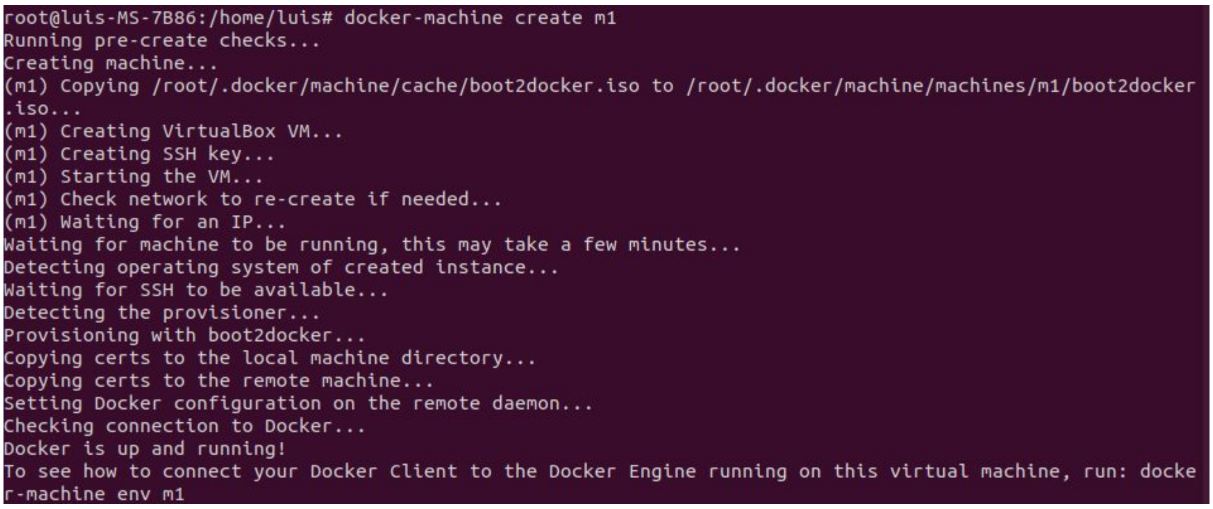
Práctica 3. Docker Swarm: Combinando múltiples máquinas para la ejecución de contenedores Docker.

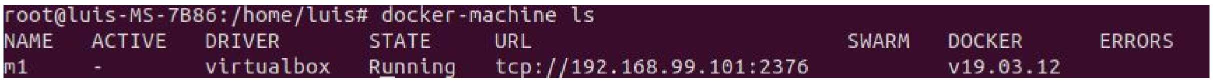
Nombre: Luis Miguel Aguilar González

1.- La creación de las máquinas virtuales con docker-machine.

- docker-machine create m1

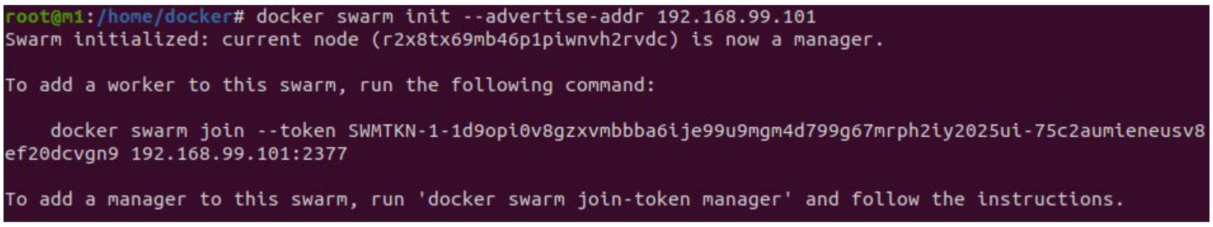


```
root@luis-MS-7B86:/home/luis# docker-machine create m1
Running pre-create checks...
Creating machine...
(m1) Copying /root/.docker/machine/cache/boot2docker.iso to /root/.docker/machine/machines/m1/boot2docker
.iso...
(m1) Creating VirtualBox VM...
(m1) Creating SSH key...
(m1) Starting the VM...
(m1) Check network to re-create if needed...
(m1) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docke
r-machine env m1
```



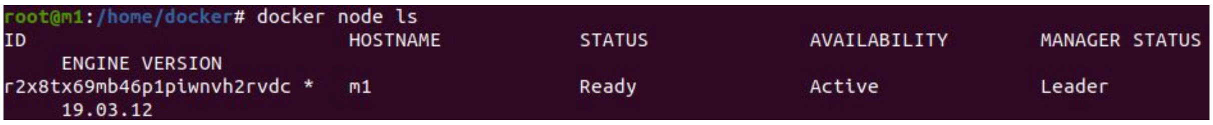
```
root@luis-MS-7B86:/home/luis# docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
m1     -        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.12
```

- docker-machine ssh m1
- docker swarm init --advertise-addr 192.168.99.101



```
root@m1:/home/docker# docker swarm init --advertise-addr 192.168.99.101
Swarm initialized: current node (r2x8tx69mb46p1piwnvh2rvdc) is now a manager.

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

    docker swarm join --token SWMTKN-1-1d9opi0v8gzxvmbbba6ije99u9mgm4d799g67mrph2iy2025ui-75c2aumieneusv8
ef20dcvgn9 192.168.99.101:2377

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

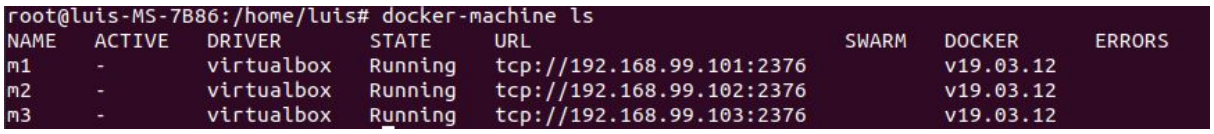
- docker node ls



```
root@m1:/home/docker# docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
     ENGINE VERSION
r2x8tx69mb46p1piwnvh2rvdc *   m1                  Ready               Active              Leader
     19.03.12
```

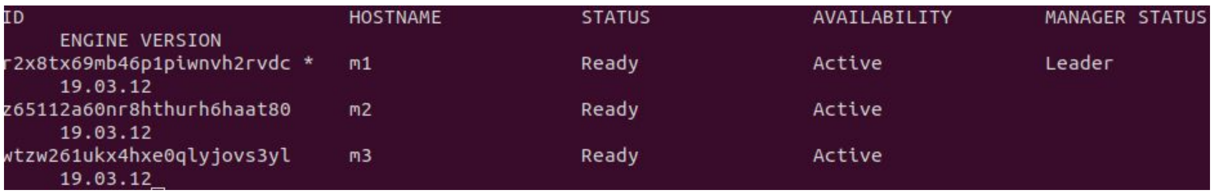
- docker-machine create m2
- docker-machine ssh m2
- docker swarm join --token SWMTKN-1-1d9opi0v8gzxvmbbba6ije99u9mgm4d799g67mrph2iy2025ui-75c2aumieneusv8ef20dcvgn9 192.168.99.101:237

- docker-machine create m3
- docker-machine ssh m3
- docker swarm join --token SWMTKN-1-1d9opi0v8gzxvmbbba6ije99u9mgm4d799g67mrph2iy2025ui-75c2aumieneusv8ef20dcvgn9 192.168.99.101:2377



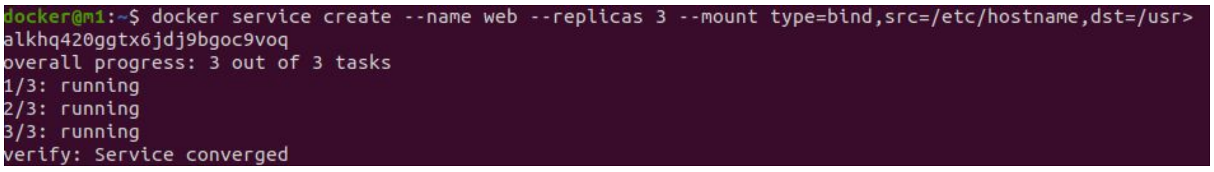
```
root@luis-MS-7B86:/home/luis# docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                         SWARM   DOCKER     ERRORS
m1     -        virtualbox   Running   tcp://192.168.99.101:2376           v19.03.12
m2     -        virtualbox   Running   tcp://192.168.99.102:2376           v19.03.12
m3     -        virtualbox   Running   tcp://192.168.99.103:2376           v19.03.12
```

En el nodo m1 podemos comprobar que todos los nodos han sido añadidos correctamente:



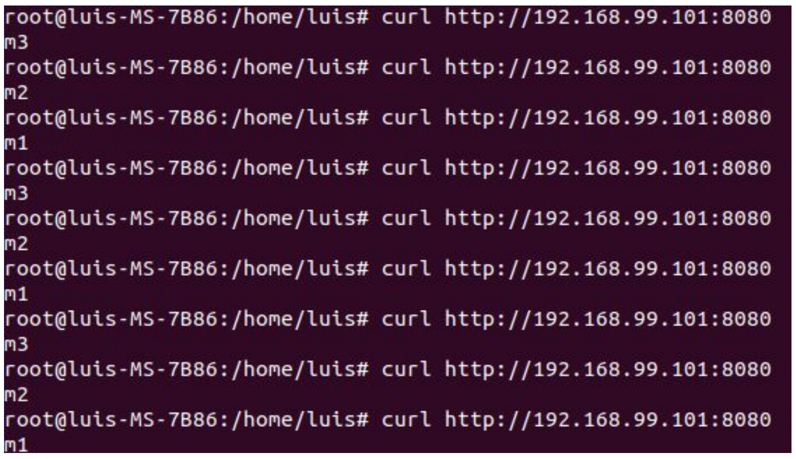
```
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
     ENGINE VERSION
r2x8tx69mb46p1piwnvh2rvdc *   m1                  Ready               Active              Leader
     19.03.12
z65112a60nr8hthurh6haat80     m2                  Ready               Active
     19.03.12
wtzw261ukx4hxe0qlyjovs3yl     m3                  Ready               Active
     19.03.12
```

2.- Ejecución del servicio web.



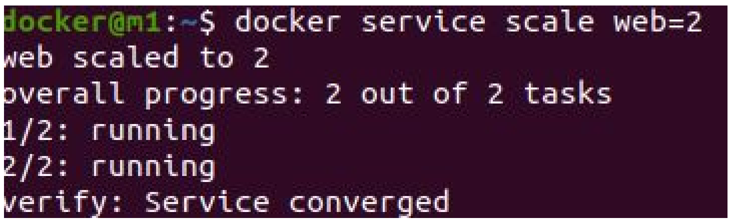
```
docker@m1:~$ docker service create --name web --replicas 3 --mount type=bind,src=/etc/hostname,dst=/usr>
alkhq420ggtx6jdj9bgoc9voq
overall progress: 3 out of 3 tasks
1/3: running
2/3: running
3/3: running
verify: Service converged
```

- Cuando los 3 nodos están activos

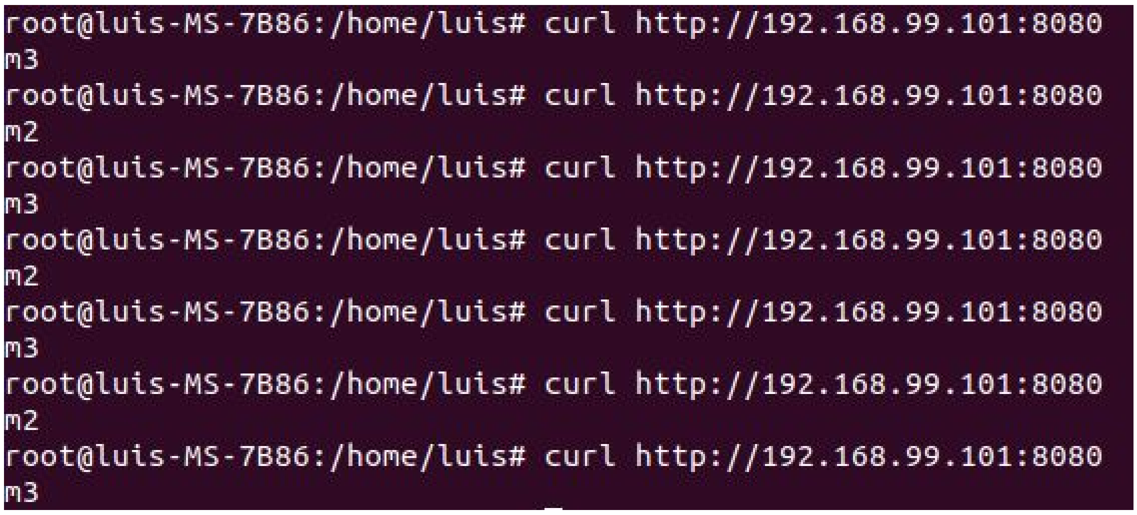


```
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
```

- Cuando se cambia de escala a 2



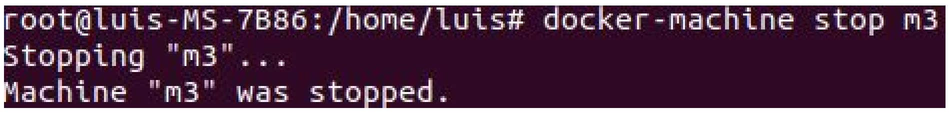
```
docker@m1:~$ docker service scale web=2
web scaled to 2
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
```



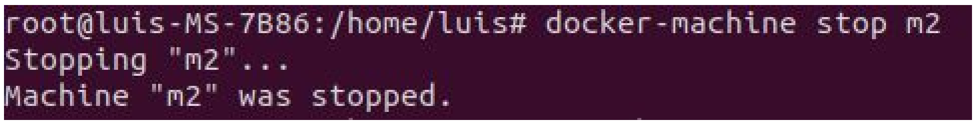
```
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m2
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m3
```

- Cuando apagamos un nodo activo y sólo ejecuta un nodo

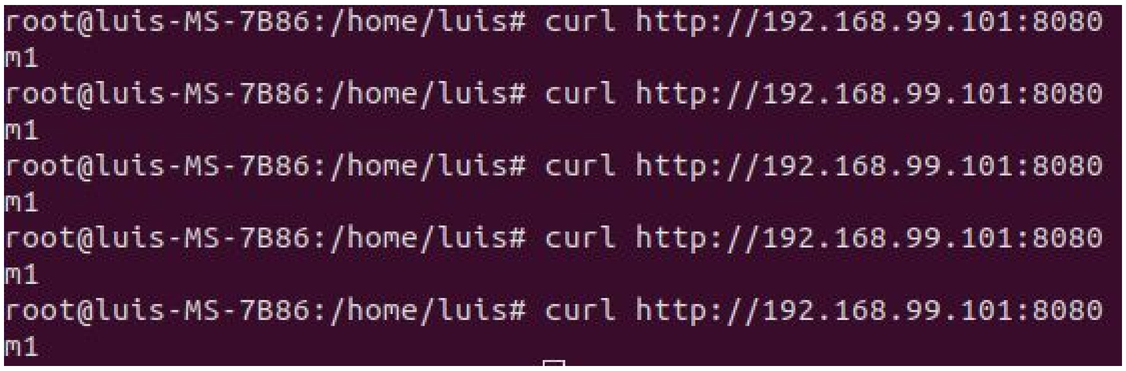
Si solo apagamos uno de los 3 nodos activos el scale web=2 antes ejecutado reestructura de forma que se balancea entre los 2 nodos restantes por eso apago el otro nodo, ejecutando así en solo un nodo.



```
root@luis-MS-7B86:/home/luis# docker-machine stop m3
Stopping "m3"...
Machine "m3" was stopped.
```



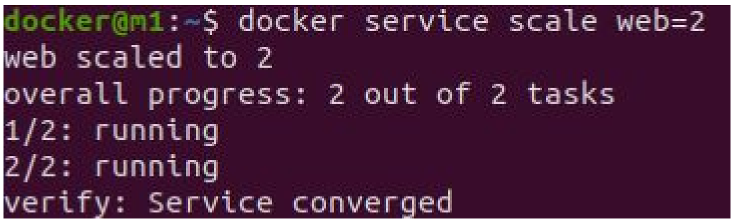
```
root@luis-MS-7B86:/home/luis# docker-machine stop m2
Stopping "m2"...
Machine "m2" was stopped.
```



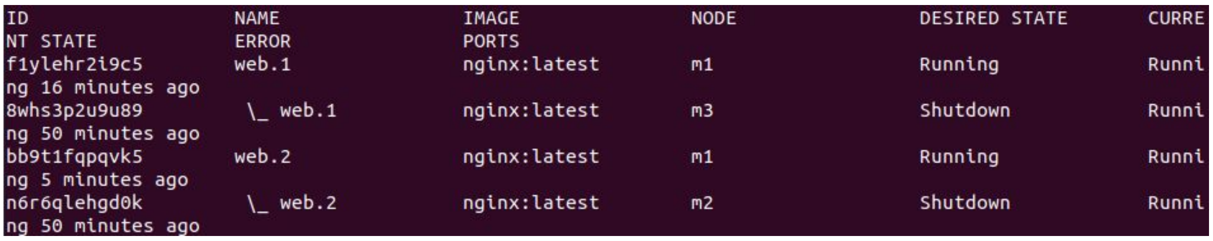
```
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
root@luis-MS-7B86:/home/luis# curl http://192.168.99.101:8080
m1
```

- Activación automática del segundo nodo

Cuando se cambia de nuevo la escala de nodos activos a 2 automáticamente se activará otro nodo para complementar al nodo actual:



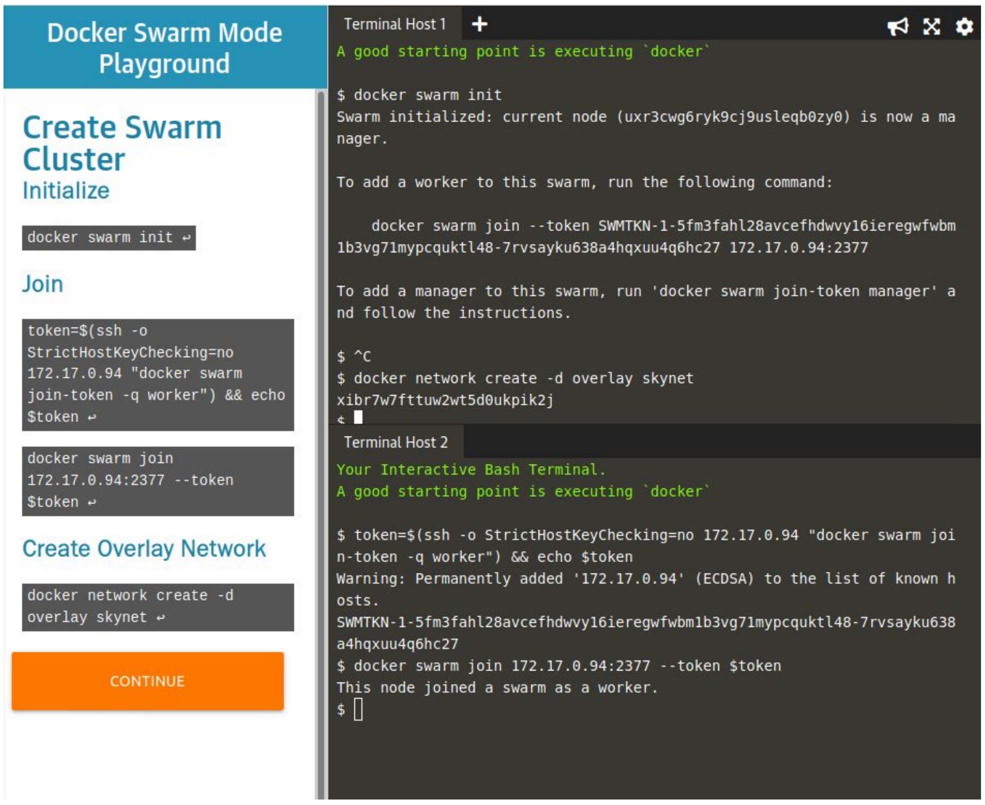
```
docker@m1:~$ docker service scale web=2
web scaled to 2
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
```



```
ID                  NAME                IMAGE               NODE                DESIRED STATE       CURRE
NT STATE            ERROR               PORTS
f1ylehr2i9c5        web.1               nginx:latest        m1                  Running             Runni
ng 16 minutes ago
8whs3p2u9u89         \_ web.1           nginx:latest        m3                  Shutdown            Runni
ng 50 minutes ago
bb9t1fqpqvk5        web.2               nginx:latest        m1                  Running             Runni
ng 5 minutes ago
n6r6qlehgd0k         \_ web.2           nginx:latest        m2                  Shutdown            Runni
ng 50 minutes ago
```

3.- Capturas de diversas ejecuciones en la plataforma katacoda

- Create a swarm cluster



Docker Swarm Mode Playground

Create Swarm Cluster

Initialize

```
docker swarm init ↵
```

Join

```
token=$(ssh -o StrictHostKeyChecking=no 172.17.0.94 "docker swarm join-token -q worker") && echo $token ↵
```

```
docker swarm join 172.17.0.94:2377 --token $token ↵
```

Create Overlay Network

```
docker network create -d overlay skynet ↵
```

CONTINUE

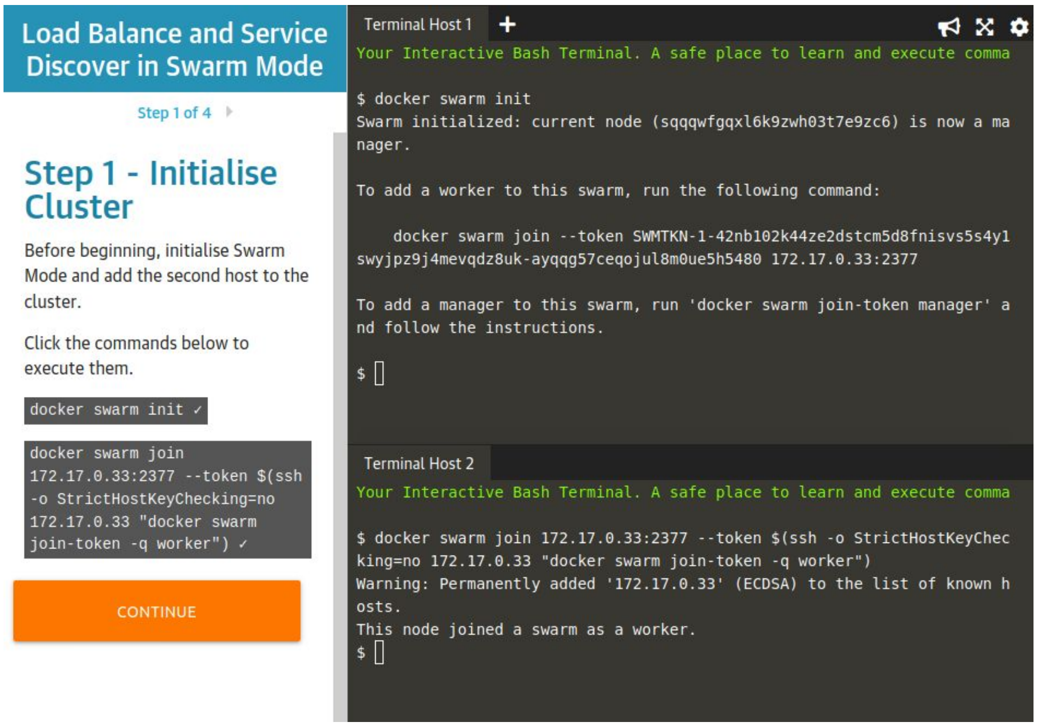
Terminal Host 1

```
A good starting point is executing `docker`

$ docker swarm init
Swarm initialized: current node (uxr3cwg6ryk9cj9usleqb0zy0) is now a ma
nager.

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

    docker swarm join --token SWMTKN-1-5fm3fahl28avcefhdwvy16ieregwfwbm
1b3vg71mypcquktl48-7rvsayku638a4hqxuu4q6hc27 172.17.0.94:2377

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

$ ^C
$ docker network create -d overlay skynet
xibr7w7fttuw2wt5d0ukpik2j
$ 
```

Terminal Host 2

```
Your Interactive Bash Terminal.
A good starting point is executing `docker`

$ token=$(ssh -o StrictHostKeyChecking=no 172.17.0.94 "docker swarm joi
n-token -q worker") && echo $token
Warning: Permanently added '172.17.0.94' (ECDSA) to the list of known h
osts.
SWMTKN-1-5fm3fahl28avcefhdwvy16ieregwfwbm1b3vg71mypcquktl48-7rvsayku638
a4hqxuu4q6hc27
$ docker swarm join 172.17.0.94:2377 --token $token
This node joined a swarm as a worker.
$ 
```

- Load Balance and Service Disconver in Swarm Mode



Step 1 - Initialise Cluster

Before beginning, initialise Swarm Mode and add the second host to the cluster.

Click the commands below to execute them.

```
docker swarm init ✓
```

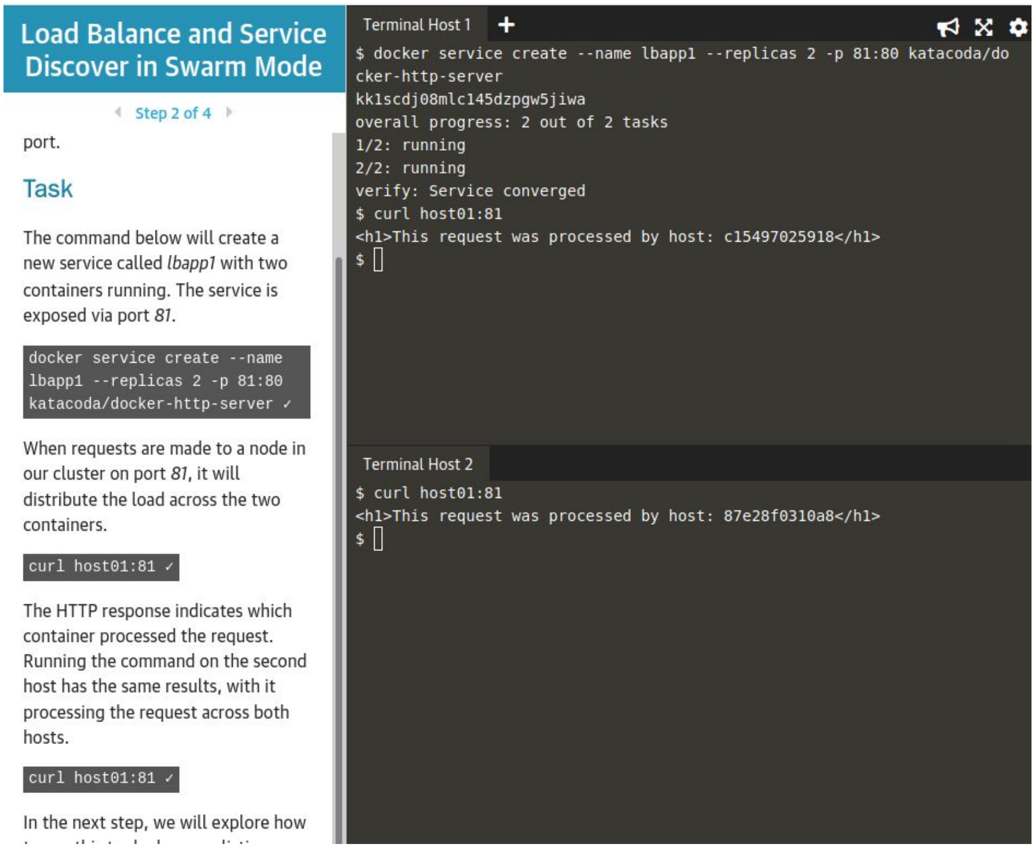
```
docker swarm join 172.17.0.33:2377 --token $(ssh -o StrictHostKeyChecking=no 172.17.0.33 "docker swarm join-token -q worker") ✓
```

Terminal Host 1:

```
Your Interactive Bash Terminal. A safe place to learn and execute comma

$ docker swarm init
Swarm initialized: current node (sqqqwfgqxl6k9zwh03t7e9zc6) is now a ma
nager.

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

    docker swarm join --token SWMTKN-1-42nb102k44ze2dstcm5d8fnisvs5s4y1
swyjpz9j4mevqdz8uk-ayqqg57ceqojul8m0ue5h5480 172.17.0.33:2377

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

$
```

Terminal Host 2:

```
Your Interactive Bash Terminal. A safe place to learn and execute comma

$ docker swarm join 172.17.0.33:2377 --token $(ssh -o StrictHostKeyChec
king=no 172.17.0.33 "docker swarm join-token -q worker")
Warning: Permanently added '172.17.0.33' (ECDSA) to the list of known h
osts.
This node joined a swarm as a worker.
$
```



port.

Task

The command below will create a new service called *lbapp1* with two containers running. The service is exposed via port *81*.

```
docker service create --name lbapp1 --replicas 2 -p 81:80 katacoda/docker-http-server ✓
```

When requests are made to a node in our cluster on port *81*, it will distribute the load across the two containers.

```
curl host01:81 ✓
```

The HTTP response indicates which container processed the request. Running the command on the second host has the same results, with it processing the request across both hosts.

```
curl host01:81 ✓
```

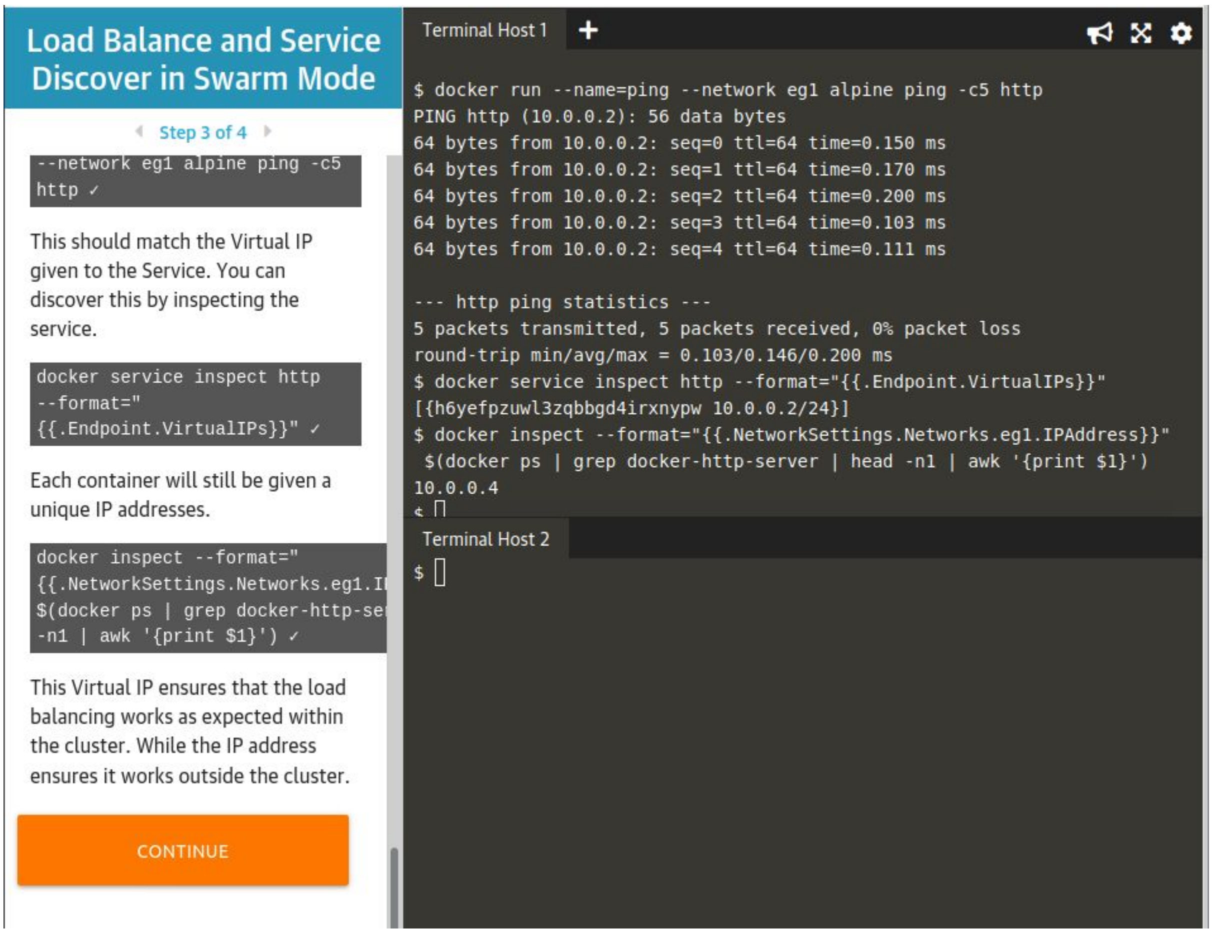
In the next step, we will explore how

Terminal Host 1:

```
$ docker service create --name lbapp1 --replicas 2 -p 81:80 katacoda/do
cker-http-server
kklscdj08mlc145dzpgw5jiwa
overall progress: 2 out of 2 tasks
1/2: running
2/2: running
verify: Service converged
$ curl host01:81
<h1>This request was processed by host: c15497025918</h1>
$
```

Terminal Host 2:

```
$ curl host01:81
<h1>This request was processed by host: 87e28f0310a8</h1>
$
```



Load Balance and Service Discover in Swarm Mode

Step 3 of 4

```
--network eg1 alpine ping -c5
http ✓
```

This should match the Virtual IP given to the Service. You can discover this by inspecting the service.

```
docker service inspect http
--format="
{{.Endpoint.VirtualIPs}}" ✓
```

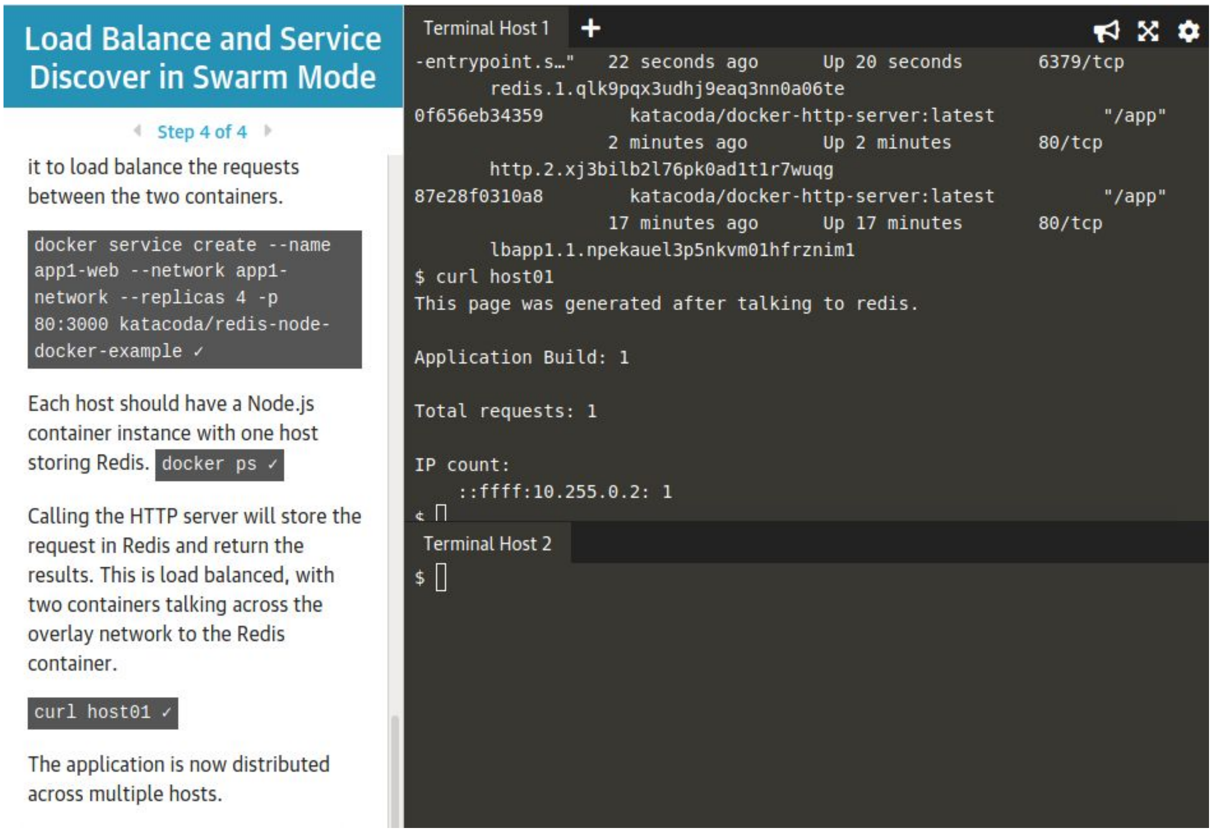
Each container will still be given a unique IP addresses.

```
docker inspect --format="
{{.NetworkSettings.Networks.eg1.I
$(docker ps | grep docker-http-se
-n1 | awk '{print $1}') ✓
```

This Virtual IP ensures that the load balancing works as expected within the cluster. While the IP address ensures it works outside the cluster.

```
$ docker run --name=ping --network eg1 alpine ping -c5 http
PING http (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: seq=0 ttl=64 time=0.150 ms
64 bytes from 10.0.0.2: seq=1 ttl=64 time=0.170 ms
64 bytes from 10.0.0.2: seq=2 ttl=64 time=0.200 ms
64 bytes from 10.0.0.2: seq=3 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: seq=4 ttl=64 time=0.111 ms

--- http ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.103/0.146/0.200 ms
$ docker service inspect http --format="{{.Endpoint.VirtualIPs}}"
[{h6yefpzuwl3zqbbgd4irxnypw 10.0.0.2/24}]
$ docker inspect --format="{{.NetworkSettings.Networks.eg1.IPAddress}}"
 $(docker ps | grep docker-http-server | head -n1 | awk '{print $1}')
10.0.0.4
$
```

```
$
```



Load Balance and Service Discover in Swarm Mode

Step 4 of 4

it to load balance the requests between the two containers.

```
docker service create --name
app1-web --network app1-
network --replicas 4 -p
80:3000 katacoda/redis-node-
docker-example ✓
```

Each host should have a Node.js container instance with one host storing Redis. `docker ps ✓`

Calling the HTTP server will store the request in Redis and return the results. This is load balanced, with two containers talking across the overlay network to the Redis container.

```
curl host01 ✓
```

The application is now distributed across multiple hosts.

```
-entrypoint.s…"   22 seconds ago      Up 20 seconds       6379/tcp
       redis.1.qlk9pqx3udhj9eaq3nn0a06te
0f656eb34359        katacoda/docker-http-server:latest          "/app"
                  2 minutes ago       Up 2 minutes        80/tcp
       http.2.xj3bilb2l76pk0ad1t1r7wuqg
87e28f0310a8        katacoda/docker-http-server:latest          "/app"
                  17 minutes ago      Up 17 minutes       80/tcp
       lbapp1.1.npekauel3p5nkvm01hfrznim1
$ curl host01
This page was generated after talking to redis.

Application Build: 1

Total requests: 1

IP count:
    ::ffff:10.255.0.2: 1
$
```

```
$
```