Docker SWARM Hands-on

Minimum Requirements

- 2+ hosts with Docker installed
    - 1+ Manager node(s)
        - 3 or 5 or 7 required for high availability
    - 1+ worker node(s)
- Hosts connected to a network and can reach one another
- All hosts can access the Docker images registry
    - docker.io
    - access.redhat.com

Demo configuration

- Docker 1.12
- VirtualBox host-only network 192.168.33.0/24
- Vagrant CENTOS 7 box on VirtualBox provider
- VMs connect via NAT network to Internet to reach docker.io central hub
- **swarma** - 1 Docker SWARM Manager node (CENTOS 7.3 Vagrant box VM)
- **swarmb** - 1 Docker SWARM Worker node (CENTOS 7.3 Vagrant box VM)
- **moby** - 1 Docker SWARM Worker node (Mac OSX Vagrant HOST)

Multi-host Vagrant file launching the VMs on the host-only network

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in
Vagrant.configure
```

```
# configures the configuration version (we support
older styles for
# backwards compatibility). Please don't change it
unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # SWARM VMs
  config.vm.define "swarmA" do |swarmA|
    swarmA.vm.box = "centos/7"
    swarmA.vm.network "private_network", ip:
"192.168.33.10"
    swarmA.vm.provider "virtualbox" do |vba|
      # Display the VirtualBox GUI when booting the
machine
      vba.gui = true
      # Customize the amount of memory on the VM:
      vba.memory = "1024"
    end
    swarmA.vm.provision "shell", inline: <<-SHELL
      yum -y update
      yum -y install epel docker
    SHELL
  end
  config.vm.define "swarmB" do |swarmB|
    swarmB.vm.box = "centos/7"
    swarmB.vm.network "private_network", ip:
"192.168.33.11"
    swarmB.vm.provider "virtualbox" do |vbb|
      # Display the VirtualBox GUI when booting the
machine
      vbb.gui = true
      # Customize the amount of memory on the VM:
      vbb.memory = "1024"
    end
    swarmB.vm.provision "shell", inline: <<-SHELL
      yum -y update
      yum -y install epel docker
    SHELL
  end
```

```
end
```

0 Extras

You can visualize the SWARM cluster by deploying the Docker SWARM Visualizer as shown below on the SWARM Manager node

```
# Needed to access Docker socket
$ sudo docker run -it -d \
                  -p 8080:8080 \
                  --privileged  \
                  -v
/var/run/docker.sock:/var/run/docker.sock \
                  dockersamples/visualizer
```

Point your browser to Docker SWARM manager to http://192.168.33.10:8080 and should see a similar page

1 Basic SWARM topology

2 Create SWARM

2.1 Choose a manager node and activate SWARM

On **swarma** VM we activate Docker SWARM

```
$ sudo docker swarm init -h
Flag shorthand -h has been deprecated, please use --
help
```

```
Usage:  docker swarm init [OPTIONS]

Initialize a swarm

Options:
      --advertise-addr string           Advertised
address (format: <ip|interface>[:port])
      --cert-expiry duration            Validity
period for node certificates (default 2160h0m0s)
      --dispatcher-heartbeat duration   Dispatcher
heartbeat period (default 5s)
      --external-ca value
Specifications of one or more certificate signing
endpoints
      --force-new-cluster               Force create
a new cluster from current state.
      --help                            Print usage
      --listen-addr value               Listen
address (format: <ip|interface>[:port]) (default
0.0.0.0:2377)
      --task-history-limit int          Task history
retention limit (default 5)

$ sudo docker swarm init
Error response from daemon: could not choose an IP
address to advertise since this system has multiple
addresses on different interfaces (10.0.2.15 on eth0
and 192.168.33.10 on eth1) - specify one with --
advertise-addr
```

We need to make sure other nodes can reach the master in order to be able to join the SWARM cluster

The IP address **192.168.33.10** sits on the host-only network we chose to connect all VMs and HOST together

```
$ sudo docker swarm init --advertise-addr
192.168.33.10
Swarm initialized: current node
(c3xr3f2h2zdg0stqkmgqcam2c) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-
2eib002f2hg37zpqixpzv1vdmk7to8gbl8tjrbtiypoe5qoa95-
65vgl10kalih2d8rl96gjp8pd \
    192.168.33.10:2377

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

The command above is very important as it is needed to be run on all of the nodes willing to join the SWARM cluster

Let's inspect what is happening on the Docker SWARM Manager node

```
$ sudo docker node ls
ID                            HOSTNAME  STATUS
AVAILABILITY  MANAGER STATUS
c3xr3f2h2zdg0stqkmgqcam2c *  swarma    Ready   Active
Leader
```

```
$ sudo docker info
```

```
...
Swarm: active
```

```
 NodeID: c3xr3f2h2zdg0stqkmgqcam2c
 Is Manager: true
 ClusterID: 4tkzs28qmbcix0sl36mjpv1ef
 Managers: 1
 Nodes: 3
 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: 192.168.33.10
 ...
```

We can that only SWARM Manager node is on-line and ready to accept connections from SWARM workers

```
$ sudo docker network ls
NETWORK ID          NAME                DRIVER
SCOPE
7d92ba8955be        bridge              bridge
local
ed4e70ba82af        docker_gwbridge     bridge
local
6013defd4e0a        host                host
local
ci824bxbs4a2        ingress             overlay
swarm
```

We can see that 2 more networks exist with respect to the standard Docker network setup.

> **docker_gwbridge**
>
> This is the network created by Docker which allows the containers to connect to the host that it is running on

> **ingress**
>
> This is the network created by Docker which Docker swarm uses to expose services to the external network and provide the routing mesh

Let's inspect the **ingress** network which is of type **swarm**

```
$ sudo docker network inspect ingress
```


```
[
    {
        "Name": "ingress",
        "Id": "ci824bxbs4a20bm3tf0icl4z2",
        "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": {
            "ingress-sbox": {
```

```
                "Name": "ingress-endpoint",
                "EndpointID":
"96a26d0c344d37de02ef95155a6e845c52cfbc9a47bf30aa7538
aad121a999a1",
                "MacAddress": "02:42:0a:ff:00:03",
                "IPv4Address": "10.255.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {

"com.docker.network.driver.overlay.vxlanid_list":
"256"
        },
        "Labels": {}
    }
]

$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS      PORTS          NAMES
```

We can see there is a container **ingress-sbox** connected to the **ingress** network

We try to see the running containers, but none is visible!

Let's see the other new network after SWARM is activated

```
$ sudo docker network inspect docker_gwbridge
```

```
[
    {
        "Name": "docker_gwbridge",
        "Id":
```

```
"ed4e70ba82afb015a7bea4ee8d5f5db9916bc9c998404f4f5370
dda064136b08",
        "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,
        "Containers": {
            "ingress-sbox": {
                "Name": "gateway_ingress-sbox",
                "EndpointID":
"5ddf5626f02c3af30a55b336d87a1ff1feaad16764e1d5846018
f52b5a5e4f80",
                "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": {}
    }
]
```

The **ingress-sbox** is a hidden container running afte SWARM mmode is activated and allows to connect containers running across host network as well as across remote SWARM nodes on the external network

Voila!

```
SWARM Network on the Manager Node swarma

  172.18.0.2
 +----+---------------------------------+
docker_gwbridge
      |
      |
   +--+--------------+
   |  ingress-sbox   |
   +--------+--------+
            |
            |
       +----+-------------------------------------+
ingress
        10.255.0.3
```

2.2 Connect SWARM Worker nodes

We execute the command on the **swarmb** VM to join the SWARM cluster

```
$ uname -a
Linux swarmb 3.10.0-514.16.1.el7.x86_64 #1 SMP Wed
Apr 12 15:04:24 UTC 2017 x86_64 x86_64 x86_64
GNU/Linux
```

```
$ sudo docker swarm join \
       --token SWMTKN-1-
2eib002f2hg37zpqixpzv1vdmk7to8gbl8tjrbtiypoe5qoa95-
65vgl10kalih2d8rl96gjp8pd \    192.168.33.10:2377
This node joined a swarm as a worker.
```

On the SWARM manager node **swarma** we check the nodes

```
$ sudo docker node ls
ID                            HOSTNAME  STATUS
AVAILABILITY  MANAGER STATUS
6ppf1p1ry4x3h9crserynkofs     swarmb    Ready   Active
c3xr3f2h2zdg0stqkmgqcam2c *   swarma    Ready   Active
Leader
```

Let's check the Worker node **swarmb** network interfaces

```
$ sudo docker network ls
NETWORK ID          NAME                DRIVER
SCOPE
7c2a307b6372        bridge              bridge
local
147feea65e7e        docker_gwbridge     bridge
local
650e48829991        host                host
local
ci824bxbs4a2        ingress             overlay
swarm
c2d96dcc5a1b        none                null
local
```

```
$ sudo docker network inspect docker_gwbridge
```

```
[
    {
        "Name": "docker_gwbridge",
        "Id":
"147feea65e7efbb3e138c445b6bdc98baa4e8fe1a91554f13657
19e0f5aa9c7b",
        "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,
        "Containers": {
            "ingress-sbox": {
                "Name": "gateway_ingress-sbox",
                "EndpointID":
"a98677533f801ecd9ffd4a0a522ea3860cfd080027d0578b5f3c
c9af95ff1e78",
                "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": {}
    }
]
```

```
$ sudo docker network inspect ingress
```

```
[
    {
        "Name": "ingress",
        "Id": "ci824bxbs4a20bm3tf0icl4z2",
        "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": {
            "ingress-sbox": {
                "Name": "ingress-endpoint",
```

```
                "EndpointID":
"18727772a1718e817a44d6d97c3567c3f4c125d36afdaba020da
fe7fa619841e",
                "MacAddress": "02:42:0a:ff:00:06",
                "IPv4Address": "10.255.0.6/16",
                "IPv6Address": ""
            }
        },
        "Options": {

"com.docker.network.driver.overlay.vxlanid_list":
"256"
        },
        "Labels": {}
    }
]
```

The network toplogy for the SWARM cluster looks like

```
SWARM Network

-------------------------------------+----------
----------------------------
Manager Node swarma                          | Worker
node swarmb
                                             |
 172.18.0.2                                  | 172.18.0.2
 +----+-----------+ docker_gwbridge          | +----+----
-------+ docker_gwbridge
      |                                      |      |
      |                                      |      |
   +--+--------------+                       |   +--+----
----------+
   |  ingress-sbox   |                       |   |
ingress-sbox   |
   +--------+--------+                       |   +-------
-+--------+
```

```
            |                                   |
|
            |                                   |
|
      +-----+-----------------------------------
-+-------------+ ingress
        10.255.0.3                              |
10.255.0.6
------------------------------------+-----------
----------------------------
```

We also add the Mac OSX HOST node as the 2nd Worker

```
$ uname -a
Darwin localhost.local 15.6.0 Darwin Kernel Version
15.6.0: Fri Feb 17 10:21:18 PST 2017; root:xnu-
3248.60.11.4.1~1/RELEASE_X86_64 x86_64
```

```
$ sudo docker swarm join \
         --token SWMTKN-1-
2eib002f2hg37zpqixpzv1vdmk7to8gbl8tjrbtiypoe5qoa95-
65vgl10kalih2d8rl96gjp8pd \ 192.168.33.10:2377
This node joined a swarm as a worker.
```

On the SWARM manager node **swarma** we check the nodes

```
$ sudo docker node ls
ID                           HOSTNAME  STATUS
AVAILABILITY  MANAGER STATUS
6ppf1p1ry4x3h9crserynkofs    swarmb    Ready   Active
c3xr3f2h2zdg0stqkmgqcam2c *  swarma    Ready   Active
Leader
```

```
cdihi1s4pkwg7896u8cxmweco    moby      Ready   Active
```

2.3 Deploy a SWARM service on the cluster

We use the simple NGINX web service and start it on the SWARM cluster from the SWARM manager node **swarma**

```
docker service create \
       --name my-web \
       --publish 8080:80 \
       --replicas 2 \
       nginx
6q8bu5hdq3c2af45ufetegqkm
```

Let's check what containers run on Manager node

```
docker ps
CONTAINER ID IMAGE         COMMAND
CREATED            STATUS            PORTS  NAMES
d772a3a6ae38 nginx:latest "nginx -g 'daemon off"
About a minute ago Up About a minute 80/tcp my-
web.1.76evp8s0rjcnqfnlv0vpr6p5b
```

We can see only one container, but we have asked for 2 replicas!

Let's inspect the SWARM service now

```
$ sudo docker service ls
ID            NAME    REPLICAS  IMAGE  COMMAND
6q8bu5hdq3c2  my-web  2/2       nginx
```

OK! We have 2 replicas!

Let's see where are the containers running and we use its ID

```
$ sudo docker service ps 6q8bu5hdq3c2
ID                         NAME      IMAGE  NODE
DESIRED STATE  CURRENT STATE           ERROR
76evp8s0rjcnqfnlv0vpr6p5b  my-web.1  nginx  swarma
Running        Running 6 minutes ago
bvfry96ncerhijqk7byqsmd7u  my-web.2  nginx  moby
Running        Running 6 minutes ago
```

We can see the 2 replicas of the NGINX container run on Manager node and on Mac OSX node.

Let's inspect the service configuration

```
$ sudo docker service inspect --pretty my-web
```

```
ID:             6q8bu5hdq3c2af45ufetegqkm
Name:           my-web
Mode:           Replicated
 Replicas:      2
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
ContainerSpec:
 Image:         nginx
Resources:
Ports:
 Protocol = tcp
 TargetPort = 80
```

```
 PublishedPort = 8080
```

We can try to scale up the service to use more replicas across the SWARM cluster

```
$ sudo docker service scale my-web=4
my-web scaled to 4

$ sudo docker service inspect --pretty my-web
ID:             6q8bu5hdq3c2af45ufetegqkm
Name:           my-web
Mode:           Replicated
 Replicas:      4
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
ContainerSpec:
 Image:         nginx
Resources:
Ports:
 Protocol = tcp
 TargetPort = 80
 PublishedPort = 8080

$ sudo docker service ls
ID            NAME    REPLICAS  IMAGE  COMMAND
6q8bu5hdq3c2  my-web  4/4       nginx
```

OK! We can see the 4 replicas are running now on the cluster so we scaled up successfully!

Now let's see where the container replicas are running

```
$ sudo docker service ps my-web
ID                         NAME      IMAGE  NODE
DESIRED STATE  CURRENT STATE           ERROR
76evp8s0rjcnqfnlv0vpr6p5b  my-web.1  nginx  swarma
Running        Running 25 minutes ago
bvfry96ncerhijqk7byqsmd7u  my-web.2  nginx  moby
Running        Running 25 minutes ago
f44qu18w6o3u77501fyjk1q3v  my-web.3  nginx  swarmb
Running        Running 17 seconds ago
31kfa3tar52dp6hgi3xf7lvwb  my-web.4  nginx  swarmb
Running        Running 17 seconds ago
```

Let's see how we can tell node to empty its workload, we drain the node to makse not service replicas are running on it

```
$ sudo docker node update --availability drain swarma
swarma

$ sudo docker node ls
ID                           HOSTNAME  STATUS
AVAILABILITY  MANAGER STATUS
6ppf1p1ry4x3h9crserynkofs    swarmb    Ready   Active
c3xr3f2h2zdg0stqkmgqcam2c *  swarma    Ready   Drain
Leader
cdihi1s4pkwg7896u8cxmweco    moby      Ready   Active
```

We can see that Manager node **swarma** is in Drain availability state

Let's see how the service behaves after the Manager node being drained

```
$ sudo docker service ps 6q8bu5hdq3c2
ID                         NAME          IMAGE  NODE
DESIRED STATE  CURRENT STATE           ERROR
```

```
9jnelp7x37wz3t4nog379lkz7  my-web.1      nginx  moby
Running        Running 9 seconds ago
76evp8s0rjcnqfnlv0vpr6p5b   \_ my-web.1  nginx
swarma  Shutdown       Shutdown 11 seconds ago
bvfry96ncerhijqk7byqsmd7u  my-web.2      nginx  moby
Running        Running 40 minutes ago
f44qu18w6o3u77501fyjk1q3v  my-web.3      nginx
swarmb  Running        Running 15 minutes ago
31kfa3tar52dp6hgi3xf7lvwb  my-web.4      nginx
swarmb  Running        Running 15 minutes ago
```

OK! The SWARM keeps SLA, e.g. 4 replicas are running on the SWARM cluster, but no replica on the **swarma** Manager node which is marked as being drained.

2.4 Use Docker SWARM mode routing mesh

We have deployed **my-web** service on the SWARM cluster

Let's review again its configuration

```
docker service inspect --pretty my-web
ID:             6q8bu5hdq3c2af45ufetegqkm
Name:           my-web
Mode:           Replicated
 Replicas:      4
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
ContainerSpec:
 Image:         nginx
Resources:
Ports:
 Protocol = tcp
 TargetPort = 80
```

```
  PublishedPort = 8080
```

It says the service has published port **8080** so we should be able to connect to the public endpoint on this port

Let's see on all SWARM cluster nodes if this is true

swarma

```
$ sudo netstat -tuplan | grep LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*
LISTEN      1/systemd
tcp        0      0 0.0.0.0:22              0.0.0.0:*
LISTEN      1012/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*
LISTEN      2112/master
tcp6       0      0 :::111                  :::*
LISTEN      1/systemd
tcp6       0      0 :::8080                 :::*
LISTEN      1013/dockerd-curren
tcp6       0      0 :::22                   :::*
LISTEN      1012/sshd
tcp6       0      0 ::1:25                  :::*
LISTEN      2112/master
tcp6       0      0 :::2377                 :::*
LISTEN      1013/dockerd-curren
tcp6       0      0 :::7946                 :::*
LISTEN      1013/dockerd-curren
```

swarmb

```
netstat -tuplan | grep LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*
LISTEN      1/systemd
tcp        0      0 0.0.0.0:22              0.0.0.0:*
```

```
LISTEN      1004/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*
LISTEN      2138/master
tcp6       0      0 :::111                  :::*
LISTEN      1/systemd
tcp6       0      0 :::8080                 :::*
LISTEN      1007/dockerd-curren
tcp6       0      0 :::22                   :::*
LISTEN      1004/sshd
tcp6       0      0 ::1:25                  :::*
LISTEN      2138/master
tcp6       0      0 :::7946                 :::*
LISTEN      1007/dockerd-curren
```

moby

```
netstat -natl | grep LISTEN
tcp4       0      0  127.0.0.1.5556         *.*
LISTEN
tcp4       0      0  192.168.33.1.5556      *.*
LISTEN
tcp4       0      0  10.37.129.2.5556       *.*
LISTEN
tcp4       0      0  10.211.55.2.5556       *.*
LISTEN
tcp4       0      0  25.48.51.26.5556       *.*
LISTEN
tcp4       0      0  192.168.1.111.5556     *.*
LISTEN
tcp4       0      0  192.168.1.111.45290    *.*
LISTEN
tcp6       0      0  ::1.8080
*.*                                          LISTEN
tcp4       0      0  *.8080                 *.*
LISTEN
.
.
```

```
.
```

OK, Docker SWARM public endpoint is available on all of them!

Let's see

Host	URL
swarma	192.168.33.10:8080
swarmb	192.168.33.11:8080
moby	192.168.33.1:8080

```
$ curl 192.168.33.10:8080
```

```
<!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>
```

```
<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 192.168.33.11:8080
```

```
<!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>

<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 192.168.33.1:8080
```

```
<!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>

<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>
```

OK! All of them are accesible!

Let's see how the load balancing works

We attack the SWARM service via Manager node URL to see if always the same replica serves the HTTP request

```
$ for fff in `find /etc`; do; curl
192.168.33.10:8080; done
```

Here is the screen capture of the load balancing HTTP requests across cluster

2.5 Undeploy a service from a SWARM cluster

In order to stop the SWARM service you can do the the following

```
$ sudo docker service ls
ID            NAME    REPLICAS  IMAGE  COMMAND
d90di81ywkzo  my-web  3/3       nginx

$sudo docker service ps d90di81ywkzo
ID                         NAME      IMAGE  NODE
DESIRED STATE  CURRENT STATE           ERROR
amb625kyyvxo5wzhr28rq8yen  my-web.1  nginx  swarma
Running        Running 23 minutes ago
9b3m5yel7u4nyj6hkblr1korm  my-web.2  nginx  moby
Running        Running 23 minutes ago
```

```
901hgnpb24ik9sm9pai8f6hsv  my-web.3  nginx  swarmb
Running        Running 23 minutes ago

$ sudo docker service rm my-web
my-web

$ sudo docker service ls
ID            NAME    REPLICAS  IMAGE  COMMAND
```