DOCKER SWARM

TEAM: 121AD0012, 121AD0014, 121AD0019, 121AD0020

What is Docker Swarm and why do we need it?

Current versions of Docker include Swarm mode for natively managing a cluster of Docker Engines called a swarm. Docker Swarm mode is built into the Docker Engine.
One of the key advantages of swarm services over standalone containers is that you can modify a service's configuration, including the networks and volumes it is connected to, without the need to manually restart the service. Docker will update the configuration, stop the service tasks with out of date configuration, and create new ones matching the desired configuration. There are many other aspects of Docker swarm mode such as follows:

1. **Cluster management**, create or manage a swarm
2. **Decentralized design**, ensuring no single point of failure
3. **Declarative service model**, allowing to set custom configurations with ease
4. **Scaling**, balance the model based on different workloads
5. **Multi-host networking**, using overlay networks
6. **Rolling updates**, all nodes get updated simultaneously, and when in need, we may go back to the previous versions of that update.
7. **Secure**, by ensuring TLS mutual authentication and encryption

A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon. Docker daemons can participate in a swarm as managers, workers, or both.
Let us define a few terminologies, before we proceed to set up a swarm.

1. **NODE**: A node is an instance of the Docker engine participating in the swarm. Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks.

Worker nodes receive and execute tasks dispatched from manager nodes. By default manager nodes also run services as worker nodes, but you can configure them to run manager tasks exclusively and be manager-only nodes.

2. **SERVICE**: A service is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm.

   A task carries a Docker container and the commands to run inside the container. It is the atomic scheduling unit of swarm.

Creating a Swarm

In order to create a swarm, we have chose a manager node with ip address of 172.16.30.191

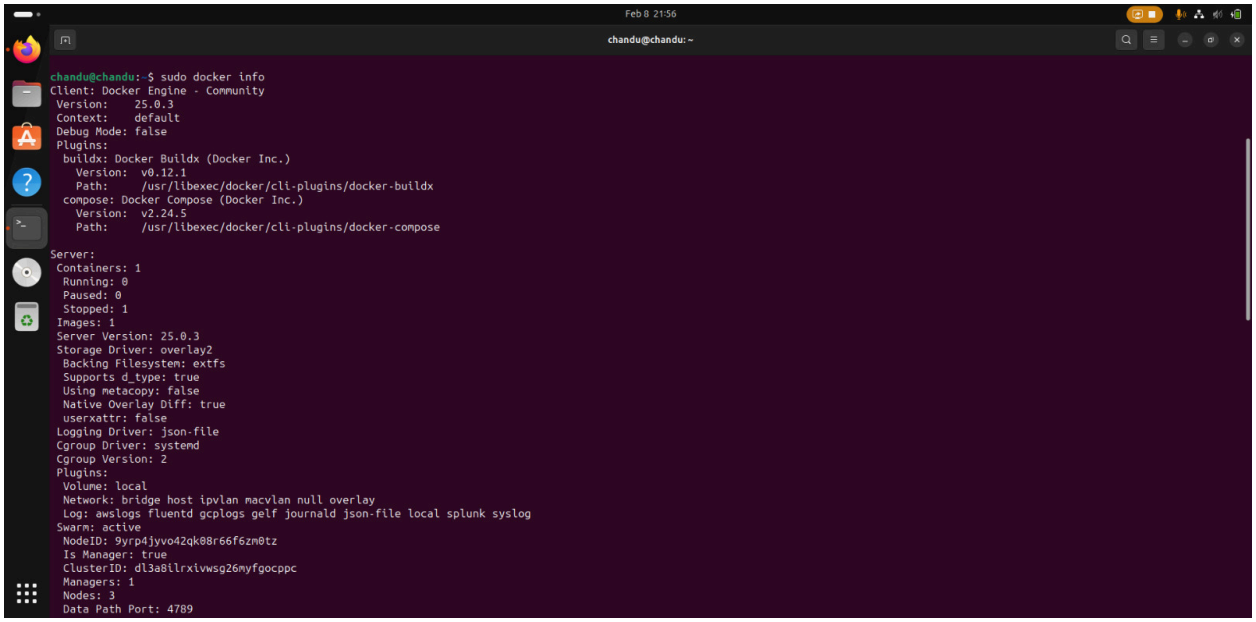
Worker nodes IPs: 172.16.30.192, 172.16.29.194, 172.16.29.193

On manager node we run the following command:

sudo docker swarm init --advertise-addr 172.16.30.191

The output of this code does two things. First, the IP is published as a swarm manager who is ready to have workers. Second, in order to join this swarm, a *token* is provided to maintain authentication.

On running commands *sudo docker info* and *sudo docker node ls* they give information on the worker nodes.



Joining worker nodes in Swarm

To add the worker nodes to the swarm we can use the *token* above in the worker machine.

```
bhuvna@bhuvana:~$ hostname -I
172.16.30.192 172.17.0.1
bhuvna@bhuvana:~$ sudo docker swarm join --token SWMTKN-1-3bnbqucr02f61w5u2z2vyucfpx8dszgex26rawftvsti5990r0-7y42w776mb
dfhrldy2pgne9nt 172.16.30.191:2377
[sudo] password for bhuvna:
This node joined a swarm as a worker.
```

```
hemanth@hemanth:~$ hostname -I
172.16.29.194 172.17.0.1
hemanth@hemanth:~$ sudo docker swarm join --token SWMTKN-1-3bnbqucr02f61w5u2z2vyucfpx8dszge
x26rawftvsti5990r0-7y42w776mbdfhrldy2pgne9nt 172.16.30.191:2377
[sudo] password for hemanth:
This node joined a swarm as a worker.
```

```
sri123@onlyforGlusty:~$ hostname -I
192.168.137.171 172.19.0.1 172.21.0.11 172.17.0.1 172.20.0.1 172.18.0.1
sri123@onlyforGlusty:~$ sudo docker swarm join --token SWMTKN-1-3bnbqucr02f61w5u2z2vyucfpx8dszgex26rawftvsti5990r0-7y42w776mbdfhrldy
2pgne9nt 172.16.30.191:2377
[sudo] password for sri123:
This node joined a swarm as a worker.
sri123@onlyforGlusty:~$
```

```
chandu@chandu:~$ sudo docker node ls
ID                            HOSTNAME        STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
1bobo5v7qv5gufhf361nm83yn     bhuvana         Ready     Active                          25.0.3
9yrp4jyvo42qk08r66f6zm0tz *   chandu          Ready     Active         Leader           25.0.3
0wt36wq69tu48wsz72ms8bxui     hemanth         Ready     Active                          25.0.3
fw2wqcresi0llmru09dlbzxip     onlyforGlusty   Ready     Active                          25.0.2
chandu@chandu:~$
```

Deploying a service over Swarm

For simplicity, we have tried to deploy a simple alpine container over the manager node

docker service create --replicas 1 --name c2 alpine ping docker.com

Here, --replicas is the desired state of running instances, --name is service name, and alpine ping docker.com is the command to be run on that service. We choose ping in order to keep the service alive for a long time.

In this case, we see that the manager node is running the service. By default manager nodes also run services as worker nodes.

Inspecting the service

For inspecting the service we use the following commands on manager:

docker service inspect --pretty c2
docker service inspect c2
docker service ps c2

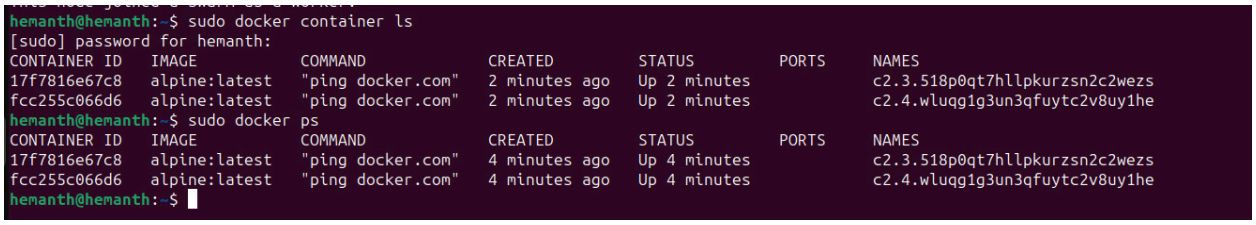
On workers:

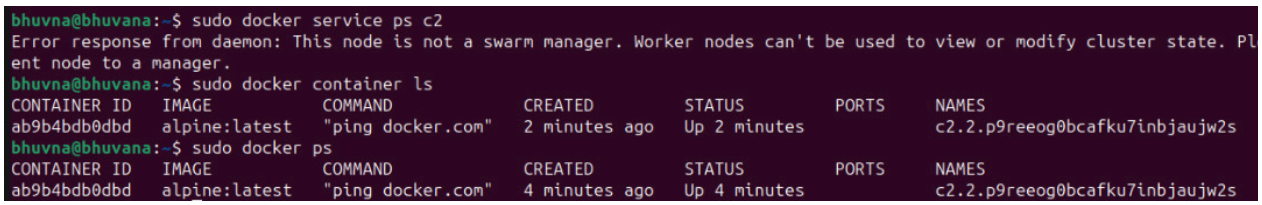
docker ps

Scaling the service

Scaling increases the number of running instances of the service we have deployed. To do so we use: *docker service scale c2=5*

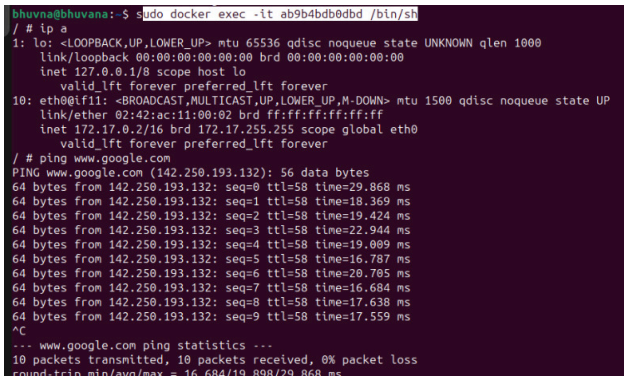
To check the status we may run *sudo docker service ps c2* or *sudo docker container ls* on both the manager and worker nodes.

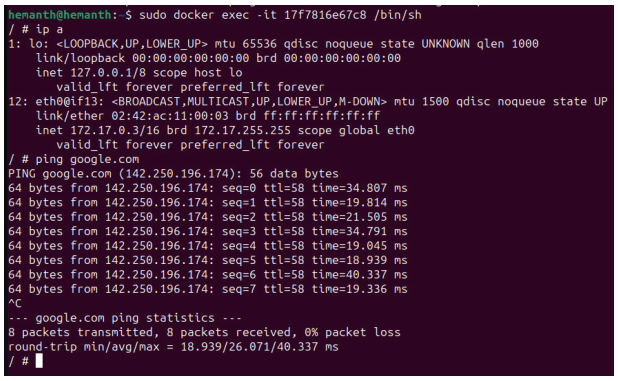


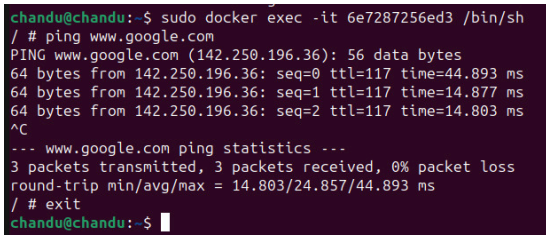


The instances running on respective worker nodes could be accessed and terminal to that instance could be opened.

This confirms that four nodes are connected in a swarm, where the manager node is specified as the leader above.







But the full details could only be seen by the manager node, the worker nodes were not given access to that data.

Removing the service

To remove the service, command *sudo docker service rm c2* has to be run on the manager node.

On removing the service on manager node, it also gets removed from the worker nodes:

```
bhuvna@bhuvana:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
bhuvna@bhuvana:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED        STATUS                     PORTS     NAMES
7057e396d659   alpine    "/bin/sh"   20 hours ago   Exited (0) 20 hours ago              c1
bhuvna@bhuvana:~$
```

```
/ # hemanth@hemanth:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
hemanth@hemanth:~$ sudo docker ps - a
"docker ps" accepts no arguments.
See 'docker ps --help'.

Usage:  docker ps [OPTIONS]

List containers
hemanth@hemanth:~$ sudo docker ps -a
CONTAINER ID   IMAGE     COMMAND     CREATED        STATUS                     PORTS     NAMES
cf932c782dd1   alpine    "/bin/sh"   4 hours ago    Exited (0) 4 hours ago               c1
```

Removing the swarm

To dissolve the swarm we may use the following command on all the nodes:

sudo docker swarm leave --force

```
hemanth@hemanth:~$ sudo docker swarm join --token SWMTKN-1-3bnbqucr02f61w5u2z2vyucfpx8dszgex26rawftvsti5990r0-7y42w776mbdfhrldy2pgne9nt
172.16.30.191:2377
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave this swarm and join another one.
hemanth@hemanth:~$ sudo docker swarm leave --force
Node left the swarm.
```

If the only manager node leaves the Swarm, the remaining worker nodes are left without any management authority. Overlay networks created for the Swarm will cease to function as there is no longer a Swarm to manage them. In summary, dissolving the Swarm by forcefully removing the last manager node effectively disbands the cluster.

Apply rolling updates to a service

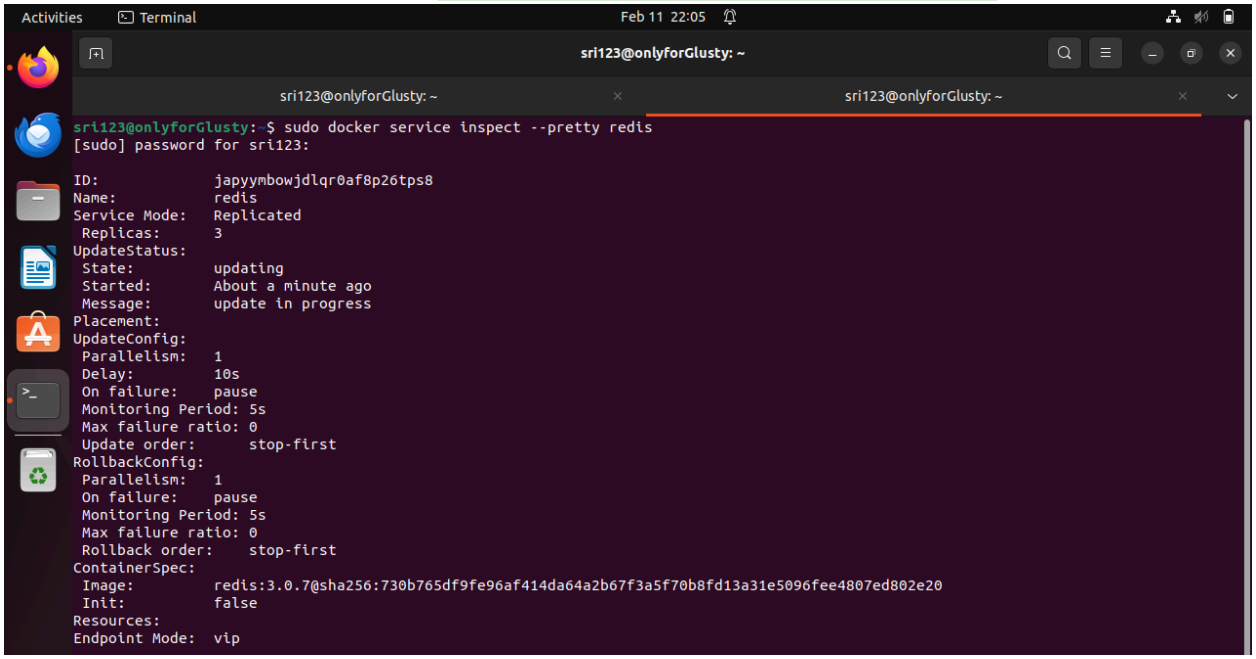
Here, we will deploy a service based on the Redis 3.0.6 container tag. Then we will upgrade the service to use the Redis 3.0.7 container image using *rolling updates*.

sudo docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6

Above command will install the 3.0.6 (old) version of redis. Redis is an open-source, networked, in-memory, key-value data store with optional durability, written in ANSI C.

```
sri123@onlyforGlusty:~$ sudo docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6
japyymbowjdlqr0af8p26tps8
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS      NAMES
96a27f6a7223   redis:3.0.6   "/entrypoint.sh redi…"   15 seconds ago   Up 13 seconds   6379/tcp   redis.2.9jqjbq44ub3bfxksnfmyrdja7
```

Inspect the downloaded service: *sudo docker service inspect --pretty redis*



Next we apply update using command: *sudo docker service update --image redis:3.0.7 redis*

```
sri123@onlyforGlusty:~$ sudo docker service update --image redis:3.0.7 redis
redis
overall progress: 2 out of 3 tasks
1/3: preparing [=================================>                 ]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
```

```
sri123@onlyforGlusty:~$ sudo docker container ls
[sudo] password for sri123:
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS      NAMES
41773bd93a4a   redis:3.0.7   "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   6379/tcp   redis.2.uu3biknu4dqculxs8i0ri6tbz
sri123@onlyforGlusty:~$
```

```
hemanth@hemanth:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS      NAMES
cfa8fe925ed2   redis:3.0.6   "/entrypoint.sh redi…"   16 seconds ago   Up 15 seconds   6379/tcp   redis.1.dydn8eewgfhmnz8f7yxhcml35
hemanth@hemanth:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS      NAMES
813993a0ee08   redis:3.0.7   "docker-entrypoint.s…"   3 minutes ago   Up 2 minutes   6379/tcp   redis.1.qt2r84z3f2o38yhvddy3i23nn
hemanth@hemanth:~$
```

The swarm manager applies the update to nodes according to the UpdateConfig policy. The schedule applies this rolling update with following rules:

- If the update to a task returns RUNNING, wait for the specified delay period then start the next task.
- If, at any time during the update, a task returns FAILED, pause the update.

Now on inspecting the service again we get:

```
sri123@onlyforGlusty:~$ sudo docker service update --image redis:3.0.7 redis
redis
overall progress: 1 out of 3 tasks
1/3: running   [==================================================>]
2/3: preparing [=================================>                 ]
3/3:
service update paused: update paused due to failure or early termination of task 60t0kpvan9lcxximtbzgp3r94
sri123@onlyforGlusty:~$ sudo docker service rm redis
```

To restart a paused update run *sudo docker service update redis*

```
sri123@onlyforGlusty:~$ sudo docker service update --image redis:3.0.7 redis
redis
overall progress: 0 out of 3 tasks
1/3: preparing [=================================>                 ]
2/3:
3/3:
service update paused: update paused due to failure or early termination of task gf8rpahgo50nc0elljn1ho72p
sri123@onlyforGlusty:~$ sudo docker service update redis
redis
overall progress: 2 out of 3 tasks
1/3: preparing [=================================>                 ]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
sri123@onlyforGlusty:~$
```

To get full details on all replicas, we run *sudo docker service ps redis*

```
sri123@onlyforGlusty:~$ sudo docker service ps redis
ID             NAME          IMAGE         NODE            DESIRED STATE   CURRENT STATE             ERROR     PORTS
4z3v4tzbp8a7   redis.1       redis:3.0.6   onlyforGlusty   Running         Running 40 seconds ago
gf8rpahgo50n   redis.2       redis:3.0.7   bhuvana         Running         Preparing 26 seconds ago
2vqi7quhy834    \_ redis.2   redis:3.0.6   bhuvana         Shutdown        Shutdown 25 seconds ago
evy2fds9sgal   redis.3       redis:3.0.6   hemanth         Running         Running 42 seconds ago
sri123@onlyforGlusty:~$ sudo docker service ps redis
ID             NAME          IMAGE         NODE            DESIRED STATE   CURRENT STATE                ERROR
        PORTS
y52pihqmvq6r   redis.1       redis:3.0.7   onlyforGlusty   Running         Running 30 seconds ago

4z3v4tzbp8a7    \_ redis.1   redis:3.0.6   onlyforGlusty   Shutdown        Shutdown 31 seconds ago

1yebqhruafxs   redis.2       redis:3.0.7   bhuvana         Running         Preparing 18 seconds ago

ex02ahlg5mz9    \_ redis.2   redis:3.0.7   bhuvana         Shutdown        Rejected 18 seconds ago      "No such image: redis:3.0.7
@sh…"
gf8rpahgo50n    \_ redis.2   redis:3.0.7   bhuvana         Shutdown        Rejected 54 seconds ago      "No such image: redis:3.0.7
@sh…"
2vqi7quhy834    \_ redis.2   redis:3.0.6   bhuvana         Shutdown        Shutdown about a minute ago

8cqp5uwwrstl   redis.3       redis:3.0.7   hemanth         Running         Running 19 seconds ago

evy2fds9sgal    \_ redis.3   redis:3.0.6   hemanth         Shutdown        Shutdown 20 seconds ago
```

Drain a node

The swarm manager can assign tasks to any Active node, so up to now all nodes have been available to receive tasks. Sometimes, such as planned maintenance times, you need to set a node to Drain availability. Drain availability prevents a node from receiving new tasks from the swarm manager.

Continuing the above rolling update service, with three replicas, we run *sudo docker service ps redis* to see how the swarm manager assigned the tasks to different nodes:

```
sri123@onlyforGlusty:~$ sudo docker service create --replicas 3 --name redis --update-delay 10s redis:3.0.6
mpg66yl5djxtaexq1wwfohqgx
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker service ps redis
ID             NAME      IMAGE         NODE            DESIRED STATE   CURRENT STATE           ERROR     PORTS
jwulct50h9oo   redis.1   redis:3.0.6   onlyforGlusty   Running         Running 34 seconds ago
y57afns94x6r   redis.2   redis:3.0.6   bhuvana         Running         Running 34 seconds ago
zgdgj5i9b2t8   redis.3   redis:3.0.6   hemanth         Running         Running 35 seconds ago
sri123@onlyforGlusty:~$
```

Command for draining a node say worker1 should be run on the manager node as follows:
sudo docker node update --availability drain worker1

```
sri123@onlyforGlusty:~$ sudo docker node ls
ID                            HOSTNAME        STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
0x8srsf72q8040ylwtok72b5g     bhuvana         Ready     Active                          25.0.3
p2gom128fnnlro3cmzlevqkyt     hemanth         Ready     Active                          25.0.3
wibcftute07kwmcdtj0b6m8gw *   onlyforGlusty   Ready     Active         Leader           25.0.2
sri123@onlyforGlusty:~$ sudo docker node update --availability drain bhuvana
bhuvana
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS      NAMES
5fd99a0630ca   redis:3.0.6   "/entrypoint.sh redi…"   3 minutes ago   Up 3 minutes   6379/tcp   redis.1.jwulct50h9oo4btmgd5pdkgbu
sri123@onlyforGlusty:~$
```

```
hemanth@hemanth:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS      NAMES
b232dbabf031   redis:3.0.6   "/entrypoint.sh redi…"   10 seconds ago   Up 9 seconds    6379/tcp   redis.3.zgdgj5i9b2t8nhinthvztccrc
hemanth@hemanth:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS      NAMES
0d85c0f3d7e4   redis:3.0.6   "/entrypoint.sh redi…"   17 seconds ago   Up 15 seconds   6379/tcp   redis.2.uzmxmy2vspxalf5djh5sxl85n
b232dbabf031   redis:3.0.6   "/entrypoint.sh redi…"   3 minutes ago    Up 3 minutes    6379/tcp   redis.3.zgdgj5i9b2t8nhinthvztccrc
```

Check the service status updated by manager by *sudo docker service ps redis*

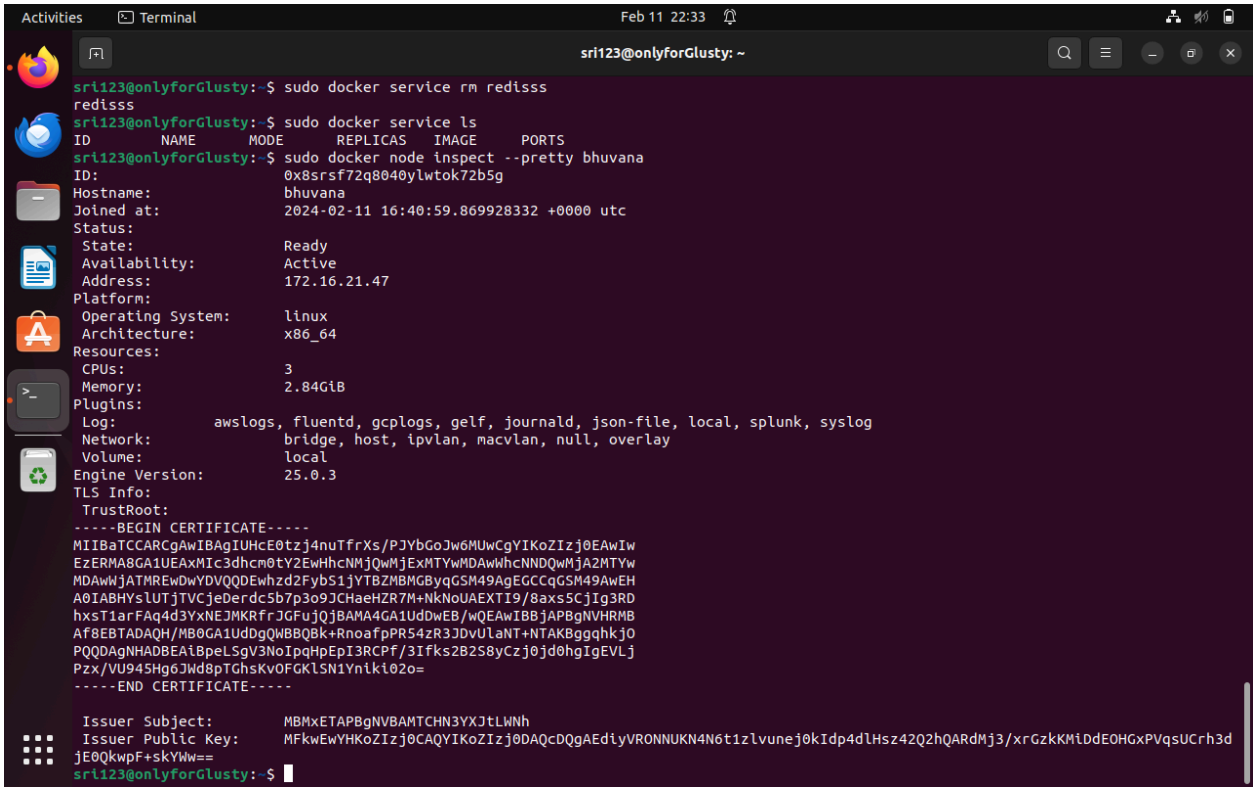
```
sri123@onlyforGlusty:~$ sudo docker service ps redis
ID             NAME          IMAGE         NODE            DESIRED STATE   CURRENT STATE                ERROR     PORTS
jwulct50h9oo   redis.1       redis:3.0.6   onlyforGlusty   Running         Running 5 minutes ago
uzmxmy2vspxa   redis.2       redis:3.0.6   hemanth         Running         Running about a minute ago
y57afns94x6r    \_ redis.2   redis:3.0.6   bhuvana         Shutdown        Shutdown about a minute ago
zgdgj5i9b2t8   redis.3       redis:3.0.6   hemanth         Running         Running 5 minutes ago
sri123@onlyforGlusty:~$
```

At this point we have drained *bhuvana,* hence the processes running on it were shifted to another worker.

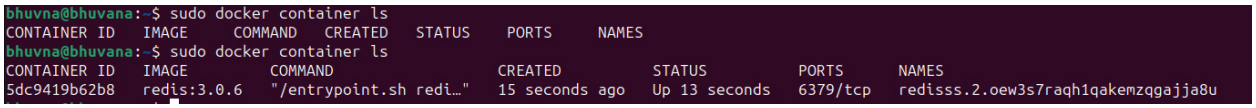
Let us make worker1 available again: *sudo docker node update --availability active worker1*

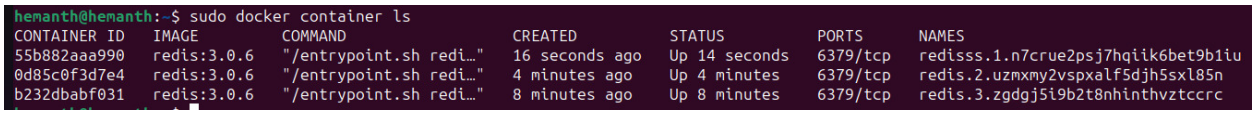
```
sri123@onlyforGlusty:~$ sudo docker node ls
ID                            HOSTNAME        STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
0x8srsf72q8040ylwtok72b5g     bhuvana         Ready     Drain                           25.0.3
p2gom128fnnlro3cmzlevqkyt     hemanth         Ready     Active                          25.0.3
wibcftute07kwmcdtj0b6m8gw *   onlyforGlusty   Ready     Active         Leader           25.0.2
sri123@onlyforGlusty:~$ sudo docker node update --availability active bhuvana
bhuvana
sri123@onlyforGlusty:~$ sudo docker node ls
ID                            HOSTNAME        STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
0x8srsf72q8040ylwtok72b5g     bhuvana         Ready     Active                          25.0.3
p2gom128fnnlro3cmzlevqkyt     hemanth         Ready     Active                          25.0.3
wibcftute07kwmcdtj0b6m8gw *   onlyforGlusty   Ready     Active         Leader           25.0.2
sri123@onlyforGlusty:~$ sudo docker service create --replicas 3 --name redisss --update-delay 10s redis:3.0.6
xv4fvuw4uxovodxqy2zv15a8o
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE         COMMAND                  CREATED          STATUS          PORTS      NAMES
66f736d89d76   redis:3.0.6   "/entrypoint.sh redi…"   35 seconds ago   Up 34 seconds   6379/tcp   redisss.3.msqrnigynrazxjajfnuiwhbs
b
5fd99a0630ca   redis:3.0.6   "/entrypoint.sh redi…"   8 minutes ago    Up 8 minutes    6379/tcp   redis.1.jwulct50h9oo4btmgd5pdkgbu
sri123@onlyforGlusty:~$
```

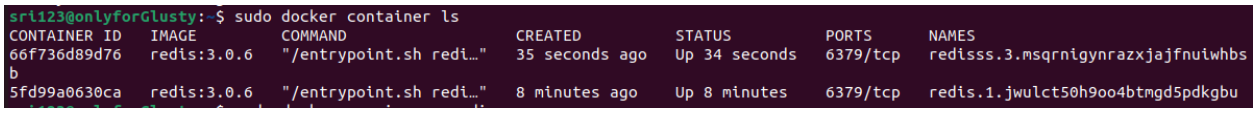
We may inspect worker1 (*sudo docker node inspect --pretty worker1*) to check the status.



After making the availability as Active again, we can see that new processes are assigned to that node as well.







Services in detail and depth

1. To create a single-replica service with no extra configuration, you only need to supply the image name.

   sudo docker service create --name my_ web nginx
   sudo docker service ls

```
sri123@onlyforGlusty:~$ sudo docker service create --name my_web nginx
pid08lbor3uglyijlbh4i1sjq
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
pid08lbor3ug   my_web    replicated   1/1        nginx:latest
sri123@onlyforGlusty:~$
```

2. For standalone servers like alpine we can specify a command that the service's containers should run, by adding it after the image name. We can also specify an image tag for the service to use.

   sudo docker service create --name helloworld alpine ping docker.com
   sudo docker service create --name helloworld alpine:3.6 ping docker.com

```
sri123@onlyforGlusty:~$ sudo docker service create --name helloworld alpine:3.6 ping docker.com
i8qwvdndlwinnb6armeqjnnlb
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker service ls
ID             NAME         MODE         REPLICAS   IMAGE        PORTS
i8qwvdndlwin   helloworld   replicated   1/1        alpine:3.6
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE        COMMAND             CREATED         STATUS         PORTS     NAMES
1c64ffdf5b29   alpine:3.6   "ping docker.com"   2 minutes ago   Up 2 minutes             helloworld.1.0dk3xwqcnj25bvjjcs2c7ksa3
sri123@onlyforGlusty:~$
```

3. gMSA (Group Managed Service Account) credential spec in Docker allows you to specify a Windows group Managed Service Account to run a Windows container with a certain level of security and convenience.

   To use a config as a credential spec, first create the Docker config containing the credential spec: *sudo docker config create credspec credspec.json*

   Now we create a service using credential-spec flag with config:
   sudo docker service create --credential-spec="config://credspec" <your image>
   PIC

4. To create a service using an image on a private registry, we can do the following:

   sudo docker login regName
   *sudo docker service create --with-registry-auth --name my_service *
   regName/acme/my_ image:latest

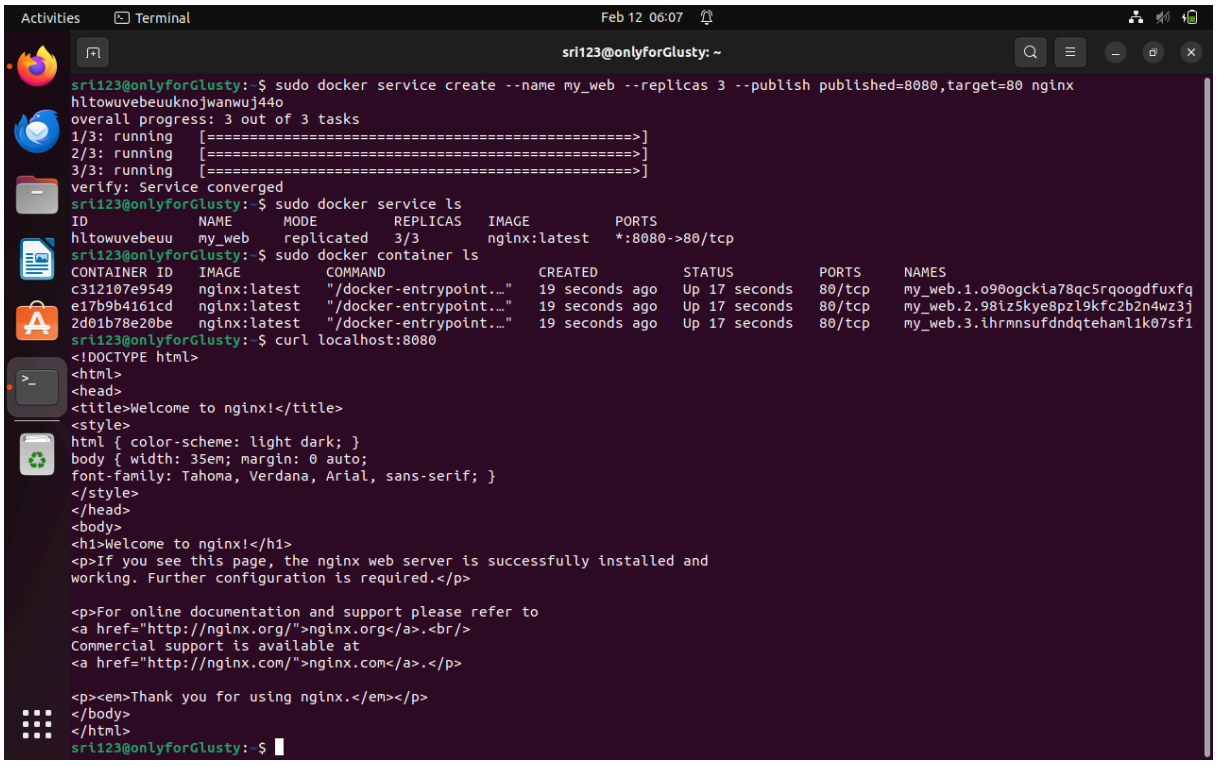
PIC

This passes the login token from your local client to the swarm nodes where the service is deployed and with this information, the nodes are able to log into the registry and pull the image.

5. To update the command an existing service runs, we can use the --args flag:
   sudo docker service update --args "ping docker.com" helloworld

```
sri123@onlyforGlusty:~$ sudo docker service create --name helloworld alpine:3.6 ping docker.com
eoj32sgh8slarju238moytcu8
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE        COMMAND             CREATED          STATUS          PORTS     NAMES
24bc8eb1e748   alpine:3.6   "ping docker.com"   19 seconds ago   Up 19 seconds             helloworld.1.pxbtmbaiiicr0xpyezhkutdlx
sri123@onlyforGlusty:~$ udo docker service update --args "ping google.com" helloworld
Command 'udo' not found, but can be installed with:
sudo apt install udo
sri123@onlyforGlusty:~$ sudo docker service update --args "ping google.com" helloworld
helloworld
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker container ls
CONTAINER ID   IMAGE        COMMAND             CREATED          STATUS         PORTS     NAMES
c4a7383d955c   alpine:3.6   "ping google.com"   20 seconds ago   Up 8 seconds             helloworld.1.tiz7lug1g4tdkx6p53byj4x6n
sri123@onlyforGlusty:~$
```

6. To create a service and publish it to some port we may use the following command:
   *sudo docker service create --name my_ web --replicas 3 --publish *
   published=8080,target=80 nginx
   Now we can test using curl the nodes connecting to localhost 8080:
   curl localhost:8080



7. In order to run a service on every swarm node, we may use the flag --mode:

```
sudo docker service create \
 --mode global \
 --publish mode=host,target=80,published=8080 \
 --name=nginx \
 nginx:latest
```

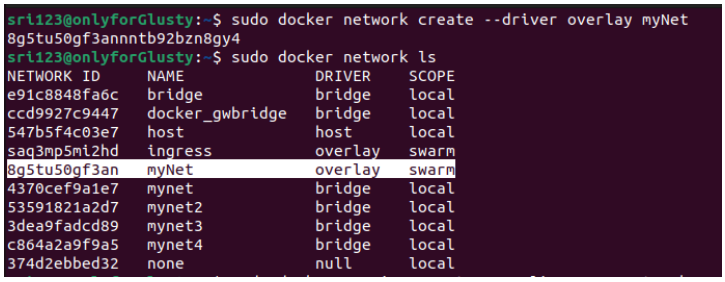
PIC

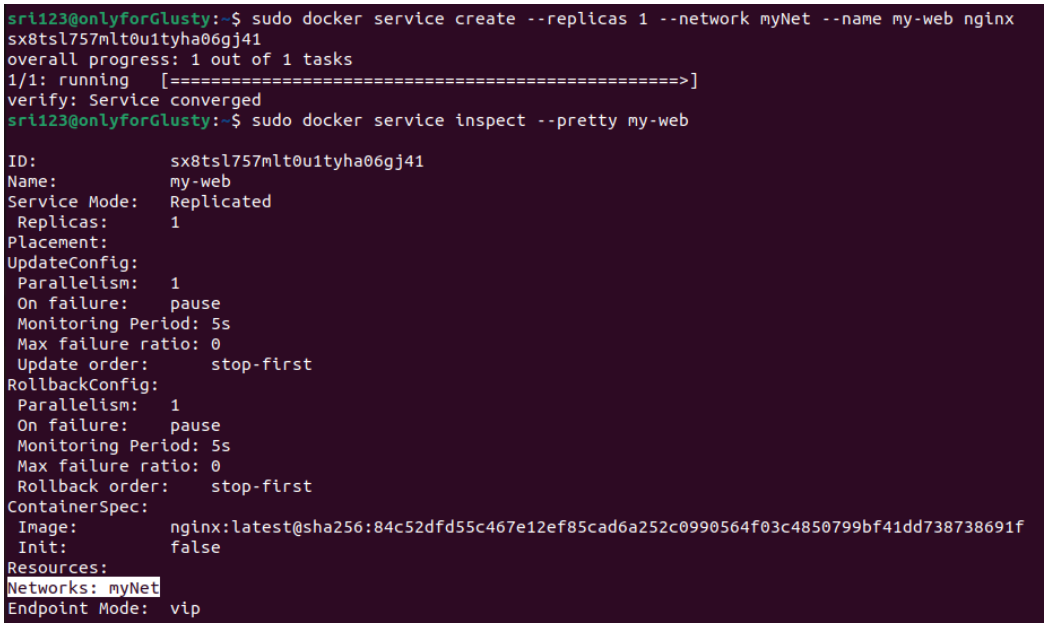
Now we can reach the nginx server on port 8080 of every swarm node. Even if we add a node to the swarm, a nginx task is started on it.

8. To connect the service to an overlay network we can create overlay network on a manager node using *sudo docker network create --driver overlay my-network*

   Then we create a service:

   sudo docker service create --replicas 3 --network my-network --name my-web nginx





   Or to connect an existing running service, we may use:

   sudo docker service update --network-add my-network my-web

To disconnect a network from running service we use:

sudo docker service update --network-rm my-network my-web

```
sri123@onlyforGlusty:~$ sudo docker service update --network-add ingress my-web
Error response from daemon: rpc error: code = InvalidArgument desc = Service cannot be explicitly attached to the ingress network "i
ngress"
sri123@onlyforGlusty:~$ sudo docker service update --network-rm myNet my-web
my-web
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker service inspect --pretty my-web

ID:             sx8tsl757mlt0u1tyha06gj41
Name:           my-web
Service Mode:   Replicated
 Replicas:      1
UpdateStatus:
 State:         completed
 Started:       13 seconds ago
 Completed:     4 seconds ago
 Message:       update completed
Placement:
UpdateConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         nginx:latest@sha256:84c52dfd55c467e12ef85cad6a252c0990564f03c4850799bf41dd738738691f
 Init:          false
Resources:
Endpoint Mode:  vip
```

9. We can configure the update behavior as we want. For example:

   sudo docker service update --update-parallelism 2 --update-failure-action continue my-web

```
sri123@onlyforGlusty:~$ sudo docker service update --update-parallelism 2 --update-failure-action continue my-web
my-web
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
sri123@onlyforGlusty:~$ sudo docker service inspect --pretty my-web

ID:             mizvr3t7qtpd6r0l3t70dcr8b
Name:           my-web
Service Mode:   Replicated
 Replicas:      1
Placement:
UpdateConfig:
 Parallelism:   2
 On failure:    continue
 Monitoring Period: 5s
 Max failure ratio: 0
 Update order:      stop-first
RollbackConfig:
 Parallelism:   1
 On failure:    pause
 Monitoring Period: 5s
 Max failure ratio: 0
 Rollback order:    stop-first
ContainerSpec:
 Image:         nginx:latest@sha256:84c52dfd55c467e12ef85cad6a252c0990564f03c4850799bf41dd738738691f
 Init:          false
Resources:
Endpoint Mode:  vip
```

The ***--update-delay*** flag configures the time delay between updates to a service task or sets of tasks. The ***--update-parallelism*** flag to configure the maximum number of service tasks that the scheduler updates simultaneously. The ***--update-failure-action*** flag specifies the action to take if an update to the service fails, in this case it is ***continue*.** Possible options are : *pause, continue* or *rollback.*