**Docker swarm:**

The first thing to do is initialize the Swarm. We will SSH into the node1 machine and initialize the swarm in there.

[node1] (local) root@192.168.0.48 ~

$ ip r

default via 172.18.0.1 dev eth1

172.17.0.0/16 dev docker0 scope link src 172.17.0.1

172.18.0.0/16 dev eth1 scope link src 172.18.0.37

192.168.0.0/23 dev eth0 scope link src 192.168.0.48

[node1] (local) root@192.168.0.48 ~

$ docker swarm init --advertise-addr 192.168.0.48

Swarm initialized: current node (w1ti8n29zbu4okyg4lnhbi6qb) is now a manager.

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

docker swarm join --token SWMTKN-1-2mu7z4mcqwte82h3i01lawa8rw42mtou3yph4k5xnvghte9mu1-49b2h87lttz0vi9clew9gygqk 192.168.0.48:2377

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

-------------------------------------------------------------------------------------------------------------

**Joining as Worker Node**

To find out what docker swarm command to use to join as a node, We will need to use the join-token <role> command.

To find out the join command for a worker, fire the following command:

[node1] (local) root@192.168.0.48 ~

docker swarm join-token worker

docker swarm join --token SWMTKN-1-2mu7z4mcqwte82h3i01lawa8rw42mtou3yph4k5xnvghte9mu1-49b2h87lttz0vi9clew9gygqk 192.168.0.48:2377

On Workers:

[node2] (local) root@192.168.0.47 ~

$ docker swarm join --token SWMTKN-1-2mu7z4mcqwte82h3i01lawa8rw42mtou3yph4k5xnvghte9mu1-49b2h87lttz0vi9clew9gygqk 192.168.0.48:2377

This node joined a swarm as a worker.

[node3] (local) root@192.168.0.46 ~

$ docker swarm join --token SWMTKN-1-2mu7z4mcqwte82h3i01lawa8rw42mtou3yph4k5xnvghte9mu1-49b2h87lttz0vi9clew9gygqk 192.168.0.48:2377

This node joined a swarm as a worker.

[node4] (local) root@192.168.0.45 ~

$ docker swarm join --token SWMTKN-1-2mu7z4mcqwte82h3i01lawa8rw42mtou3yph4k5xnvghte9mu1-49b2h87lttz0vi9clew9gygqk 192.168.0.48:2377

This node joined a swarm as a worker.

------------------------------------------------------------------------------------------------------------

Keep in mind that We can have a node join as a worker or as a manager. At any point in time, there is only one LEADER and the other manager nodes will be as backup in case the current LEADER opts out.

$ [node1] (local) root@192.168.0.48 ~

$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS ENGINE VERSION

w1ti8n29zbu4okyg4lnhbi6qb \* node1 Ready Active Leader 19.03.4

hl4e8ypdj30ipn9dgokhtfq51 node2 Ready Active 19.03.4

n6gg9dcytsf6p7o3zdyajz9ei node3 Ready Active 19.03.4

k1urn3eqav6274lvg514qq0nf node4 Ready Active 19.03.4

**Create a Service**

Now that we have our swarm up and running, it is time to schedule our containers on it. This is the whole beauty of the orchestration layer. We are going to focus on the app and not worry about where the application is going to run.

All we are going to do is tell the manager to run the containers for us and it will take care of scheduling out the containers, sending the commands to the nodes and distributing it.

To start a service, We would need to have the following:

What is the Docker image that We want to run. In our case, we will run the standard nginx image that is officially available from the Docker hub.

We will expose our service on port 80.

We can specify the number of containers (or instances) to launch. This is specified via the replicas parameter.

We will decide on the name for our service. And keep that handy.

What I am going to do then is to launch 2 replicas of the nginx container. To do that, I am again in the SSH session for my node1. And I give the following docker service create command:

[node1] (local) root@192.168.0.48 ~

$ docker service create --replicas 2 -p 80:80 --name web nginx

wxy2lg28lgia889uhqlv8dml8

overall progress: 2 out of 2 tasks

1/2: running [==================================================>]

2/2: running [==================================================>]

verify: Service converged

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

rj6z01nxhfxy web.1 nginx:latest node2 Running Running 11 seconds ago

ltla907oyhyd web.2 nginx:latest node1 Running Running 11 seconds ago

[node1] (local) root@192.168.0.48 ~

$ docker service ls

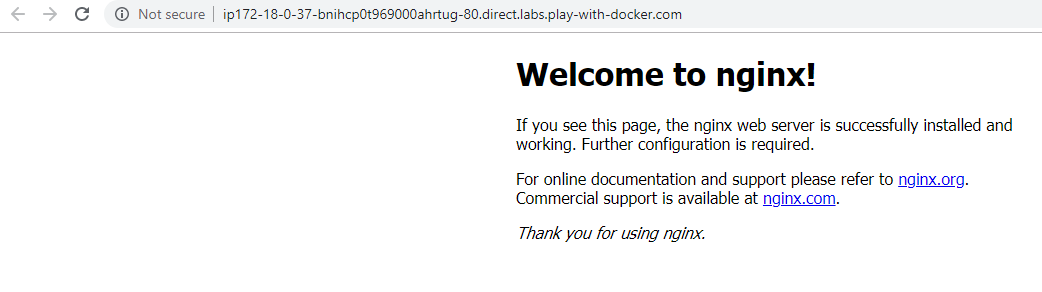
ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 2/2 nginx:latest \*:80->80/tcp

**Accessing the Service**

We can access the service by hitting any of the manager or worker nodes. It does not matter if the particular node does not have a container scheduled on it. That is the whole idea of the swarm.

Try out a curl to any of the Docker Machine IPs (manager1 or worker1/2/3/4/5) or hit the URL (http://<machine-ip>) in the browser. We should be able to get the standard NGINX Home page.



**Scaling up and Scaling down**

Here I increased the number of replica to 3. So, it will check if there is any free worker available reporting to leader. If yes, then it will host the service on the new worker. If no, then it will host another instance of the service on the existing nodes where the service is already up.

[node1] (local) root@192.168.0.48 ~

$ docker service update web --replicas 3

web

overall progress: 3 out of 3 tasks

1/3: running [==================================================>]

2/3: running [==================================================>]

3/3: running [==================================================>]

verify: Service converged

[node1] (local) root@192.168.0.48 ~

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 3/3 nginx:latest \*:80->80/tcp

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

rj6z01nxhfxy web.1 nginx:latest node2 Running Running 2 minutes ago

ltla907oyhyd web.2 nginx:latest node1 Running Running 2 minutes ago

vql6z4y6ryr5 web.3 nginx:latest node3 Running Running 19 seconds ago

[node1] (local) root@192.168.0.48 ~

$ docker service scale web=4 -- **This command will also scale up the service.**

web scaled to 4

overall progress: 4 out of 4 tasks

1/4: running [==================================================>]

2/4: running [==================================================>]

3/4: running [==================================================>]

4/4: running [==================================================>]

verify: Service converged

[node1] (local) root@192.168.0.48 ~

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 4/4 nginx:latest \*:80->80/tcp

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

rj6z01nxhfxy web.1 nginx:latest node2 Running Running 3 minutes ago

ltla907oyhyd web.2 nginx:latest node1 Running Running 3 minutes ago

vql6z4y6ryr5 web.3 nginx:latest node3 Running Running about a minute ago

m9mhoz07vcts web.4 nginx:latest node4 Running Running 13 seconds ago

[node1] (local) root@192.168.0.48 ~

[node1] (local) root@192.168.0.48 ~

$ docker service scale web=8 **-- Further scaling up to 8 instances of the service.**

web scaled to 8

overall progress: 8 out of 8 tasks

1/8: running [==================================================>]

2/8: running [==================================================>]

3/8: running [==================================================>]

4/8: running [==================================================>]

5/8: running [==================================================>]

6/8: running [==================================================>]

7/8: running [==================================================>]

8/8: running [==================================================>]

verify: Service converged

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

rj6z01nxhfxy web.1 nginx:latest node2 Running Running 7 minutes ago

ltla907oyhyd web.2 nginx:latest node1 Running Running 7 minutes ago

vql6z4y6ryr5 web.3 nginx:latest node3 Running Running 5 minutes ago

m9mhoz07vcts web.4 nginx:latest node4 Running Running 4 minutes ago

fj48c9k8ylxp web.5 nginx:latest node2 Running Running 55 seconds ago

s6yjswf31ab7 web.6 nginx:latest node3 Running Running 55 seconds ago

8nkb2f5ao6cd web.7 nginx:latest node4 Running Running 54 seconds ago

qy65i1jgndqh web.8 nginx:latest node1 Running Running 54 seconds ago

[node1] (local) root@192.168.0.48 ~

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 8/8 nginx:latest \*:80->80/tcp

[node1] (local) root@192.168.0.48 ~

$ docker service scale web=5 --- **Scaling down the service from 8 to 5 instances**.

web scaled to 5

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

[node1] (local) root@192.168.0.48 ~

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 5/5 nginx:latest \*:80->80/tcp

[node1] (local) root@192.168.0.48 ~

**If the node is ACTIVE, it is ready to accept tasks from the Master i.e. Manager. That is why if we need to do some maintenance on a node or if we want to restrict a node to get instructions from master, we drain the particular node. When we drain the instance of the service hosted gets migrated to some other node.**

$ docker node update --availability drain node2

node2

[node1] (local) root@192.168.0.48 ~

$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

wxy2lg28lgia web replicated 5/5 nginx:latest \*:80->80/tcp

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

xv3g1t50tnqz web.1 nginx:latest node1 Running Running 6 seconds ago

rj6z01nxhfxy \\_ web.1 nginx:latest node2 Shutdown Shutdown 8 seconds ago

ltla907oyhyd web.2 nginx:latest node1 Running Running 10 minutes ago

vql6z4y6ryr5 web.3 nginx:latest node3 Running Running 8 minutes ago

m9mhoz07vcts web.4 nginx:latest node4 Running Running 7 minutes ago

klcjxyq1c6om web.5 nginx:latest node3 Running Running 6 seconds ago

fj48c9k8ylxp \\_ web.5 nginx:latest node2 Shutdown Shutdown 7 seconds ago

[node1] (local) root@192.168.0.48 ~

**This is straight forward. In case We have an updated Docker image to roll out to the nodes, all We need to do is fire an service update command.**

$ docker service update --image nginx:latest web

web

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

[node1] (local) root@192.168.0.48 ~

**Remove the Service**

We can simply use the service rm command as shown below:

$ docker service rm web

web

[node1] (local) root@192.168.0.48 ~

$ docker service ps web

no such service: web

[node1] (local) root@192.168.0.48 ~

$