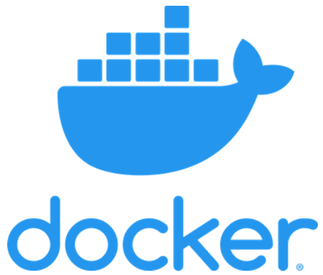
Docker Deep Dive

Chapter #8



Docker Swarm

Docker

SWARM

www.networknuts.net

DOCKER SWARM

Introduction to Docker Swarm

DEPLOYING SWARM SERVICE

Running application in swarm cluster

DOCKER SWARM

Enterprise grade secure cluster of docker hosts.

ORCHESTRATOR

Deploy & manage complicated microservices with ease.

MANAGING ENTERPRISE APPS

You can perform rolling updates, rollbacks, and scaling operations.

Docker Swarm

Docker swarm: deep dive

Docker Swarm is two things:

Enterprise-grade secure cluster of Docker hosts, and an engine for orchestrating microservices apps.

On the **clustering front**, it groups one or more Docker nodes and lets you manage them as a cluster. Out-of-the-box you get an encrypted distributed cluster store, encrypted networks, mutual TLS, secure cluster join tokens.

On the **orchestration front**, swarm exposes a rich API that allows you to deploy and manage complicated microservices apps with ease. You can define your apps in declarative manifest files, and deploy them with native Docker commands. You can even perform rolling updates, rollbacks, and scaling operations.

A swarm consists of one or more Docker nodes. These can be physical servers, VMs, Raspberry Pi's, or cloud instances.

Nodes are configured as **managers** or **workers**.

Managers look after the control plane of the cluster, meaning things like the state of the cluster and dispatching tasks to workers.

Workers accept tasks from managers and execute them.

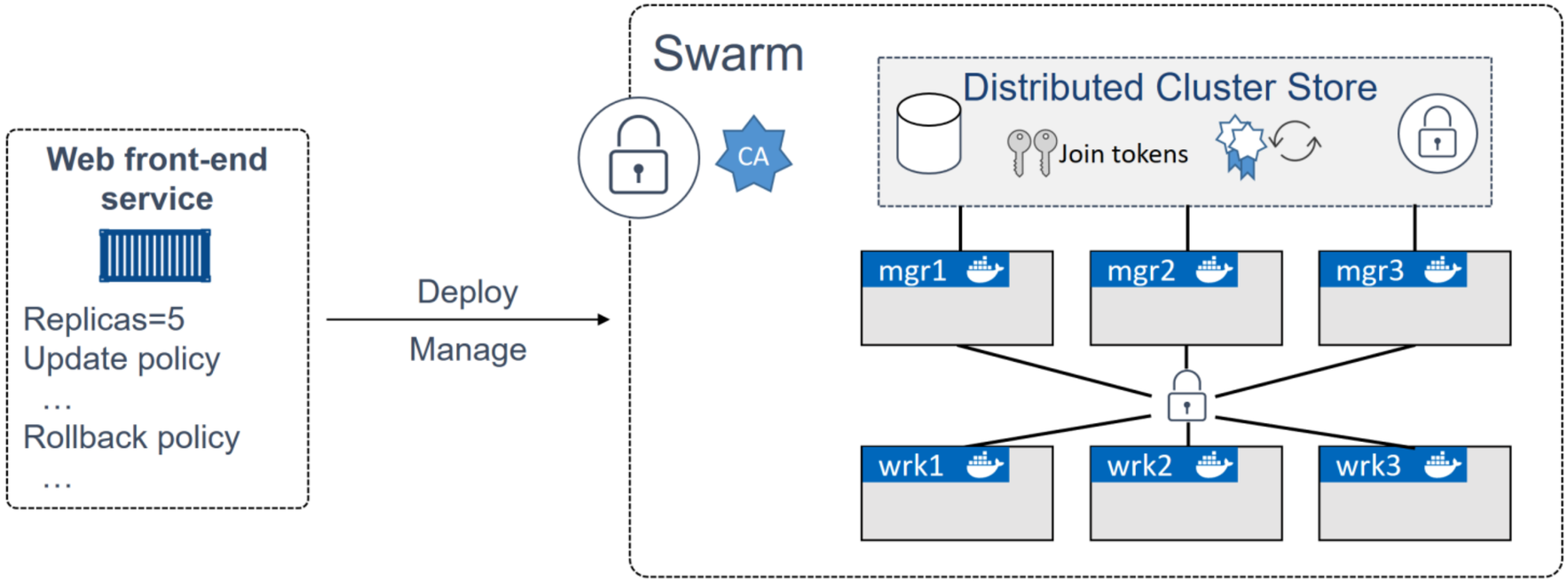
The configuration and state of the swarm is held in a distributed etcd database located on all managers. It's kept in memory and is extremely up-to-date.

The best thing about "etcd" is that it requires zero configuration — it’s installed as part of the swarm and just takes care of itself.

TLS is so tightly integrated that it’s impossible to build a swarm without it. Swarm uses TLS to encrypt communications, authenticate nodes, and authorize roles.

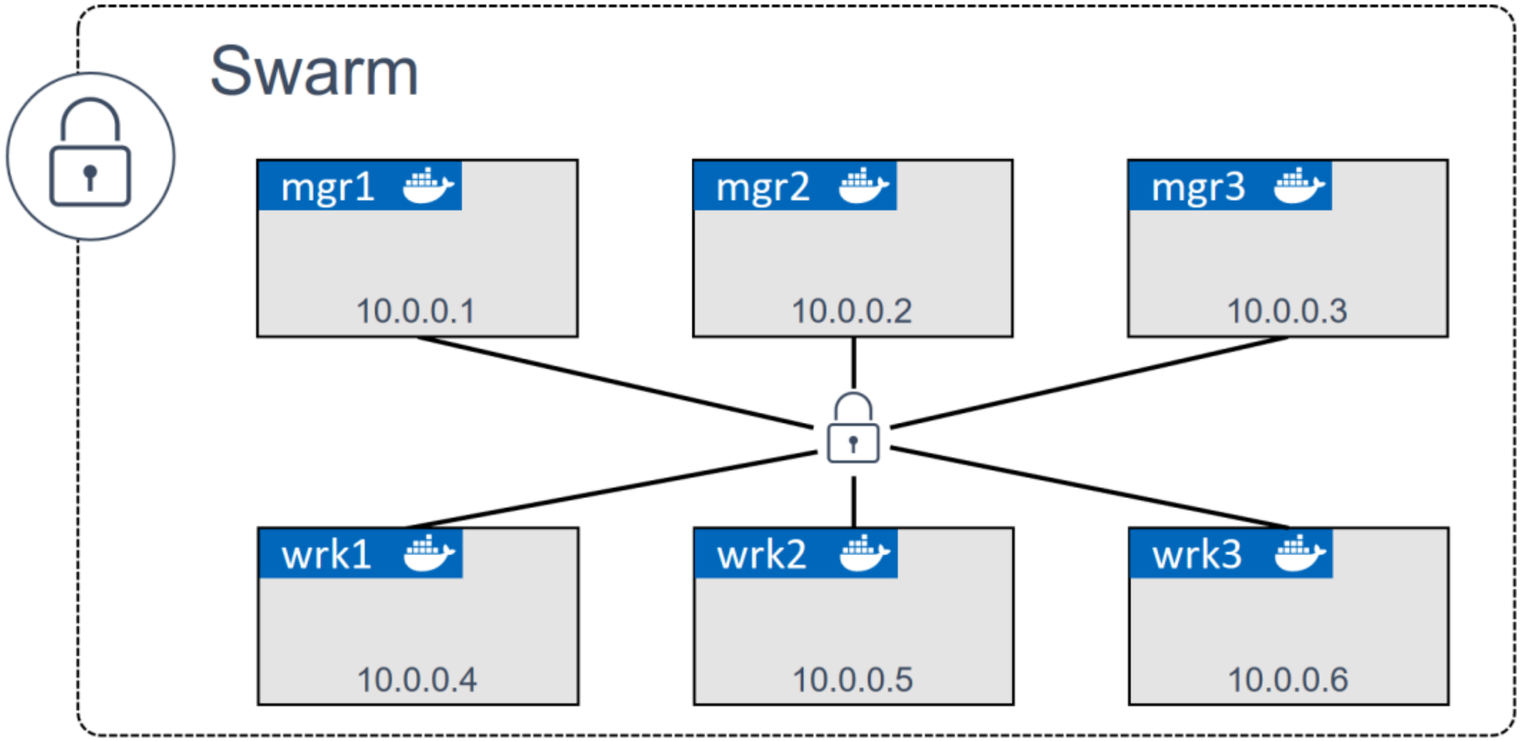
On the application orchestration front, the atomic unit of scheduling on a swarm is the service. This is a new object in the API, introduced along with swarm, and is a higher level construct that wraps some advanced features around containers.

When a container is wrapped in a service we call it a task or a replica, and the service construct adds things like scaling, rolling updates, and simple rollbacks.


Web front-end service
Replicas=5
Update policy
…
Rollback policy
…
Deploy
Manage
Swarm
CA
Distributed Cluster Store
Join tokens
mgr1
mgr2
mgr3
wrk1
wrk2
wrk3

Build a secure swarm cluster

We are going to build a secure swarm cluster with three manager nodes and three worker nodes. The IP's will be different as I will be building on AWS cloud.



Each of the nodes needs Docker installed and needs to be able to communicate with the rest of the swarm. It' also beneficial if name resolution is configured — it makes it easier to identify nodes in command outputs and helps when troubleshooting.

On the networking front, you should have the following ports open on routers & firewalls:

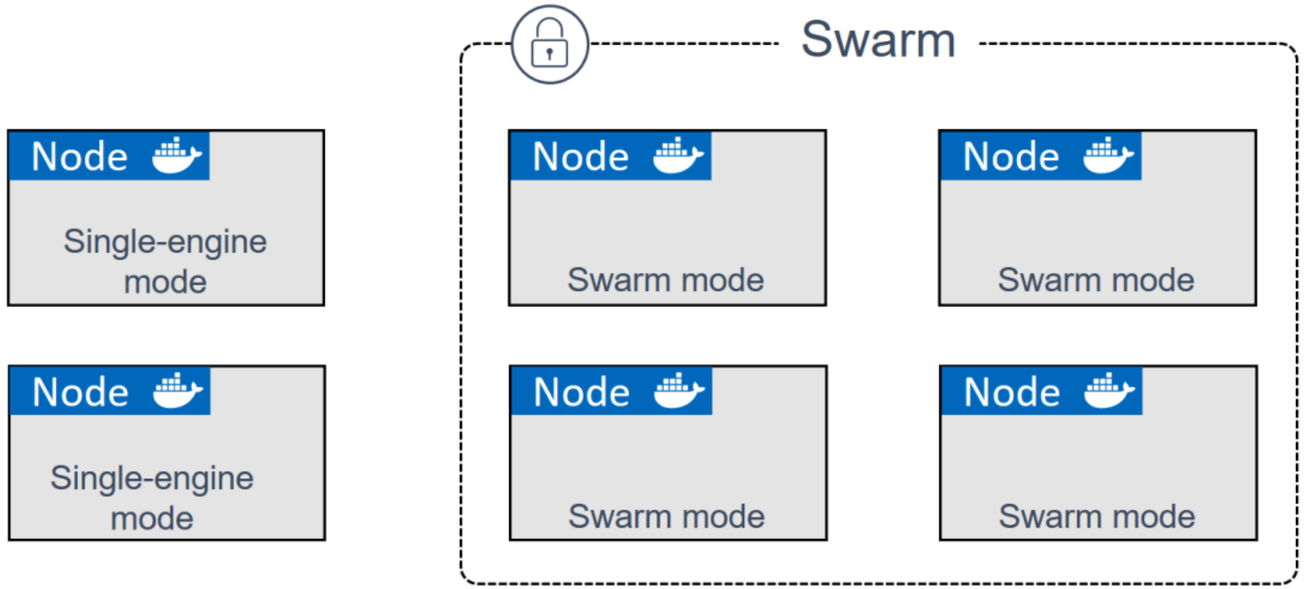
- 2377/tcp: for secure client-to-swarm communication
- 7946/tcp and 7946/udp: for control plane gossip
- 4789/udp: for VXLAN-based overlay networks

You can disable the firewall also. Not recommended in production.

The process of building a swarm is sometimes called initializing a swarm, and the high-level process is this:

Initialize the first manager node > Join additional manager nodes > Join worker nodes

Docker nodes that are not part of a swarm are said to be in single-engine mode. Once they're added to a swarm they're switched into swarm mode.



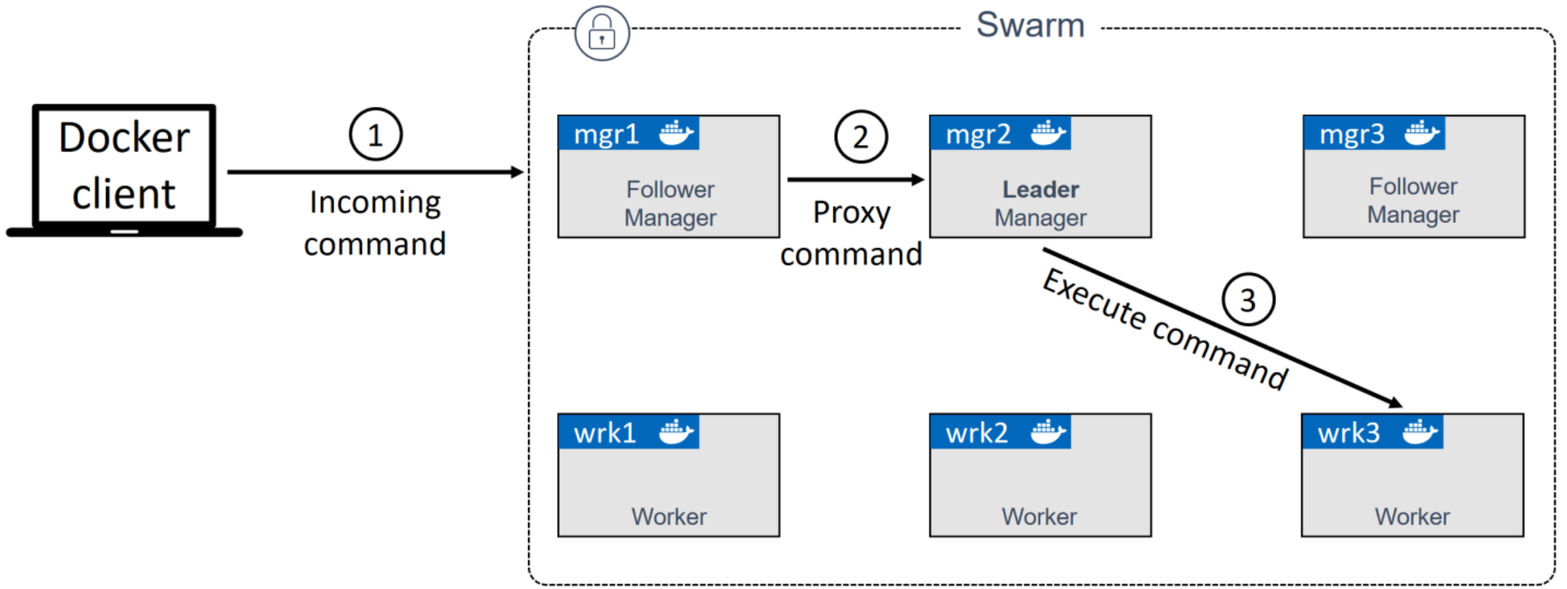
HA for Swarm Manager

We are using three manager nodes to a swarm. Why?

Swarm managers have native support for high availability (HA). This means one or more can fail, and the survivors will keep the swarm running.

Swarm implements a form of **active-passive** multi-manager HA. This means that although you might — and should — have multiple managers, only one of them is ever considered active. We call this active manager the **“leader”**, and the leader’s the only one that will ever issue live commands against the swarm.

So it’s only ever the leader that changes the config, or issues tasks to workers. If a passive (non-active) manager receives commands for the swarm, it proxies them to the leader.



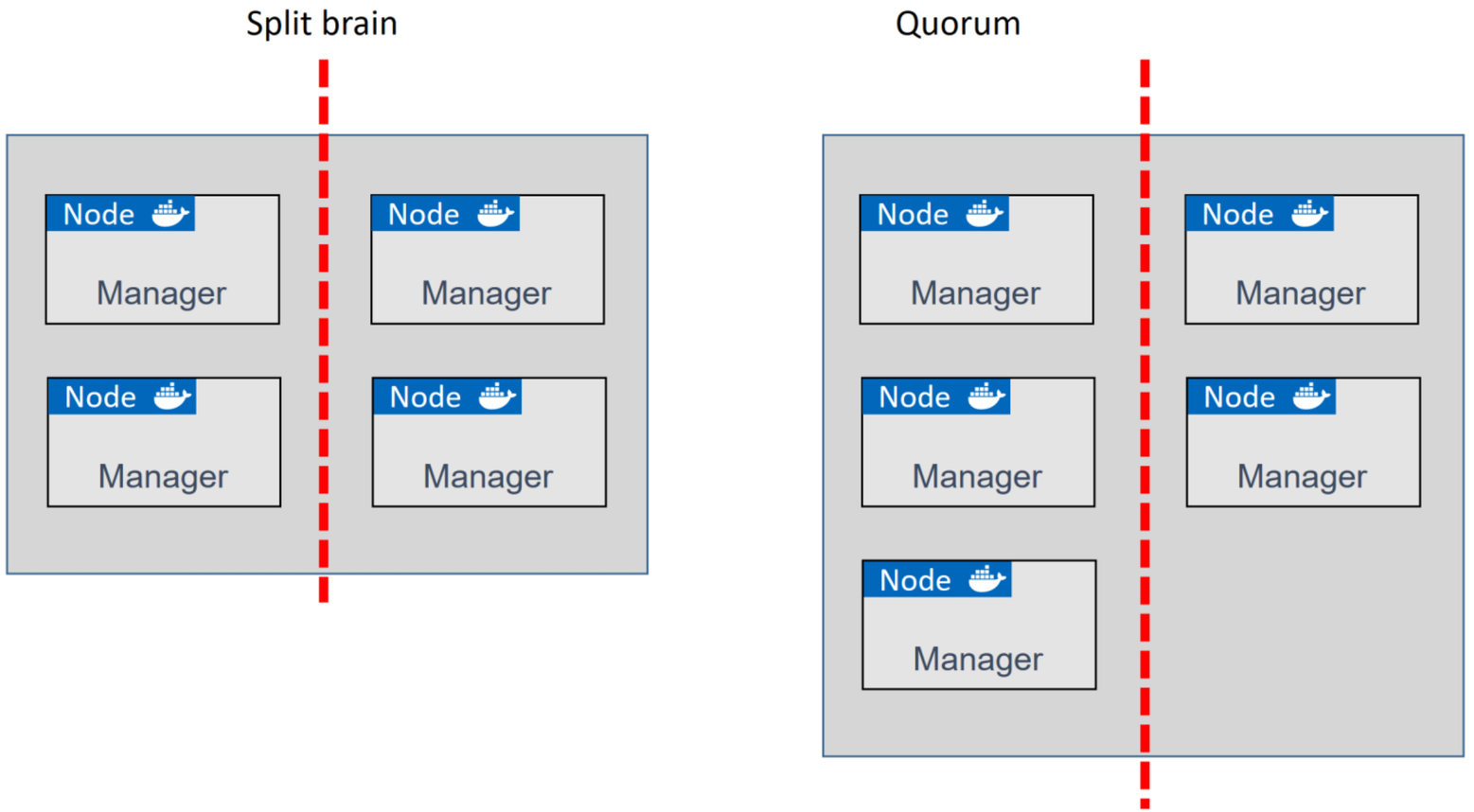
Managers are either leaders or followers. This is Raft terminology, because swarm uses an implementation of the **Raft consensus algorithm** to power manager HA. And on the topic of HA, the following two best practices apply:

1. Deploy an odd number of managers
2. Don't deploy too many managers (3 or 5 is recommended)

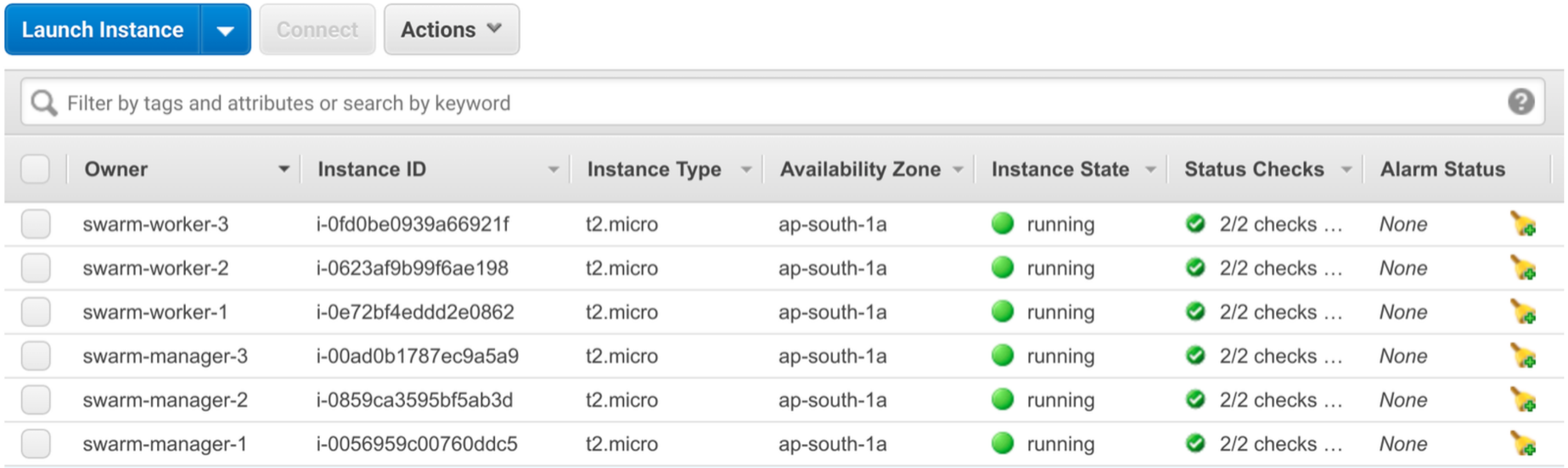
Having an odd number of managers reduces the chances of **split-brain** conditions.

If you had 4 managers and the network partitioned, you could be left with two managers on each side of the partition. This is known as a split brain — each side knows there used to be 4 but can now only see 2. But crucially, neither side has any way of knowing if the other two are still alive and whether it holds a majority (quorum). The cluster continues to operate during split-brain conditions, but you are no longer able to alter the configuration or add and manage application workloads.

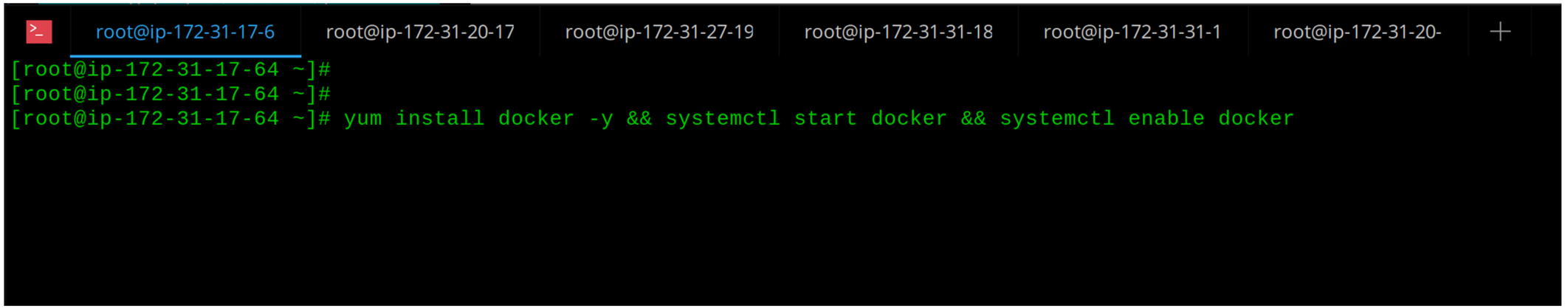
However, if you had 3 or 5 managers and the same network partition occurred, it would be impossible to have the same number of managers on both sides of the partition. This means that one side achieve quorum and cluster management would remain available.


Split brain
Node
Manager
Node
Manager
Node
Manager
Node
Manager
Quorum
Node
Manager
Node
Manager
Node
Manager
Node
Manager
Node
Manager

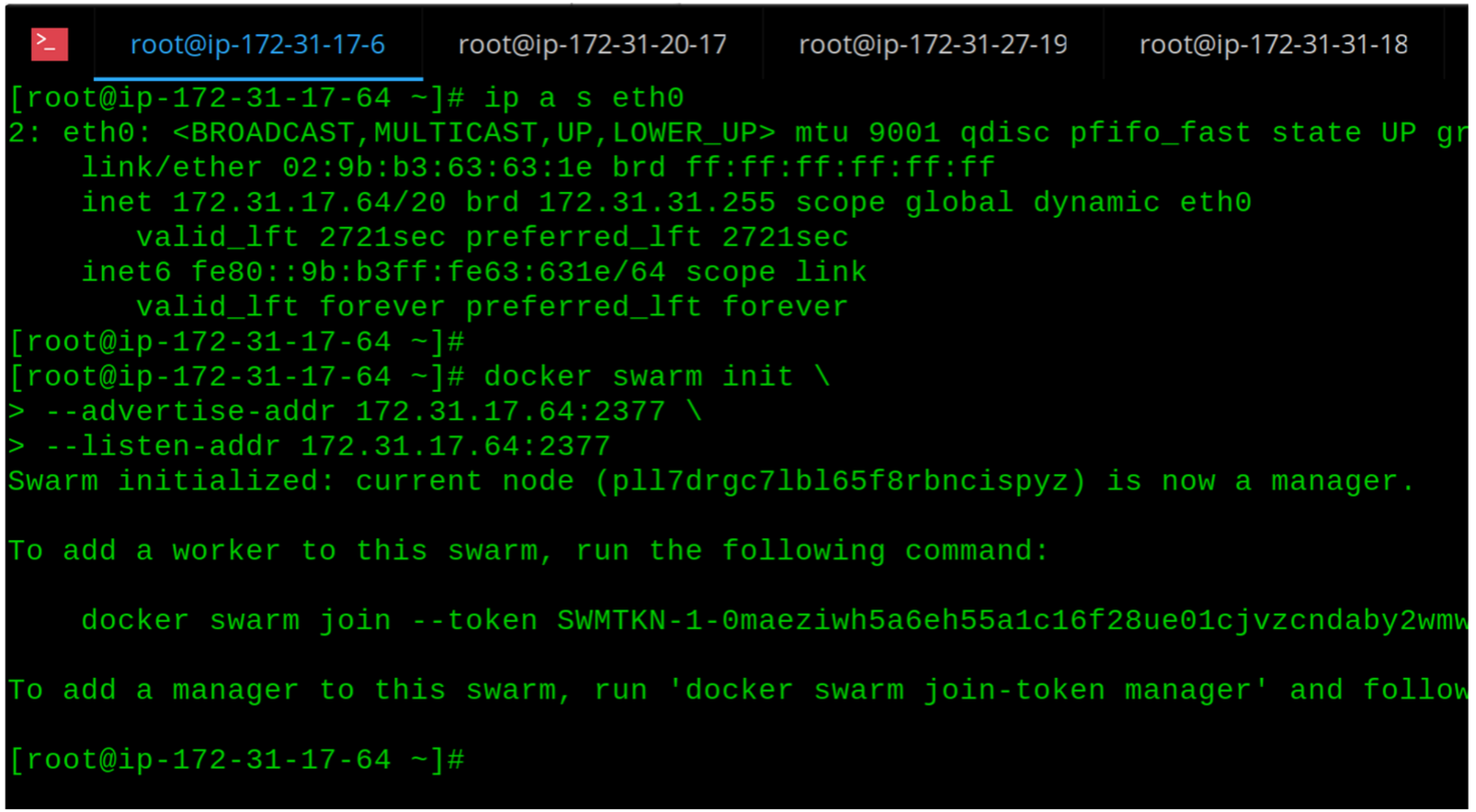
I have created six ec2 instances for this lab. You can create virtual machines, depending on how much compute power you have. The fastest way to do this is on cloud. All my six machines have same ssh-key to access them.



SSH into all the instances and install docker on all machines. Also start docker service on all machines.



On "swarm-manager-1" initialize a new swarm.



The command can be broken down as follows:

- **docker swarm init** tells Docker to initialize a new swarm and make this node the first manager. It also enables swarm mode on the node.
- **--advertise-addr** is the IP and port that other nodes should use to connect to this manager. It’s an optional flag, but it gives you control over which IP gets used on nodes with multiple IPs.
- **--listen-addr** lets you specify which IP and port you want to listen on for swarm traffic. This will usually match the --advertise-addr

Its recommended to use both flags.

List the nodes in the swarm.

docker node ls

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker node ls
ID                            HOSTNAME                                    STATUS              AVAILABILITY        MANAGER STATUS
   ENGINE VERSION
pll7drgc7lbl65f8rbncispyz *   ip-172-31-17-64.ap-south-1.compute.internal   Ready               Active              Leader
   19.03.6-ce
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]#
```

From the manager run the docker swarm join-token command to extract the commands and tokens required to add new workers and managers to the swarm.

 # docker swarm join-token manager

 # docker swarm join-token worker

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0maeziwh5a6eh55a1c16f28ue01cjvzcndab

[root@ip-172-31-17-64 ~]# docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0maeziwh5a6eh55a1c16f28ue01cjvzcndab

[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]#
```

Now copy and paste the respective commands on respective machines, based on their roles.

Check the status of swarm cluster using: docker node ls

```
root@ip-172-31-17-6   root@ip-172-31-20-17   root@ip-172-31-27-19   root@ip-172-31-31-18   root@ip-172-31-31-1   root@ip-172-31-20-
[root@ip-172-31-17-64 ~]# docker node ls
ID                            HOSTNAME                                        STATUS              AVAILABILITY        MANAGER STATUS
    ENGINE VERSION
pll7drgc7lbl65f8rbncispyz *   ip-172-31-17-64.ap-south-1.compute.internal     Ready               Active              Leader
    19.03.6-ce
j6p4dg39jprp81airhdcl8wgg     ip-172-31-20-9.ap-south-1.compute.internal      Ready               Active
    19.03.6-ce
stxofiyac688s6dm6gihqkb7s     ip-172-31-20-176.ap-south-1.compute.internal    Ready               Active              Reachable
    19.03.6-ce
q1vgm6fngeduoxlper2cpiv3z     ip-172-31-27-195.ap-south-1.compute.internal    Ready               Active              Reachable
    19.03.6-ce
1f9v4ckdk3fywt7k08mlud0xf     ip-172-31-31-10.ap-south-1.compute.internal     Ready               Active
    19.03.6-ce
5797a9xwwqg504i574b1aobm7     ip-172-31-31-185.ap-south-1.compute.internal    Ready               Active
    19.03.6-ce
[root@ip-172-31-17-64 ~]#
```

Swarm Service

Create a new service with the docker service create command.

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service create --name nn-web -p 80:80 --replicas 3 lovelearnlinux/webserver:v1
s16mu2fuaq5la293kog2gxxxm
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
[root@ip-172-31-17-64 ~]#
```

We used **docker service create** to tell Docker we are declaring a new service, and we used the --name flag to name it nn-web.

We told Docker to map port 80 on every node in the swarm to 80 inside of each service replica.

Next, we used the --replicas flag to tell Docker that there should always be 3 replicas of this service.

Finally, we told Docker which image to use for the replicas — it’s important to understand that all service replicas use the same image and config.

All services are constantly monitored by the swarm. Swarm runs a background reconciliation loop that constantly compares the actual state of the service to the desired state.

If the two states match, the world is a happy place and no further action is needed. If they don’t match, swarm takes actions so that they do. Put another way, the swarm is constantly making sure that actual state matches desired state.

Use the **docker service ls** command to check state of services.

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                             PORTS
s16mu2fuaq5l        nn-web              replicated          3/3                 lovelearnlinux/webserver:v1       *:80->80/tcp
[root@ip-172-31-17-64 ~]#
```

Use the docker service ps command to see a list of service replicas and the state of each replica.

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service ps nn-web
ID                  NAME                IMAGE                          NODE                                          DESIRED STATE
    CURRENT STATE           ERROR                  PORTS
lr6orv6gdoso        nn-web.1            lovelearnlinux/webserver:v1    ip-172-31-20-176.ap-south-1.compute.internal   Running
    Running 8 minutes ago
25cl2tbguflg        nn-web.2            lovelearnlinux/webserver:v1    ip-172-31-27-195.ap-south-1.compute.internal   Running
    Running 8 minutes ago
v75l2hk0fr8k        nn-web.3            lovelearnlinux/webserver:v1    ip-172-31-17-64.ap-south-1.compute.internal    Running
    Running 8 minutes ago
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]#
```

Point your browser to the IP address of any of the nodes in the swarm on port 80 to see the service running.

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# curl http://172.31.17.64
<html>
<title>
Network Nuts - Kubernetes training
</title>
<h1>
welcome to networknuts
</h1>
<h2>
you are learning kubernetes<br>
lets love learn linux
</h2>
</html>
```

For detailed information about a service, use the docker service inspect command.

```
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service inspect nn-web
[
    {
        "ID": "s16mu2fuaq5la293kog2gxxxm",
        "Version": {
            "Index": 55
        },
        "CreatedAt": "2020-06-09T16:48:03.328545308Z",
        "UpdatedAt": "2020-06-09T16:48:03.332148288Z",
        "Spec": {
            "Name": "nn-web",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "lovelearnlinux/webserver:v1@sha256:ac9ab88604c58
                    "Init": false,
                    "StopGracePeriod": 10000000000,
```

Replicated vs Global Service

The default replication mode of a service is replicated . This will deploy a desired number of replicas and distribute them as evenly as possible across the cluster.

The other mode is global , which runs a single replica on every node in the swarm. To deploy a global service you need to pass the --mode global flag to the docker service create command.

Scaling Service

Another powerful feature of services is the ability to easily scale them up and down.

```
[root@ip-172-31-17-64 ~]# docker service scale nn-web=5
nn-web scaled to 5
overall progress: 3 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: preparing [=================================>                 ]
5/5: preparing [=================================>                 ]
```

Scaling Service

Another powerful feature of services is the ability to easily scale them up and down.

```
[root@ip-172-31-17-64 ~]# docker service scale nn-web=5
nn-web scaled to 5
overall progress: 3 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: preparing [=================================>                 ]
5/5: preparing [=================================>                 ]
```

Use the docker service ls command to verify the operation was successful.

```
[root@ip-172-31-17-64 ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                          PORTS
s16mu2fuaq5l        nn-web              replicated          5/5                 lovelearnlinux/webserver:v1    *:80->80/tcp
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service ps nn-web
ID                  NAME                IMAGE                          NODE                                          DESIRED STATE
    CURRENT STATE               ERROR               PORTS
jutsnqt68lxw        nn-web.1            lovelearnlinux/webserver:v1    ip-172-31-17-64.ap-south-1.compute.internal   Running
    Running 15 minutes ago
r5xnzs6stw1z        nn-web.2            lovelearnlinux/webserver:v1    ip-172-31-20-176.ap-south-1.compute.internal  Running
    Running 15 minutes ago
hzpdhn8jmn2l        nn-web.3            lovelearnlinux/webserver:v1    ip-172-31-27-195.ap-south-1.compute.internal  Running
    Running 15 minutes ago
wcgctpzarp3q        nn-web.4            lovelearnlinux/webserver:v1    ip-172-31-31-10.ap-south-1.compute.internal   Running
    Running about a minute ago
9v95ohryc2ee        nn-web.5            lovelearnlinux/webserver:v1    ip-172-31-20-9.ap-south-1.compute.internal    Running
    Running about a minute ago
[root@ip-172-31-17-64 ~]#
```

Removing Service

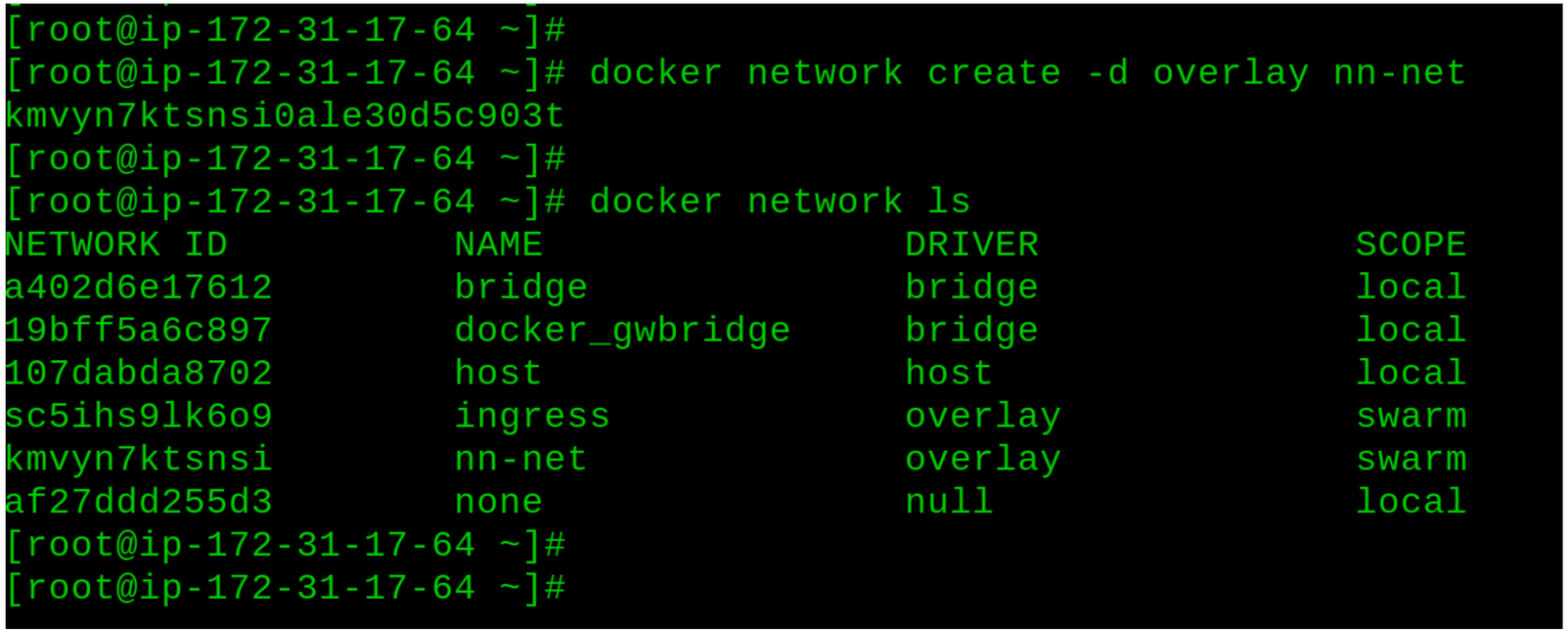
docker service rm command will delete the service deployed earlier.

```
[root@ip-172-31-17-64 ~]# docker service rm nn-web
nn-web
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service ls
ID                  NAME                MODE                REPLICAS
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]#
```

Rolling Updates

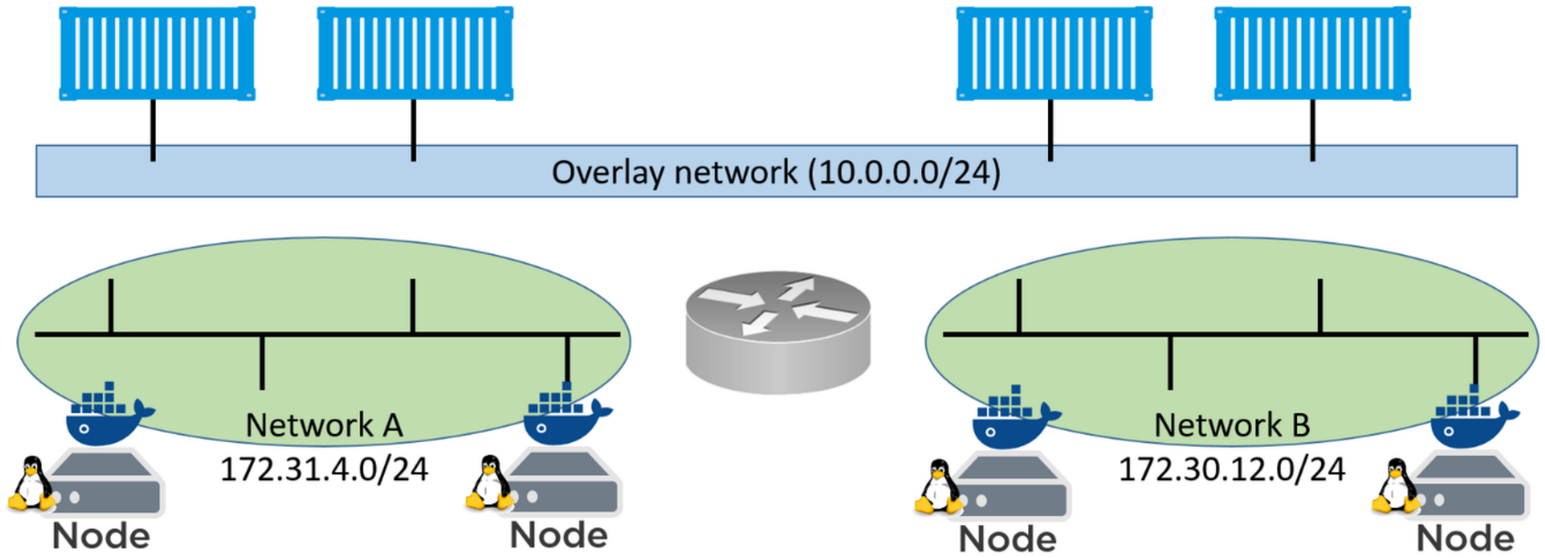
First we are creating a new overlay network called “nn-net” that we’ll be able to leverage with the service we’re about to create. An overlay network creates a new layer 2 network that we can place containers on, and all containers on it will be able to communicate.

This works even if the Docker hosts the containers are running on are on different underlying networks. Basically, the overlay network creates a new layer 2 container network on top of potentially multiple different underlying networks.



This creates a new overlay network called “nn-net” that we’ll be able to use with the service we’re about to create. An overlay network creates a new layer 2 network that we can place containers on, and all containers on it will be able to communicate.

Docker hosts are connected to the two underlay networks and containers are connected to the overlay. All containers on the overlay can communicate even if they are on Docker hosts plumbed into different underlay networks.



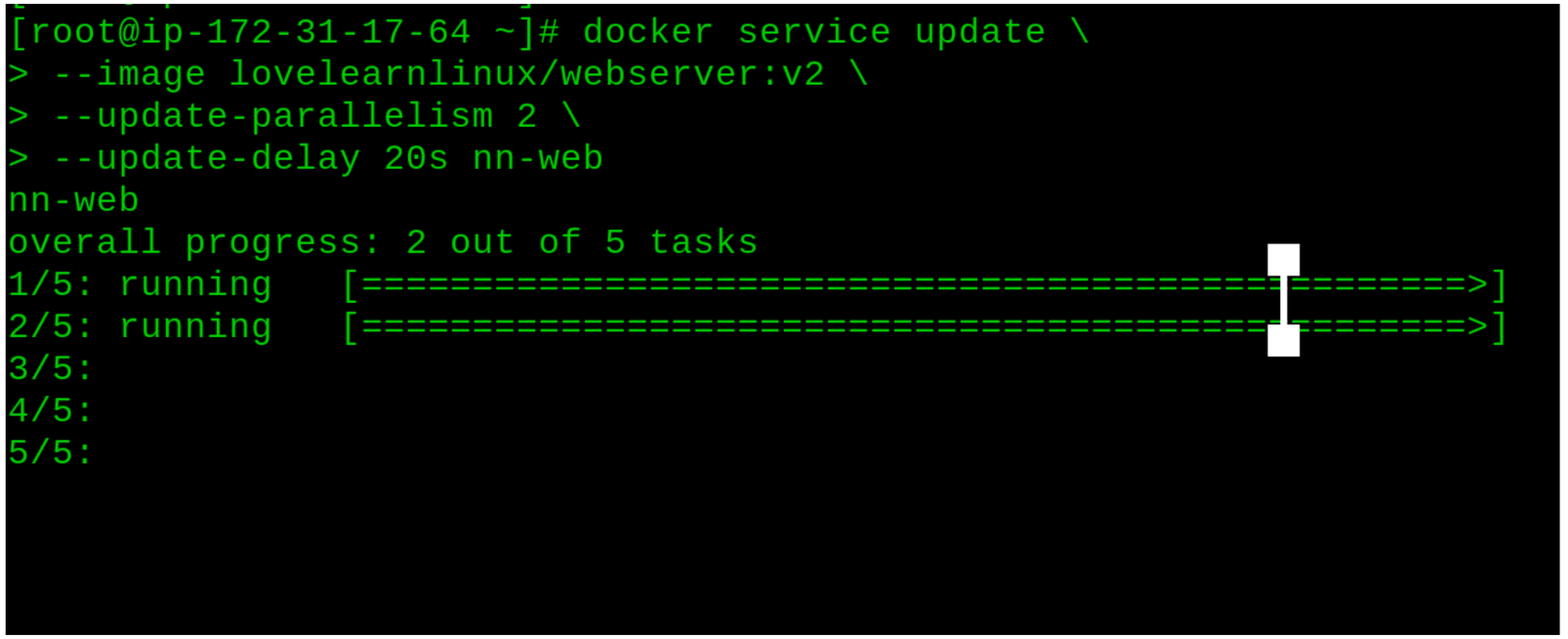
Now create a new service and attach it to the network.

```
[root@ip-172-31-17-64 ~]# docker service create --name nn-web \
> --network nn-net -p 80:80 \
> --replicas 5 \
> lovelearnlinux/webserver:v1
m3eepeq7178n2fuexu0fzg9xt
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
[root@ip-172-31-17-64 ~]#
```

Run a **docker service ls** and a **docker service ps** command to verify the state of the new service.

```
[root@ip-172-31-17-64 ~]# docker service ls
ID                  NAME                MODE                REPLICAS            IMAGE                             PORTS
m3eepeq7178n        nn-web              replicated          5/5                 lovelearnlinux/webserver:v1       *:80->80/tcp
[root@ip-172-31-17-64 ~]#
[root@ip-172-31-17-64 ~]# docker service ps nn-web
ID                  NAME                IMAGE                             NODE                                                DESIRED STATE
    CURRENT STATE               ERROR               PORTS
oz589276jqys        nn-web.1            lovelearnlinux/webserver:v1       ip-172-31-20-176.ap-south-1.compute.internal        Running
    Running about a minute ago
vbrma7iqlcfw        nn-web.2            lovelearnlinux/webserver:v1       ip-172-31-27-195.ap-south-1.compute.internal        Running
    Running about a minute ago
jt2bnxe5lkdg        nn-web.3            lovelearnlinux/webserver:v1       ip-172-31-31-185.ap-south-1.compute.internal        Running
    Running about a minute ago
fcdj2rfchnsf        nn-web.4            lovelearnlinux/webserver:v1       ip-172-31-31-10.ap-south-1.compute.internal         Running
    Running about a minute ago
92ykd7lll9t1        nn-web.5            lovelearnlinux/webserver:v1       ip-172-31-17-64.ap-south-1.compute.internal         Running
    Running about a minute ago
[root@ip-172-31-17-64 ~]#
```

Now you have to push a updated version of application. You have to push it into swarm in a staged manner — 2 replicas at a time with a 20 second delay between each. We can use the following **docker service update** command.



If we run a docker service ps against the service we'll see that some of the replicas are at v2 while some are still at v1 .

```
[root@ip-172-31-17-64 ~]# docker service ps nn-web
ID                  NAME                IMAGE                          NODE                                                DESIRED STATE
    CURRENT STATE                ERROR               PORTS
9vpznjutqndj        nn-web.1            lovelearnlinux/webserver:v2    ip-172-31-31-10.ap-south-1.compute.internal    Running
    Running 11 seconds ago
oz589276jqys         \_ nn-web.1        lovelearnlinux/webserver:v1    ip-172-31-20-176.ap-south-1.compute.internal   Shutdown
    Shutdown 13 seconds ago
k8uxksei6cu1        nn-web.2            lovelearnlinux/webserver:v2    ip-172-31-20-9.ap-south-1.compute.internal     Running
    Running about a minute ago
vbrma7iqlcfw         \_ nn-web.2        lovelearnlinux/webserver:v1    ip-172-31-27-195.ap-south-1.compute.internal   Shutdown
    Shutdown about a minute ago
44ycvywz1uo5        nn-web.3            lovelearnlinux/webserver:v2    ip-172-31-31-185.ap-south-1.compute.internal   Running
    Running 37 seconds ago
jt2bnxe5lkdg         \_ nn-web.3        lovelearnlinux/webserver:v1    ip-172-31-31-185.ap-south-1.compute.internal   Shutdown
    Shutdown 38 seconds ago
iay042y1xalh        nn-web.4            lovelearnlinux/webserver:v2    ip-172-31-27-195.ap-south-1.compute.internal   Running
    Running 37 seconds ago
fcdj2rfchnsf         \_ nn-web.4        lovelearnlinux/webserver:v1    ip-172-31-31-10.ap-south-1.compute.internal    Shutdown
    Shutdown 39 seconds ago
ewh9z6rrfkjp        nn-web.5            lovelearnlinux/webserver:v2    ip-172-31-17-64.ap-south-1.compute.internal    Running
    Running about a minute ago
92ykd7lll9t1         \_ nn-web.5        lovelearnlinux/webserver:v1    ip-172-31-17-64.ap-south-1.compute.internal    Shutdown
    Shutdown about a minute ago
[root@ip-172-31-17-64 ~]#
```

Point your browser to node IP. You will get the new application.

```
[root@ip-172-31-17-64 ~]# curl http://172.31.17.64
<html>
<title>
Network Nuts - Kubernetes training
</title>
<h1>
welcome to networknuts
</h1>
<h2>
you are learning kubernetes<br>
lets love learn linux
<h3>
VERSION 2
</h3>
</h2>
</html>

[root@ip-172-31-17-64 ~]#
```

WHAT WE LEARNED

- Swarm has a secure clustering component, and an orchestration component.
- The secure clustering component is enterprise-grade and offers a wealth of security and HA features that are automatically configured and extremely simple to modify.
- The orchestration component allows you to deploy and manage microservices applications in a simple declarative manner. Native Docker Swarm apps are supported, and so are Kubernetes apps.