What is Docker Swarm?

Docker Swarm is a container orchestration tool that runs on the Docker platform. It helps users to create and manage a cluster of Docker nodes. Clustering in Docker is a crucial concept in providing redundancy by enabling Docker Swarm to fail over should one or more nodes in the cluster fail.

Docker Swarm uses the standard Docker API to communicate with other tools such as Docker Engine. It intelligently assigns containers to worker nodes and ensures resource optimization by scheduling container workloads to run on the most suitable node(s)

Lab setup

To demonstrate how Docker Swarm works, we have a simple cluster that comprises a Swarm Manager node and two worker nodes as shown. The Manager nodes handle all the cluster management tasks while the worker nodes will run the containers.

• swarm-manager	10.128.0.57
• worker-node-1	10.128.0.58
• worker-node-2	10.128.0.59

Step 1) Configure the Cluster hosts file

To start off, log into each of the nodes and update the /etc/hosts file with the following entries:

```
swarm-manager       10.128.0.57
worker-node-1       10.128.0.58
worker-node-2       10.128.0.59
```

Next, ensure that all the nodes can ping each other. Therefore, on the manager node, run the commands:

```
$ ping -c 4 10.128.0.58
$ ping -c 4 10.128.0.59
```

On worker Node 1

```
$ ping -c 4 10.128.0.57
$ ping -c 4 10.128.0.59
```

On worker Node 2

```
$ ping -c 4 10.128.0.57
$ ping -c 4 10.128.0.58
```

Step 2) Install Docker CE on all the nodes

The next step is to install Docker on all the nodes. We are going to install Docker Community Edition (Docker CE) which is free to install and use.

Therefore, log into each of the nodes and update the local package index.

```
$ sudo apt update
```

Next, install the prerequisites package needed during the installation

```
$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common -y
```

Once all the packages have been installed, add the Docker GPG key

```
$ sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg
```

In the next step, add the official Docker repository to your Ubuntu 22.04 system

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
```

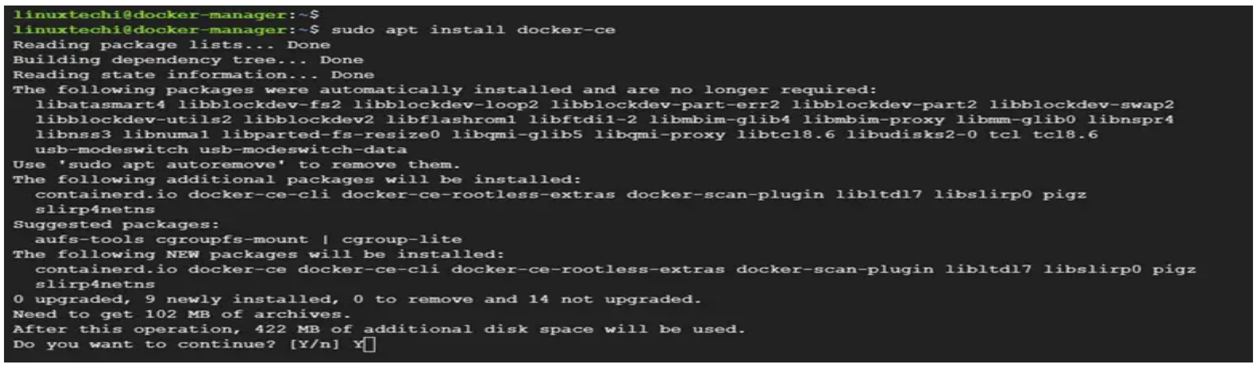
Next, update the local package index to make the system, aware of the newly added repository.

```
$ sudo apt update
```

Then install Docker from the official Docker repository,

```
$ sudo apt install docker-ce -y
```

The command installs Docker alongside additional packages that will be required by Docker to function as expected.



```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo apt install docker-ce
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libatasmart4 libblockdev-fs2 libblockdev-loop2 libblockdev-part-err2 libblockdev-part2 libblockdev-swap2
  libblockdev-utils2 libblockdev2 libflashrom1 libftdi1-2 libmbim-glib4 libmbim-proxy libmm-glib0 libnspr4
  libnss3 libnuma1 libparted-fs-resize0 libqmi-glib5 libqmi-proxy libtcl8.6 libudisks2-0 tcl tcl8.6
  usb-modeswitch usb-modeswitch-data
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  containerd.io docker-ce-cli docker-ce-rootless-extras docker-scan-plugin libltdl7 libslirp0 pigz
  slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-scan-plugin libltdl7 libslirp0 pigz
  slirp4netns
0 upgraded, 9 newly installed, 0 to remove and 14 not upgraded.
Need to get 102 MB of archives.
After this operation, 422 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

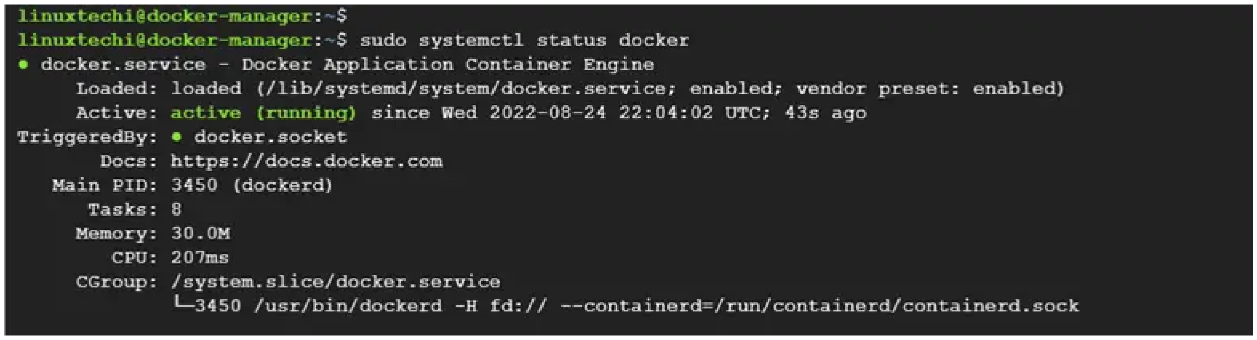
Once Docker is installed, add the currently logged-in user to the Docker group to avoid running Docker as a sudo user every time you run Docker.

```
$ sudo usermod -aG docker ${USER}
$ newgrp docker
```

Step 3) Verify Docker is running on all the nodes

Once installed, the Docker daemon starts automatically. You can verify that the service is running by running the command:

```
$ sudo systemctl status docker
```



```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2022-08-24 22:04:02 UTC; 43s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 3450 (dockerd)
      Tasks: 8
     Memory: 30.0M
        CPU: 207ms
     CGroup: /system.slice/docker.service
             └─3450 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

Additionally, be sure to enable the Docker service so that it starts automatically on boot time.

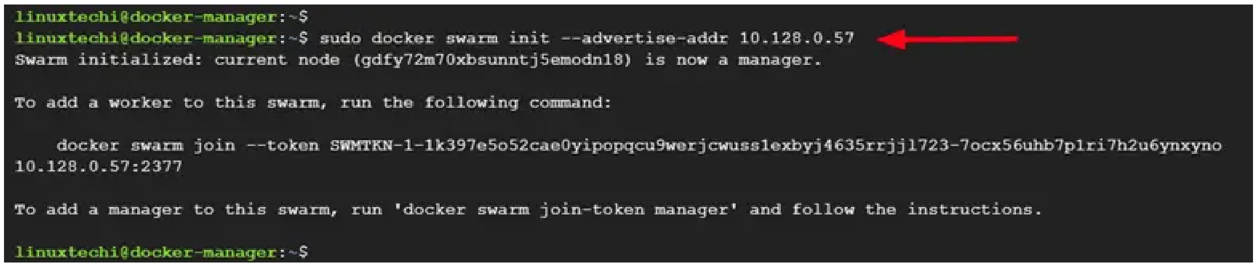
```
$ sudo systemctl enable docker
```

Step 4) Create Docker Swarm Cluster

The next step is to initialize the Docker Swarm on the Manager node. Once initialized, we will then add the worker nodes to the cluster.

To create a Docker Swarm Cluster, run the command:

```
$ sudo docker swarm init --advertise-addr 10.128.0.57
```



```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo docker swarm init --advertise-addr 10.128.0.57
Swarm initialized: current node (gdfy72m70xbsunntj5emodn18) is now a manager.

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

    docker swarm join --token SWMTKN-1-1k397e5o52cae0yipopqcu9werjcwuss1exbyj4635rrjjl723-7ocx56uhb7p1ri7h2u6ynxyno
10.128.0.57:2377

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

linuxtechi@docker-manager:~$
```

Once Docker Swarm has been initialized, a command for joining the worker nodes to the cluster will be displayed on the terminal. Copy the command as you will need to run it on each of the worker nodes as previously mentioned.

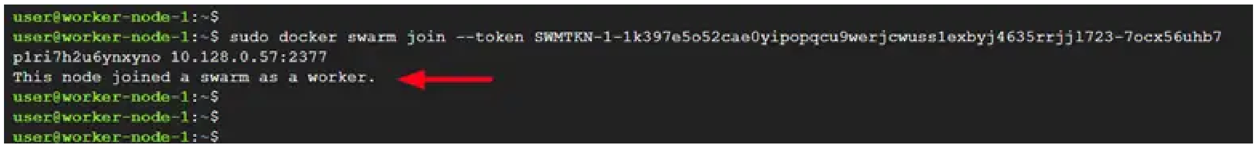
Next, login back to each of the worker nodes and paste the command in order to join the cluster.

```
$ sudo docker swarm join --token
SWMTKN-1-1k397e5o52cae0yipopqcu9werjcwuss1exbyj4635rrjjl723-7ocx
56uhb7p1ri7h2u6ynxyno 10.128.0.57:2377
```

If all goes well, you should get the following output

Output

This node joined a swarm as a worker

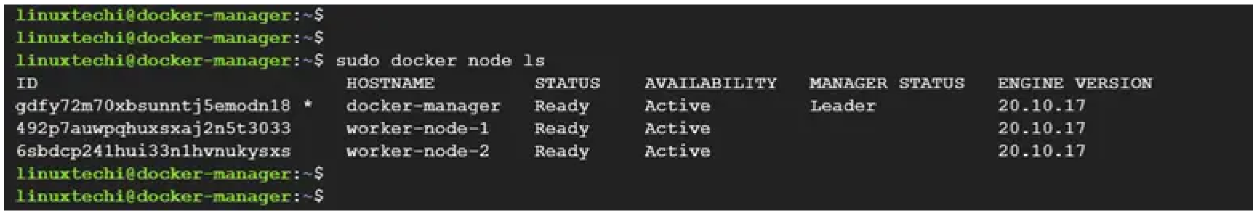


```
user@worker-node-1:~$
user@worker-node-1:~$ sudo docker swarm join --token SWMTKN-1-1k397e5o52cae0yipopqcu9werjcwuss1exbyj4635rrjjl723-7ocx56uhb7
p1ri7h2u6ynxyno 10.128.0.57:2377
This node joined a swarm as a worker.
user@worker-node-1:~$
user@worker-node-1:~$
user@worker-node-1:~$
```

Next, confirm that all the nodes have joined the cluster as follows.

```
$ sudo docker node ls
```

You should get the following output displaying all the nodes in the cluster.

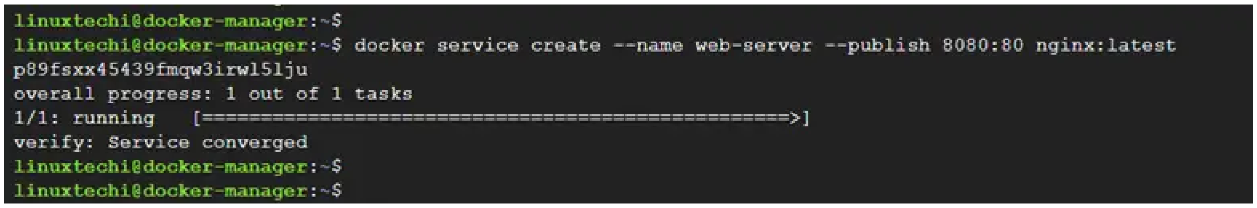


```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo docker node ls
ID                            HOSTNAME         STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
gdfy72m70xbsunntj5emodn18 *   docker-manager   Ready     Active         Leader           20.10.17
492p7auwpqhuxsxaj2n5t3033     worker-node-1    Ready     Active                          20.10.17
6sbdcp241hui33n1hvnukysxs     worker-node-2    Ready     Active                          20.10.17
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
```

Step 5) Test Docker Swarm Installation

To test docker swarm installation, head over to the manager node and deploy a container application to the cluster. In this example, we are deploying an Nginx web server container and mapping it to port 8080 on the host.

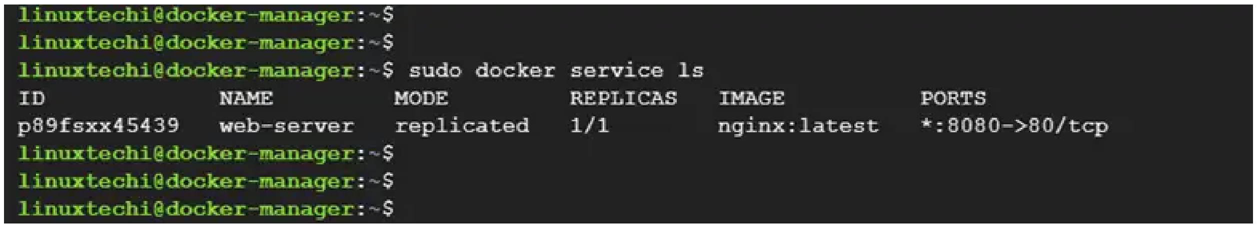
```
$ sudo docker service create --name web-server --publish 8080:80
nginx:latest
```



```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ docker service create --name web-server --publish 8080:80 nginx:latest
p89fsxx45439fmqw3irwl5lju
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
```

Next, verify the status of the application service deployed.

```
$ sudo docker service ls
```

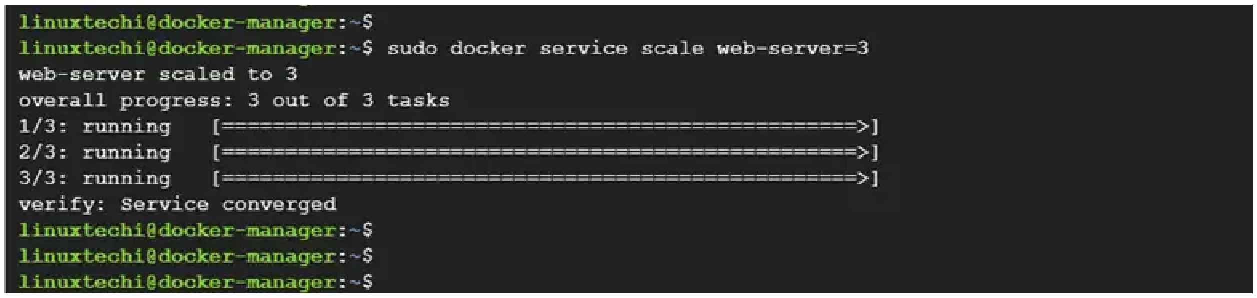


```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo docker service ls
ID             NAME         MODE         REPLICAS   IMAGE          PORTS
p89fsxx45439   web-server   replicated   1/1        nginx:latest   *:8080->80/tcp
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
```

Step 6) Create replicas of the service

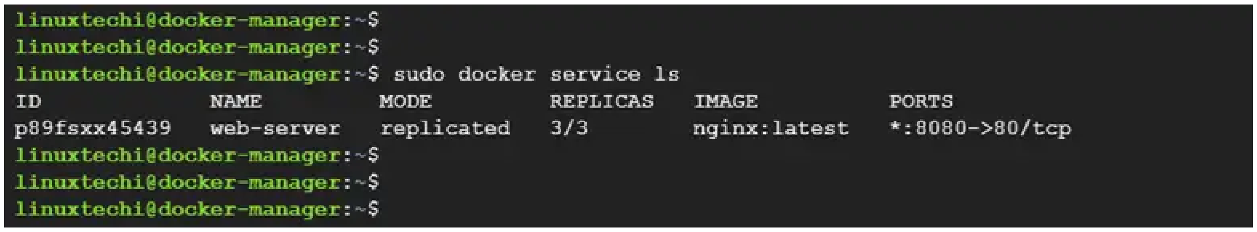
Finally, create three replicas of the service and scale them across both the Docker manager and the worker nodes.

```
$ sudo docker service scale web-server=3
```



```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo docker service scale web-server=3
web-server scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
```

Next, confirm the status of the replicas. This time around, you will notice that we have 3 replicas.



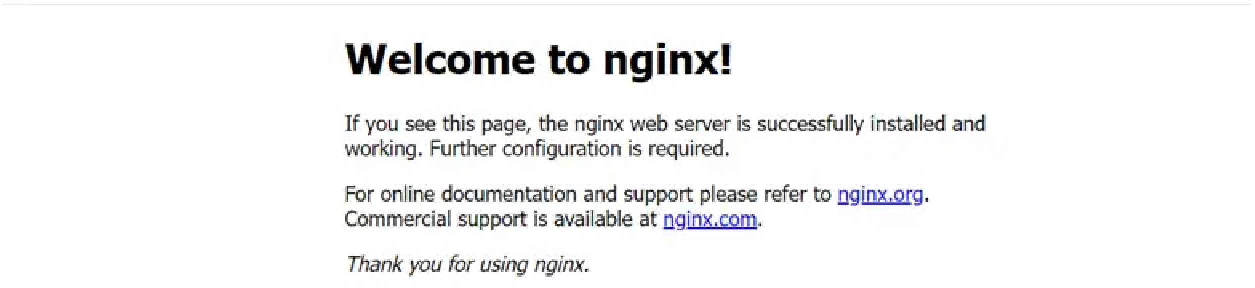
```
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$ sudo docker service ls
ID             NAME         MODE         REPLICAS   IMAGE          PORTS
p89fsxx45439   web-server   replicated   3/3        nginx:latest   *:8080->80/tcp
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
linuxtechi@docker-manager:~$
```

At this point, Nginx web server container should be running across all the nodes in the cluster on port 8080. To confirm this, head over to your browser, and access the web server from all the nodes.

http://manager-node:8080

http://worker-node-1:8080

http://worker-node-2:8080



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Conclusion

In this guide, we managed to install and configure Docker Swarm. We also went a step further and deployed an application to the cluster and later scaled it across all the nodes in the cluster.