What is the most important feature of any web-based application? There are many, but for me **high availability** is the most important. That is what Docker Swarm helps us achieve! It helps in the application being highly available.

In my previous blog, I explained how Docker Compose works. This blog on Docker Swarm is a continuation to the former and here the benefits of using Docker Swarm for containerizing any multi-container application has been explained.

In this blog's case, it is only an Angular application which will be Docker Swarm'ed.

***Note**: The method to containerize the MEAN Stack app is the same.*

So, What Is Docker Swarm?

Docker Swarm is a technique to create and maintain a cluster of**Docker Engines**. The Docker engines can be hosted on different nodes, and these nodes which are in remote locations form a*Cluster* when connected in Swarm mode.

Why Use Docker Swarm?

For reasons mentioned already! Achieving **high availability** without any downtime is a priority for every service provider out there. Will high availability impress your clients? Well, they won't be impressed if they face downtime. That is a no-brainer.

Other Benefits Of Docker Swarm

Like a lot of other services, Docker Swarm does auto **load-balancing**for us. Hence, there is no need for DevOps engineers to route processing requests to other nodes when one fails. The cluster's manager will automatically perform load balancing for us.

Decentralized access is another benefit. What does that mean? It means all nodes can be accessed easily from the manager. The manager will also prompt the nodes on a regular basis, and keep a track of its health/ status to cope with downtime. However, nodes cannot access or track the services running in other nodes/ managers.

You can check the no. of containers running in a node, **scale-up** the no. of containers or **scale-down** the no. based on our requirement, by just executing a single command.

Even after an application has been deployed, we can issue **rolling-updates** and make sure that CI (Continuous Integration) is achieved. Rolling updates are issued to one node after the other thus making sure there is no downtime and load is distributed between other nodes in the cluster.

So, what next? To do the obvious. Get started with Docker Swarm if you have already worked on Docker or if your organization wishes to containerize a reliable web service.

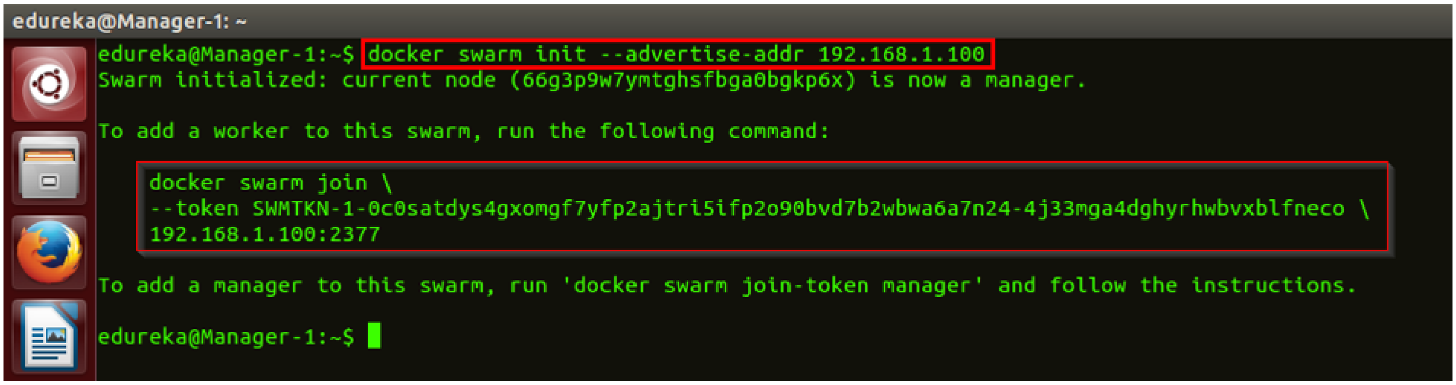
***Note**: Docker engines are installed on independent hosts/ servers or in multiple VMs in a host.*

Getting Started With Swarm Mode

Docker Swarm is initiated by the manager, or let me put it this way, the instance which starts the Swarm cluster becomes the manager. The command to start the cluster is:

```
$ docker swarm init --advertise-addr ip-address
```

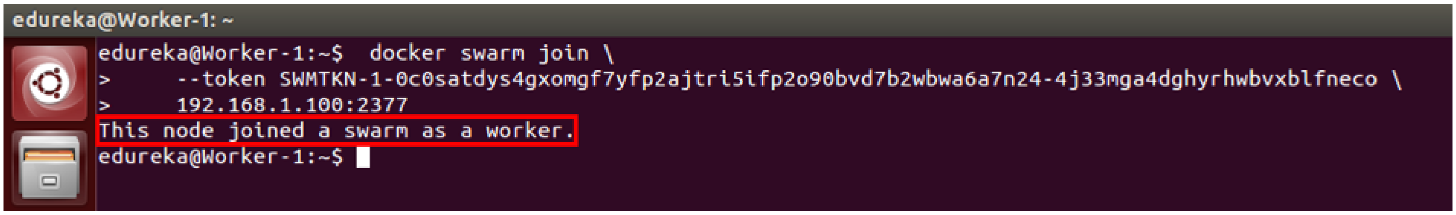
Here, the '–advertise-addr' flag is used for advertising itself to other nodes who want to join the cluster. The IP address of the manager needs to be specified along with the flag. Below is the sample screenshot.



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker swarm init --advertise-addr 192.168.1.100
Swarm initialized: current node (66g3p9w7ymtghsfbga0bgkp6x) is now a manager.

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

    docker swarm join \
    --token SWMTKN-1-0c0satdys4gxomgf7yfp2ajtri5ifp2o90bvd7b2wbwa6a7n24-4j33mga4dghyrhwbvxblfneco \
    192.168.1.100:2377

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

edureka@Manager-1:~$
```

When the Swarm cluster is initiated, a token is generated at the manager's end. This token needs to be used by other nodes to join the swarm cluster.

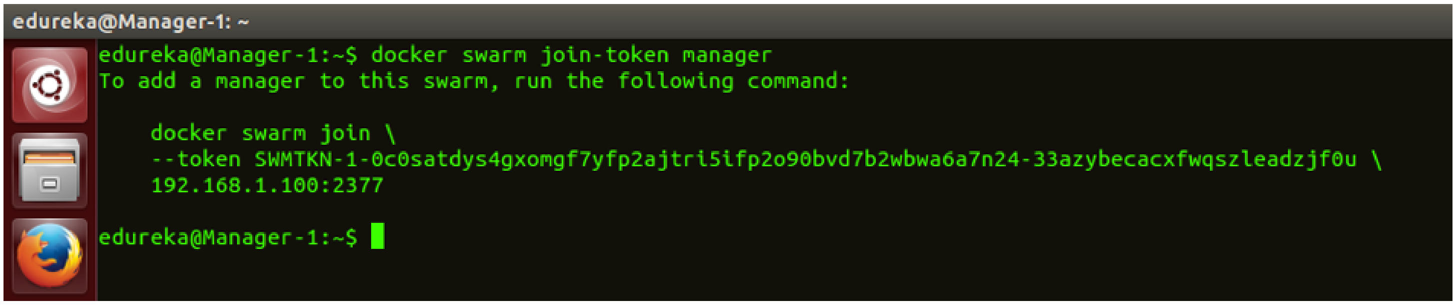
How is it exactly? Copy the entire token generated at the manager's docker engine, paste it at the node's docker engine and execute it. The highlighted portion of the screenshot above is a token. When the token is executed at a worker node, it will look like the below screenshot.



```
edureka@Worker-1: ~
edureka@Worker-1:~$  docker swarm join \
>     --token SWMTKN-1-0c0satdys4gxomgf7yfp2ajtri5ifp2o90bvd7b2wbwa6a7n24-4j33mga4dghyrhwbvxblfneco \
>     192.168.1.100:2377
This node joined a swarm as a worker.
edureka@Worker-1:~$
```

Any node that joins the cluster can be later promoted to a manager. In case you want a docker engine to join as a manager, execute the below command at the manager's end:

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



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-0c0satdys4gxomgf7yfp2ajtri5ifp2o90bvd7b2wbwa6a7n24-33azybecacxfwqszleadzjf0u \
    192.168.1.100:2377

edureka@Manager-1:~$
```

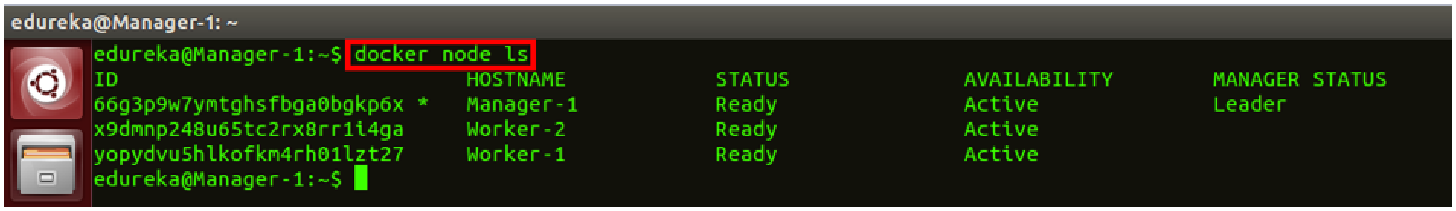
And at a later point in time, if you want the token for a node to join the cluster, run the below command:

```
$ docker swarm join-token node
```

Go ahead, and execute the token at every node you want, to join the cluster. When all that is done, you can run a docker node list command to check how many nodes have joined the cluster along with their status. The command is:

```
$ docker node ls
```

The screenshot is below:



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS
66g3p9w7ymtghsfbga0bgkp6x *   Manager-1           Ready               Active              Leader
x9dmnp248u65tc2rx8rr1i4ga     Worker-2            Ready               Active
yopydvu5hlkofkm4rh01lzt27     Worker-1            Ready               Active
edureka@Manager-1:~$
```

Creating A Docker Image For Angular App

If all is well, then we can start our Swarm service, provided the Docker Image is built. The Docker image can be built from the Dockerfile. The Dockerfile used to build the applications is below:

```
FROM node:6
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY package.json /usr/src/app
RUN npm cache clean
RUN npm install
COPY . /usr/src/app
EXPOSE 4200
CMD ["npm","start"]
```

The Dockerfile is used to execute a set of commands together for building a custom Docker image from a base image. As you can see, the base image I have used is 'Node:6'. NodeJS is the image I from Docker Hub which is tagged with version 6.

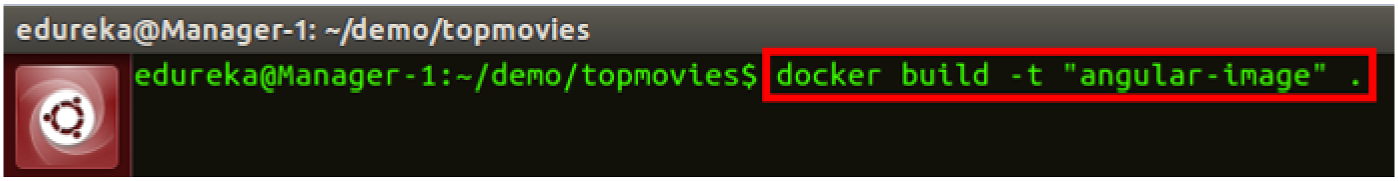
I am then creating a new Docker directory inside the container and making it the working directory inside my container.

I am the copying the 'package.json' file from my local machine to the container's working directory. I am then specifying 'RUN npm cache clean' and 'RUN npm install' commands. *npm install* command downloads the version of dependencies mentioned in the package.json file.

I am then copying all the project codes from the local machine to the container, exposing port number 4200 for accessing the Angular application on the browser and finally, I am specifying the npm start command which containerizes the application.

Now, to create the Docker image based on this Dockerfile, run the below command:

```
$ docker build -t angular-image .
```



```
edureka@Manager-1: ~/demo/topmovies
edureka@Manager-1:~/demo/topmovies$ docker build -t "angular-image" .
```

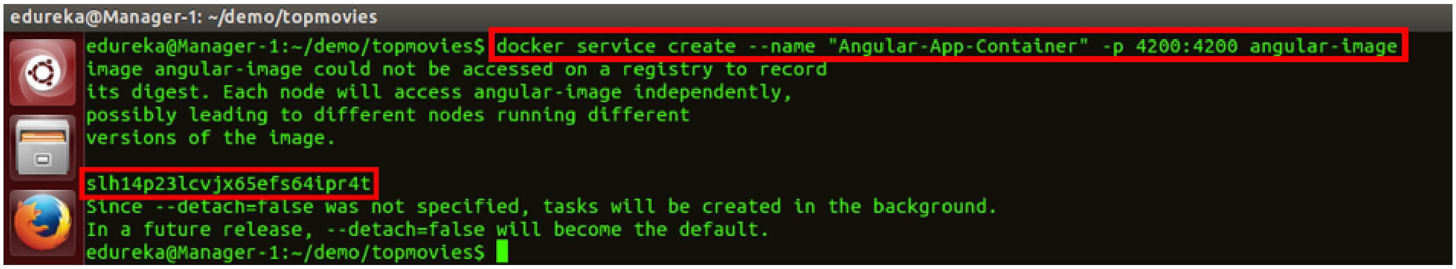
Note: *The Docker Images need to be built in all the nodes in the cluster. Without it, containers cannot be spun in other Docker engines.*

Learn Docker From Industry Experts

Starting The Docker Swarm Service

Given that our Docker Image is built, we can spin a container out of this image. But, we will do something better: create a Docker Swarm service out of it. Command to create a swarm service is:

```
$ docker service create --name "Angular-App-Container" -p 4200:4200 angular-image
```



```
edureka@Manager-1: ~/demo/topmovies
edureka@Manager-1:~/demo/topmovies$ docker service create --name "Angular-App-Container" -p 4200:4200 angular-image
image angular-image could not be accessed on a registry to record
its digest. Each node will access angular-image independently,
possibly leading to different nodes running different
versions of the image.

slh14p23lcvjx65efs64ipr4t
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
edureka@Manager-1:~/demo/topmovies$
```

Here, the 'name' flag is used to give a name to my service and 'p' flag is used to expose the container port to the host port. In the package.json file, I have specified the container port on which the Angular app should be hosted. And the 4200 in this command helps map the container's port 4200 to host's port 4200. 'angular-image' is the name of the image I earlier built.

Remember: When we create a service, it can be hosted on any docker engine in the cluster. The manager of the swarm will decide where it will be hosted. But, no matter in which node it is hosted, the application can be accessed on localhost:4200 from any of the nodes connected in the cluster.

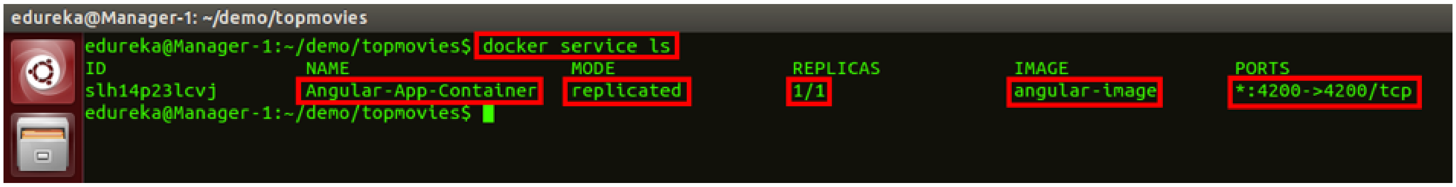
How is that possible? Because Swarm internally exposes the port numbers to be accessible by every other node in the cluster. That means, port no. 4200 on any node/ manager in the cluster would render the Angular application.

Now what? Is the container active?

You can verify if the service is containerized by running the docker service list command. But, it may take a minute for the container to be deployed. Below is the command:

```
$ docker service ls
```

This command will list down all the services managed by the Swarm cluster. In our case, it should display one active container. Look at the below screenshot for reference.



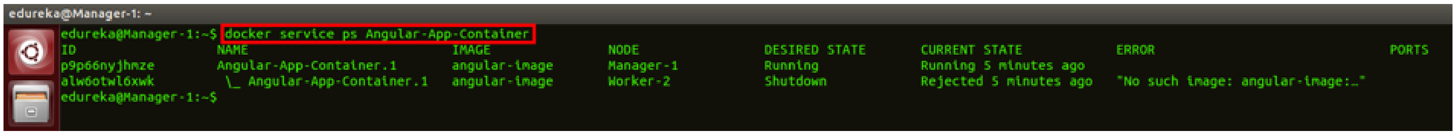
```
edureka@Manager-1: ~/demo/topmovies
edureka@Manager-1:~/demo/topmovies$ docker service ls
ID              NAME                    MODE          REPLICAS        IMAGE           PORTS
slh14p23lcvj    Angular-App-Container   replicated    1/1             angular-image   *:4200->4200/tcp
edureka@Manager-1:~/demo/topmovies$
```

Here, “REPLICAS=1/1” indicates that there is one single ‘service’ of that container, in the cluster. And “MODE=replicated” indicates that the service is replicated on all the nodes in the cluster.

Now, to identify on which the node/ manager, the app is hosted, we can run the command docker service ps command followed by the container name. The command is:

```
$ docker service ps Angular-App-Container
```

The screenshot for the same is below.



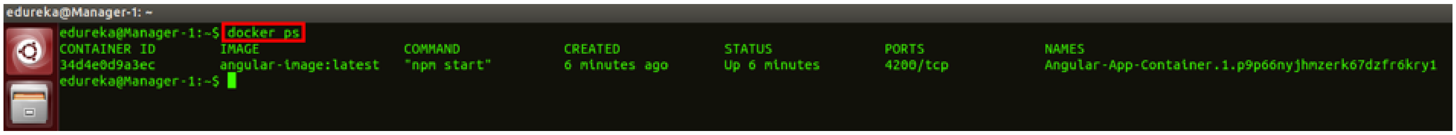
```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service ps Angular-App-Container
ID             NAME                         IMAGE           NODE        DESIRED STATE   CURRENT STATE           ERROR                                PORTS
p9p66nyjhmze   Angular-App-Container.1      angular-image   Manager-1   Running         Running 5 minutes ago
alw6otwl6xwk    \_ Angular-App-Container.1  angular-image   Worker-2    Shutdown        Rejected 5 minutes ago  "No such image: angular-image:…"
edureka@Manager-1:~$
```

This mentions details about the node on which the application is hosted along with the command used to start withe service.

The ‘docker ps’ command throws light on the details about the active container. The command is:

```
$ docker ps
```

Look at the below screenshot for reference.



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker ps
CONTAINER ID    IMAGE                  COMMAND        CREATED         STATUS         PORTS       NAMES
34d4e0d9a3ec    angular-image:latest   "npm start"    6 minutes ago   Up 6 minutes   4200/tcp    Angular-App-Container.1.p9p66nyjhmzerk67dzfr6kry1
edureka@Manager-1:~$
```

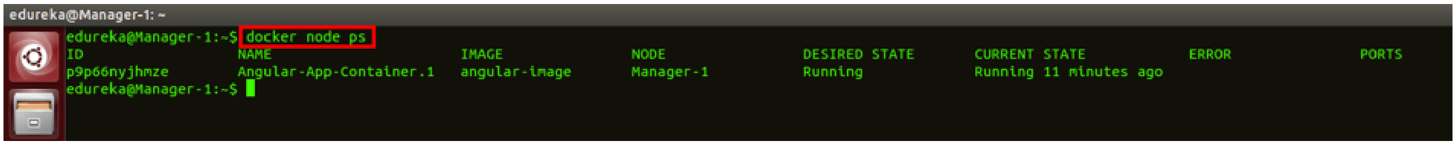
But, this command will only work on the cluster manager and the node where the service is actually hosted.

To check how many nodes are running, run the node list command. Command is:

```
$ docker node ls
```

To check the containers running in a particular host, run the node ps command. Command is:

```
$ docker node ps
```



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker node ps
ID                  NAME                       IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
p9p66nyjhmze        Angular-App-Container.1    angular-image       Manager-1           Running             Running 11 minutes ago
edureka@Manager-1:~$
```

If you remember, I earlier mentioned that the service is currently running in replicated MODE. This means the service is replicated across all the nodes in the clusters. Do you think there is an alternative?

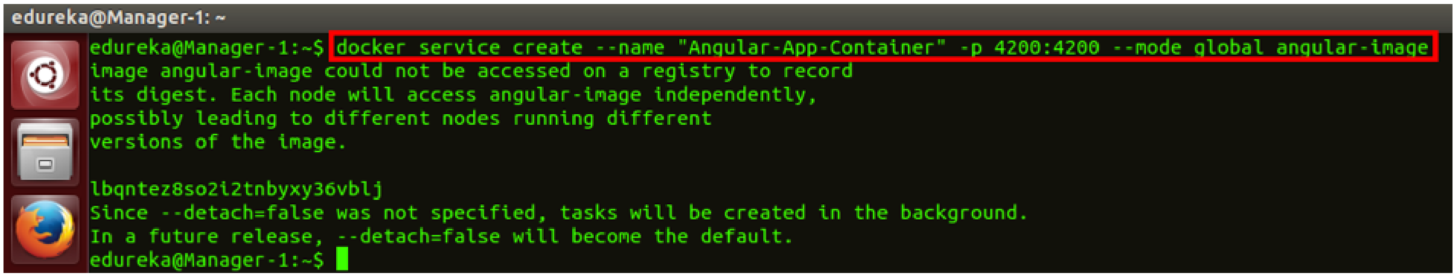
Absolutely! There is something called as Global MODE. In this mode, there is a service of this container running at every single/ manager in the cluster. Remember to stop the current service/ container before to spinning another set of containers.

The command for that is:

```
$ docker service rm Angular-App-Container
```

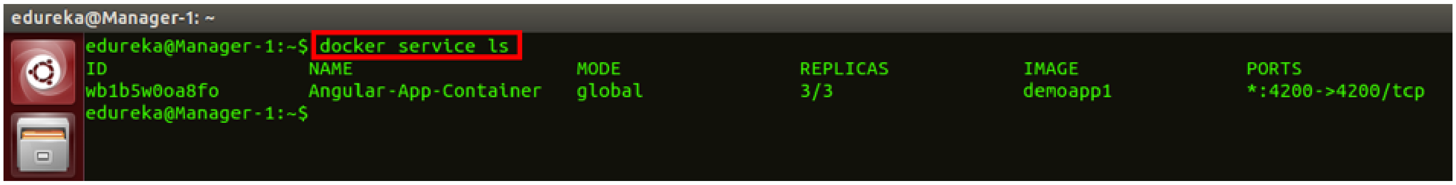
The command to spin the container in Global mode is:

```
$ docker service create --name "Angular-App-Container" -p 4200:4200 --mode global angular-image
```



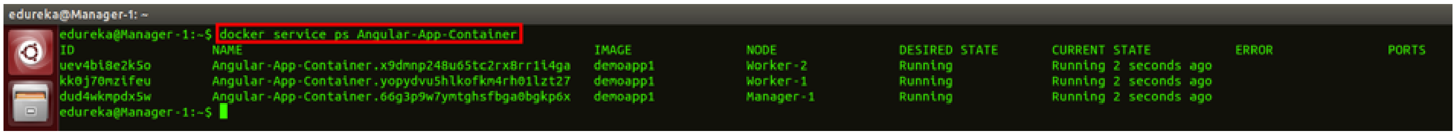
```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service create --name "Angular-App-Container" -p 4200:4200 --mode global angular-image
image angular-image could not be accessed on a registry to record
its digest. Each node will access angular-image independently,
possibly leading to different nodes running different
versions of the image.

lbqntez8so2i2tnbyxy36vblj
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
edureka@Manager-1:~$
```

This would create 3 services on the 3 nodes in our cluster. You can verify it by running the docker service list command. The screenshot of this is below.



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service ls
ID                  NAME                    MODE                REPLICAS            IMAGE               PORTS
wb1b5w0oa8fo        Angular-App-Container   global              3/3                 demoapp1            *:4200->4200/tcp
edureka@Manager-1:~$
```

When the docker service ps command is executed, you will see something like this:

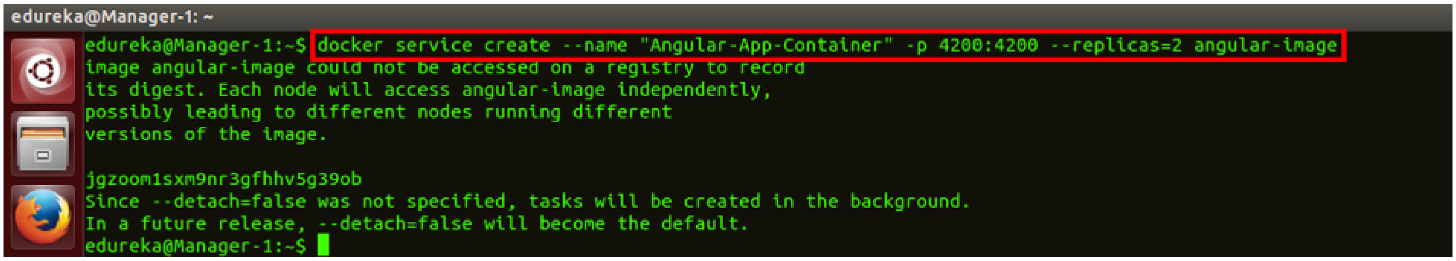


```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service ps Angular-App-Container
ID                  NAME                                                IMAGE               NODE                DESIRED STATE       CURRENT STATE           ERROR               PORTS
uev4bi8e2k5o        Angular-App-Container.x9dmnp248u65tc2rx8rr1i4ga     demoapp1            Worker-2            Running             Running 2 seconds ago
kk0j70mzifeu        Angular-App-Container.yopydvu5hlkofkm4rh01lzt27     demoapp1            Worker-1            Running             Running 2 seconds ago
dud4wkmpdx5w        Angular-App-Container.66g3p9w7ymtghsfbga0bgkp6x     demoapp1            Manager-1           Running             Running 2 seconds ago
edureka@Manager-1:~$
```

As you can see, it says the mode is replicated and the replicas of this container is 3. Now comes the best part of this blog.

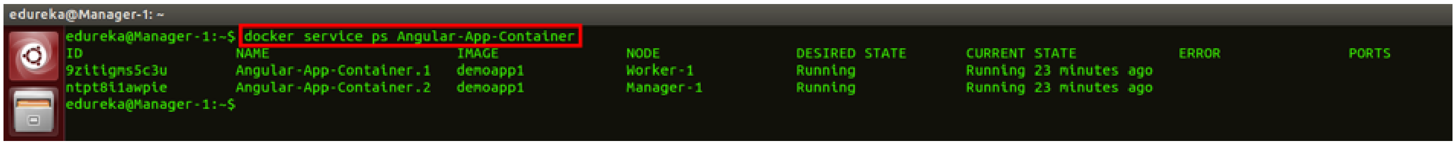
To have 2 replicas of the services running between the three containers, we can use the replicas flag. Look at the command below:

```
$ docker service create --name "Angular-App-Container" -p 4200:4200 --replicas=2 angular-image
```



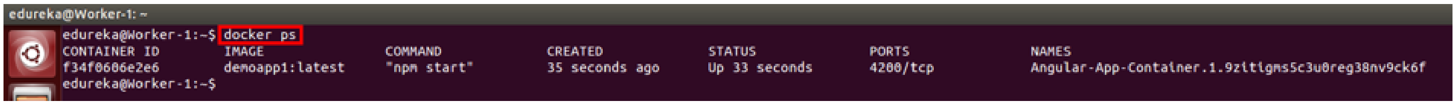
```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service create --name "Angular-App-Container" -p 4200:4200 --replicas=2 angular-image
image angular-image could not be accessed on a registry to record
its digest. Each node will access angular-image independently,
possibly leading to different nodes running different
versions of the image.

jgzoom1sxm9nr3gfhhv5g39ob
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
edureka@Manager-1:~$ 
```

You will notice that these 2 services are load balanced between the three nodes in the cluster. Run the docker service process command to verify, in which nodes the containers are active. Look at the below screenshot for reference. The containers are active in one manager node and one worker node.



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service ps Angular-App-Container
ID                  NAME                      IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
9zitigms5c3u        Angular-App-Container.1   demoapp1            Worker-1            Running             Running 23 minutes ago
ntpt8i1awpie        Angular-App-Container.2   demoapp1            Manager-1           Running             Running 23 minutes ago
edureka@Manager-1:~$
```

From the Worker node, you can verify the container is running by executing the 'docker ps' command.



```
edureka@Worker-1: ~
edureka@Worker-1:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
f34f0606e2e6        demoapp1:latest     "npm start"         35 seconds ago      Up 33 seconds       4200/tcp            Angular-App-Container.1.9zitigms5c3u0reg38nv9ck6f
edureka@Worker-1:~$
```

Docker Swarm For High Availability

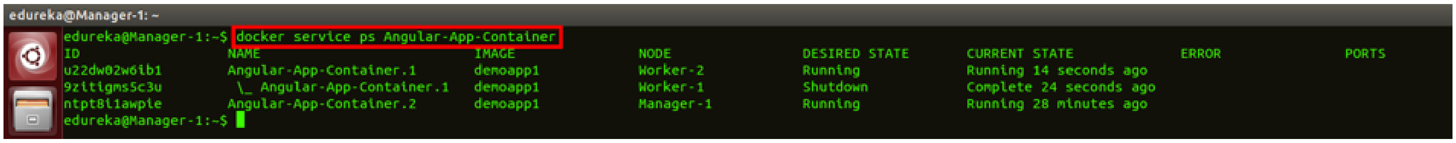
Now to actually verify that there is high availability in our cluster, we need to experience a scenario where one of the nodes goes down and other nodes in the cluster make up for it. We can bring about that scenario by manually stopping the container from one of the nodes using this command:

```
$ docker stop Angular-App-Container
```

Run the above command on the node: Worker-1 where the container is running. From the manager, run the command:

```
$ docker service ps Angular-App-Container
```

You will now notice that the container is now running in node: Worker-2 and Manager. However, it has been shutdown from node: Worker-1. The same is visible from the below screenshot.



```
edureka@Manager-1: ~
edureka@Manager-1:~$ docker service ps Angular-App-Container
ID                  NAME                         IMAGE               NODE                DESIRED STATE       CURRENT STATE            ERROR               PORTS
u22dw02w6ib1        Angular-App-Container.1      demoapp1            Worker-2            Running             Running 14 seconds ago
9zitigms5c3u         \_ Angular-App-Container.1  demoapp1            Worker-1            Shutdown            Complete 24 seconds ago
ntpt8i1awpie        Angular-App-Container.2      demoapp1            Manager-1           Running             Running 28 minutes ago
edureka@Manager-1:~$ 
```

This is how ***Docker High Availability*** is achieved. In spite of the container being inactive in Worker-1, the application can be rendered at port number 4200 on that worker node. This is because it is