   Replicated
 Replicas:      2
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Max failure ratio: 0
ContainerSpec:
 Image:
nginx:1.11@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe33
5
Resources:
Endpoint Mode:  vip
Ports:
 PublishedPort 30000
  Protocol = tcp
  TargetPort = 80

user@ubuntu:~$
```

The inspect metadata shows that our service was given a virtual IP address (10.255.0.4 in the example) on a network with the ID lpvkdljar2difu0dx32odzd65. List the networks available:

```
user@ubuntu:~$ docker network ls

NETWORK ID          NAME                DRIVER              SCOPE
e943915b1cb2        bridge              bridge              local
3efa59a674bd        docker_gwbridge     bridge              local
f077f1a35d46        host                host                local
lpvkdljar2di        ingress             overlay             swarm
eeaef4dce20c        none                null                local

user@ubuntu:~$
```

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

```
user@ubuntu:~$ docker network inspect ingress

[
```

```
    {
        "Name": "ingress",
        "Id": "lpvkdljar2difu0dx32odzd65",
        "Created": "2017-03-07T23:04:48.394276296-08:00",
        "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,
        "Attachable": false,
        "Containers": {
            "0c76de7d0183152784f535495d124c7462d5acaf06eb8722913eb75057082225": {
                "Name": "website.2.pv64kr6ulh1m2jplqodm7zcfj",
                "EndpointID":
"ff60a5f7018d78bae53461ad8e06df27699966fb32ea69cda2ac4cdb1a63fd08",
                "MacAddress": "02:42:0a:ff:00:06",
                "IPv4Address": "10.255.0.6/16",
                "IPv6Address": ""
            },
            "65cdd00293c8712096e0e50e62dce63d088f696bb2a9bf23d4e60ebd88204e2a": {
                "Name": "website.1.n6r23600qp4s3xb0uelaxnc1x",
                "EndpointID":
"6514ab61fc2409c6daf3a7f21192d2988b79d4b8fccdb9019d0c9c82f25df24e",
                "MacAddress": "02:42:0a:ff:00:05",
                "IPv4Address": "10.255.0.5/16",
                "IPv6Address": ""
            },
            "ingress-sbox": {
                "Name": "ingress-endpoint",
                "EndpointID":
"fdb6f4e3aef14ff0237b53736f2700fd92724aeb8e94a50219598dc302f338a0",
                "MacAddress": "02:42:0a:ff:00:03",
                "IPv4Address": "10.255.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid_list": "4096"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "ubuntu-1d403af45dc2",
                "IP": "172.16.151.148"
            }
        ]
    }
]
```

```
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 172.16.151.2 dev ens33
172.16.151.0/24 dev ens33  proto kernel  scope link  src 172.16.151.148
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

user@ubuntu:~$
```

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 container ls

CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS              PORTS
NAMES
65cdd00293c8
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   3 minutes ago       Up 3 minutes        80/tcp, 443/tcp
website.1.n6r23600qp4s3xb0uelaxnc1x
0c76de7d0183
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   3 minutes ago       Up 3 minutes        80/tcp, 443/tcp
website.2.pv64kr6ulh1m2jplqodm7zcfj

user@ubuntu:~$ docker container exec -it 65cdd00293c8 bash

root@65cdd00293c8:/# ip a show

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
109: eth0@if110: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state
UP group default
    link/ether 02:42:0a:ff:00:05 brd ff:ff:ff:ff:ff:ff
    inet 10.255.0.5/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet 10.255.0.4/32 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:aff:feff:5/64 scope link
       valid_lft forever preferred_lft forever
111: eth1@if112: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default
```

```
     link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff
     inet 172.18.0.3/16 scope global eth1
        valid_lft forever preferred_lft forever
     inet6 fe80::42:acff:fe12:3/64 scope link
        valid_lft forever preferred_lft forever

root@65cdd00293c8:/# ip route

default via 172.18.0.1 dev eth1
10.255.0.0/16 dev eth0  proto kernel  scope link  src 10.255.0.5
172.18.0.0/16 dev eth1  proto kernel  scope link  src 172.18.0.3

root@65cdd00293c8:/# 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 (eth0@if94 above) 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 -f \
'{{range .IPAM.Config}}{{.Subnet}}{{end}}'

172.18.0.0/16

user@ubuntu:~$
```

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 10.255.0.6.

Finally, let's display the nodes that our service tasks are running on:

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

ID            NAME       IMAGE         NODE    DESIRED STATE  CURRENT STATE
ERROR  PORTS
n6r23600qp4s  website.1  nginx:1.11    ubuntu  Running        Running 5 minutes
ago
pv64kr6ulh1m  website.2  nginx:1.11    ubuntu  Running        Running 5 minutes
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 -f \
'{{range .Endpoint.Ports}}{{.PublishedPort}}{{end}}'

30000

user@ubuntu:~$
```

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

```
user@ubuntu:~$ ip a show ens33

2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 00:0c:29:fe:f9:fb brd ff:ff:ff:ff:ff:ff
    inet 172.16.151.148/24 brd 172.16.151.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fefe:f9fb/64 scope link
       valid_lft forever preferred_lft forever

user@ubuntu:~$ curl 172.16.151.148:30000

<!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:~$
```

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
```

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>
```

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": "3efa59a674bdb415462574d0bedb0773633800a2ccc68aa71c724557878b1295",
        "Created": "2017-03-07T23:04:48.660350572-08:00",
        "Scope": "local",
```

```
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "0c76de7d0183152784f535495d124c7462d5acaf06eb8722913eb75057082225": {
                "Name": "gateway_0c76de7d0183",
                "EndpointID":
"cee8c9266022e23b8aa29348ee1901713487f6bad57a29100d1202bd9aa0e348",
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
            },
            "65cdd00293c8712096e0e50e62dce63d088f696bb2a9bf23d4e60ebd88204e2a": {
                "Name": "gateway_65cdd00293c8",
                "EndpointID":
"dcf76c825526042d35bd68bf5e09ac3fa690320775822233d0cabbf6730c55a9",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            },
            "ingress-sbox": {
                "Name": "gateway_ingress-sbox",
                "EndpointID":
"8be832b69e8a599e0199a0183b43d582aa7a414ca2c0aa008cdf30981147c971",
                "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 container logs --tail 0 -f 65cdd00293c8

10.255.0.3 - - [08/Mar/2017:07:20:58 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.47.0" "-"
10.255.0.3 - - [08/Mar/2017:07:20:59 +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>
```

Docker sends every other request (round robin) 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  PORTS
n6r23600qp4s  website.1  nginx:1.11    ubuntu  Running        Running 12 minutes
ago
pv64kr6ulh1m  website.2  nginx:1.11    ubuntu  Running        Running 12 minutes
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  PORTS
n6r23600qp4s  website.1  nginx:1.11    ubuntu  Running        Running 13 minutes
ago
pv64kr6ulh1m  website.2  nginx:1.11    ubuntu  Running        Running 13 minutes
ago
1k3u8gcfwhp5  website.3  nginx:1.11    ubuntu  Running        Running 4 seconds
ago

user@ubuntu:~$ docker container ls

CONTAINER ID        IMAGE
COMMAND                  CREATED             STATUS              PORTS
NAMES
13eaee71c8b6
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   19 seconds ago      Up 19 seconds       80/tcp, 443/tcp
website.3.1k3u8gcfwhp5ihaj5ie1eaitx
65cdd00293c8
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   13 minutes ago      Up 13 minutes       80/tcp, 443/tcp
website.1.n6r23600qp4s3xb0uelaxnc1x
0c76de7d0183
nginx@sha256:52a189e49c0c797cfc5cbfe578c68c225d160fb13a42954144b29af3fe4fe335
"nginx -g 'daemon ..."   13 minutes ago      Up 13 minutes       80/tcp, 443/tcp
website.2.pv64kr6ulh1m2jplqodm7zcfj

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 `service rm` subcommand. Remove your website service:

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

website

user@ubuntu:~$ docker service ls

ID  NAME  MODE  REPLICAS  IMAGE
```

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 `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 `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.
```

[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 container exec` into it:

```
user@ubuntu:~$ docker container ls

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:1.11          "nginx -g 'daemon off"   5 minutes ago
Up 5 minutes        80/tcp, 443/tcp     website.2.8mwtjjttnqv25mqau17vxz8sf
5778636029c2        nginx:1.11          "nginx -g 'daemon off"   5 minutes ago
```

```
Up 5 minutes        80/tcp, 443/tcp     website.1.1kddy0zpcbej0747ul1muby7d

user@ubuntu:~$ docker container 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!