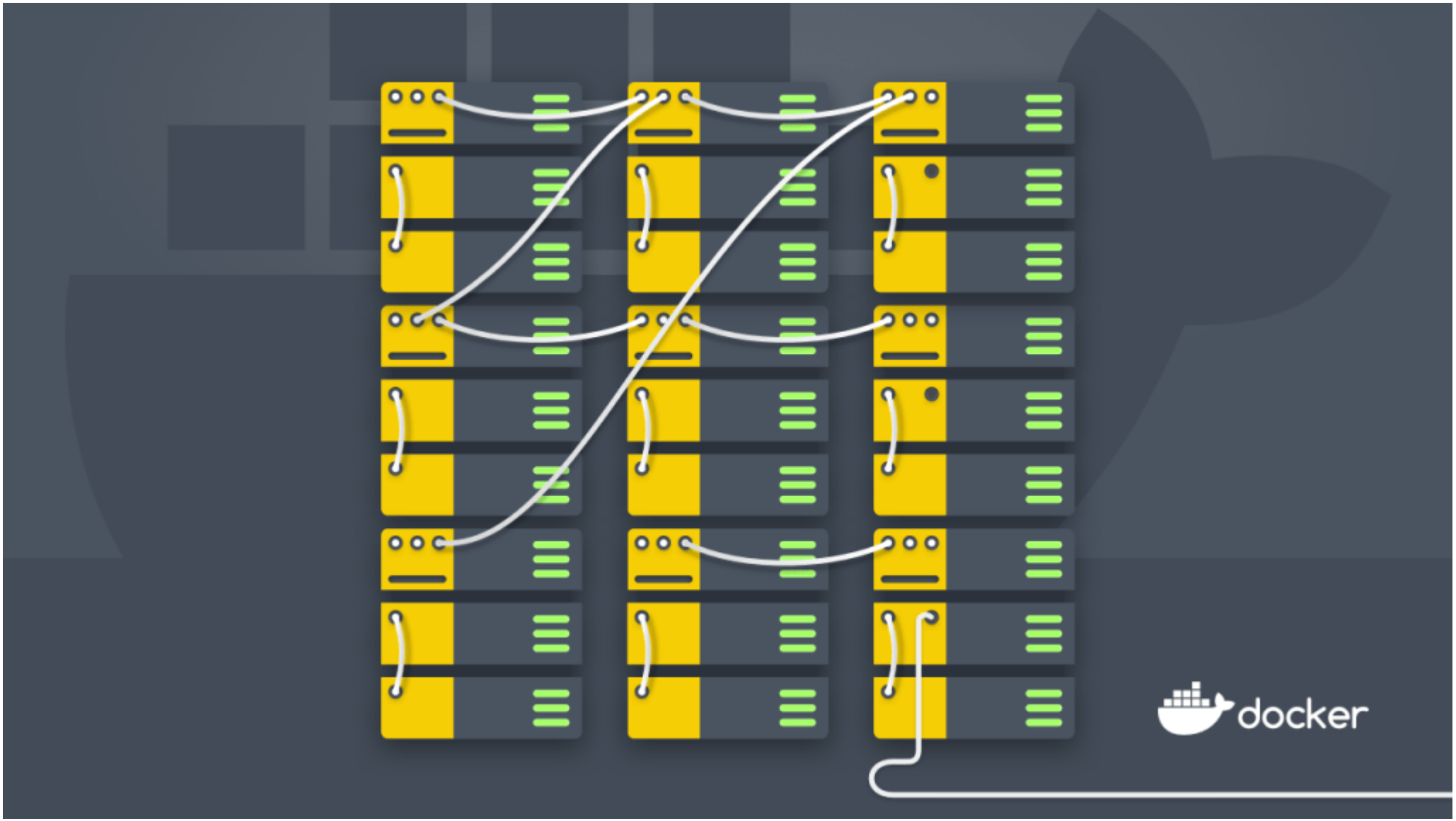


Summary

In six steps, you'll have a cluster up and running:

- Install Docker
- Assign 3 managers for the Docker Swarm
- Add two nodes or 'workers' to the Swarm
- Discover how to view your Swarm status
- Run a test service across all machines
  Anybody that is already familiar with Docker knows the limitations of running standalone containers in production. Containers are ephemeral and without proper scheduling and orchestration they may fail or stop running. Docker Swarm solves many of these challenges by introducing powerful container orchestration that allows services to be scheduled across multiple hosts. Services can be scheduled across multiple hosts in the Swarm and each service may run multiple containers, ensuring maximum reliability and performance.

Prerequisites

For the sake of this article, you'll need to have a few things already in place:

- A network of connected machines (i.e. VPC)
- At least 2 public subnets
- 5 EC2 instances on AWS with an elastic load balancer (ELB
  Set these ingress rules on your EC2 security groups:

- HTTP port 80 from 0.0.0.0\0
- SSH from 0.0.0.0\0 (for increased security replace this with your own IP)
  Now that you've got your machines configured properly, let's get started. We're using Ubuntu 14.04 LTS, though the process will work pretty much the same on most Linux-based operating systems.

Step 1: Installing Docker

The installation process is well documented elsewhere and you can read Docker's documentation if you'd like the play by play. The simplest method is to run the following commands on each machine:

```
sudo apt-get update
sudo apt-get install curl -y
```

```
curl -sSL https://get.docker.com/ | sh
```

This will install the latest version of Docker engine.

Step 2: Assign First Manager Machine

To begin, all nodes must be able to access your swarm manager machine. To get the internal IP address of your machines you can look for the Private IP as listed on the EC2 dashboard or issue this command from your SSH console (EC2 metadata docs here):

`curl http://169.254.169.254/latest/meta-data/local-hostname`

This will respond with something like this: `ip-10-0-2-194.ec2.internal`

Next, use the following command to assign your first host machine as manager:

(***Note:*** replace everything bracketed <> with your own info)

`docker swarm init --advertise-addr <manager_ip>`

Once issued and completed, the following output should appear on your screen:

```
Swarm initialized: current node (uyz40wx3xg0fmr4i472lhn0vt) is now a manager.
```

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

```
docker swarm join \
--token SWMTKN-1-3jmkpf9b48bvzv9eyrxghw44l44fdqca1dgfbsjky1qz8cqvkp-2ab4shwgwv5gepyuz2wcns47n \

10.0.2.194:2377
```

You can see that we've already been given the command to add a worker to the cluster, but we'll save that for step 4.

As a quick review, you can issue the following commands at any time to get the tokens necessary for adding workers and managers.

To get manager token:

```
docker swarm join-token manager
```

To get worker token:

```
docker swarm join-token worker
```

Step 3: Add Manager Nodes

At this point, it is important to set up two more managers to improve your Swarm's fault-tolerance. Always create clusters with an odd number of managers in your Swarm, as a 'majority' vote is needed between managers to agree on proposed management tasks. An

odd—rather than even—number is strongly recommended to have a tie-breaking consensus. *Having two managers is actually worse than having one.*

We're going to setup 3 managers to establish our High Availability setup.

To get the manager token run `docker swarm join-token manager`.
To add your two additional manager nodes, issue this command from the nodes you wish to assign as managers:

```
docker swarm join --token <manager_token> <manager_ip>:2377
```

Example:

```
docker swarm join \
--token SWMTKN-1-3jmkpf9b48bvzv9eyrxghw44l44fdqca1dgfbsjky1qz8cqvkp-5ti6s2ofrfa6rhijpla0ngtok \

10.0.2.194:2377
```

You'll see a message like this:

```
This node joined a swarm as a manager.
```

Step 4: Add Worker Nodes

Now that you have your manager nodes assigned, you're ready to add some workers to the Swarm. From each of the nodes that you want to connect, run this command:

```
docker swarm join --token <worker_token> <manager_ip>:2377
```

Repeat this action to add as many worker nodes as you want to the cluster.

Step 5: View Swarm Status

Let's take a look at all that you have created in your first cluster. Run command `docker node ls` from a manager machine to view your Swarm's connected nodes.
You can see we now have 3 managers and 2 workers.

```
jp@ip-10-0-2-194:~$ docker node ls
ID                            HOSTNAME        STATUS  AVAILABILITY  MANAGER STATUS
2p55nujq25a1xvktdpi8isojm     ip-10-0-2-87    Ready   Active        Reachable
ln85jiyan50c76agctsn6k9fl     ip-10-0-2-175   Ready   Active
m44w85ytt2h06zdbj2i9arzu6     ip-10-0-2-150   Ready   Active        Reachable
uyz40wx3xg0fmr4i472lhn0vt *   ip-10-0-2-194   Ready   Active        Leader
zax54w8wrad3kdpu7r5tytr6l     ip-10-0-2-141   Ready   Active
jp@ip-10-0-2-194:~$
```

Step 6: Create a Service

With our Swarm is up and running, let's get a service deployed to see how the scheduling works. To start a service on the Swarm go back to any manager machine.

Let's start an nginx service, called webserver, with the command:

```
docker service create -p 80:80 --name webserver nginx
```

Docker will now pull the latest nginx image and start one container.

We can see our service by issuing a docker service ls command, which will return something like this:

```
jp@ip-10-0-2-194:~$ docker service ls
ID            NAME       MODE        REPLICAS  IMAGE
```

```
nogqyhc73mfa  webserver  replicated  1/1       nginx:latest
```

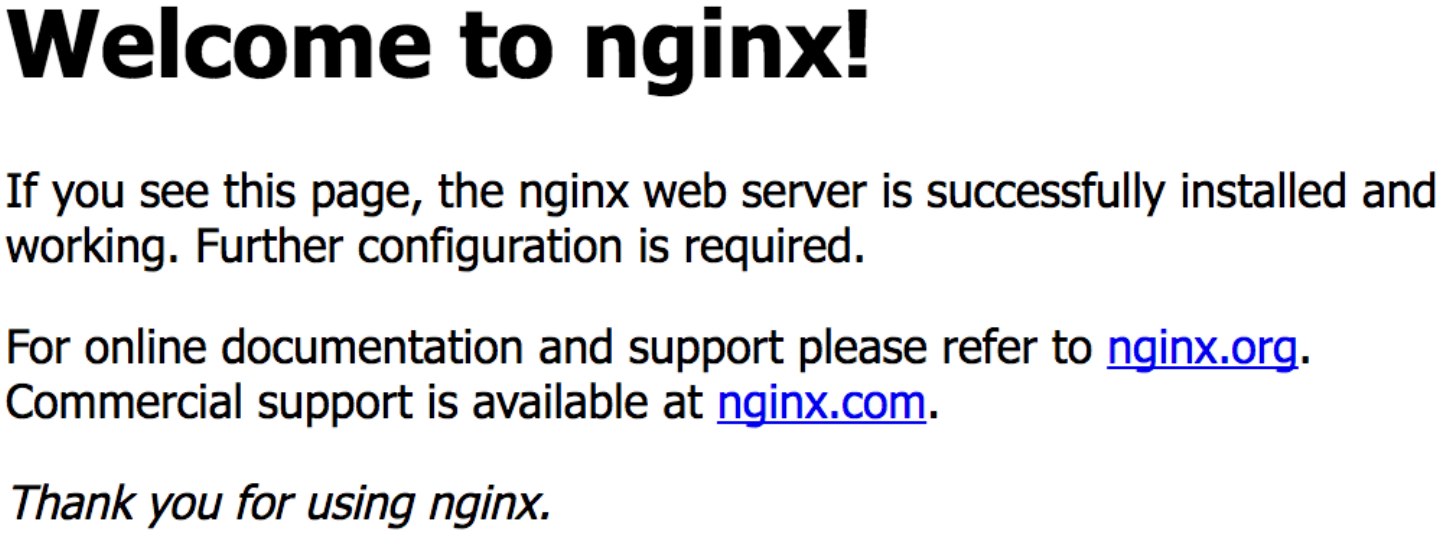
This will deploy the nginx service on one machine, but since Swarm is all about running containers on multiple nodes let's scale it up!

`docker service scale webserver=10`

Next, issue the command `docker service ps webserver` and you will see the instances of the service running.

```
jp@ip-10-0-2-194:~$ docker service ps webserver
ID            NAME          IMAGE         NODE          DESIRED STATE  CURRENT STATE           ERROR  PORTS
ramnvxel444p  webserver.1   nginx:latest  ip-10-0-2-141  Running        Running 52 seconds ago
t6w0l9e1pw57  webserver.2   nginx:latest  ip-10-0-2-141  Running        Running 52 seconds ago
srrnqgpk0lh1  webserver.3   nginx:latest  ip-10-0-2-175  Running        Running 51 seconds ago
8r6oga1fhzci  webserver.4   nginx:latest  ip-10-0-2-175  Running        Running 51 seconds ago
tcx6cd47pmyo  webserver.5   nginx:latest  ip-10-0-2-194  Running        Running 51 seconds ago
hhr0avw18jbd  webserver.6   nginx:latest  ip-10-0-2-87   Running        Running 54 seconds ago
m2bruzwv0s6z  webserver.7   nginx:latest  ip-10-0-2-150  Running        Running 54 seconds ago
7915sfp3h52r  webserver.8   nginx:latest  ip-10-0-2-194  Running        Running 51 seconds ago
ruvj42pfeauw  webserver.9   nginx:latest  ip-10-0-2-87   Running        Running 54 seconds ago
m6pwrajqlknb  webserver.10  nginx:latest  ip-10-0-2-150  Running        Running 54 seconds ago
jp@ip-10-0-2-194:~$
```

Since this service was published on port 80, you can fire up a web browser and point it to any public IP on the cluster and check out the nginx default page.



Optional: Removing the Service

Tearing down the nginx service you just deployed is extremely easy.

```
docker service rm webserver
```