Demo 3: Docker Swarm

Introduction

The learning objectives of this demonstration are as follows:

- Setting up a two-node Swarm cluster
- Creating, updating and deleting Swarm services
- Replica and global mode of services
- Updating a service
- Overlaying network and routing across multiple nodes
- Setting up the following voting application using Swarm: https://github.com/dockersamples/example-voting-app

In this demonstration, we will first launch a swarm cluster and then explore different swarm commands one by one.

Prerequisites:

- Launch two EC2 instances, one of which will be used to initialise the swarm cluster and will be assigned as manager. Another will be called a worker. **Keep port 22, 80, 8080 and 443** open for all IPs to access the instances from your local machine over the internet.

- Docker swarm requires **TCP port 2376, TCP port 2377, TCP & UDP port 7946 and UDP port 4789** to communicate among nodes. Keep all ports open for the same security group.

Setting up a Two-Node Swarm Cluster

1. Run the following command from the manager node :

   #To initialise swarm cluster and become manager node

   docker swarm init --advertise-addr <IP>

   # A token will appear as a output;copy that

   #To join as worker node run following command in the terminal of the worker nodes

   **docker swarm join --token \
   SWMTKN-1-3phyd9eyv1j9rh1mn72abf0cdoqupjp933zl6e60fe9f7vc88u-06m5uap**

#from manager node run this command to find the list of nodes attached to the cluster
docker node ls

Now, run the series of commands under each heading from the terminal of the manager node and check the output at every step. Also, explore what each command does.

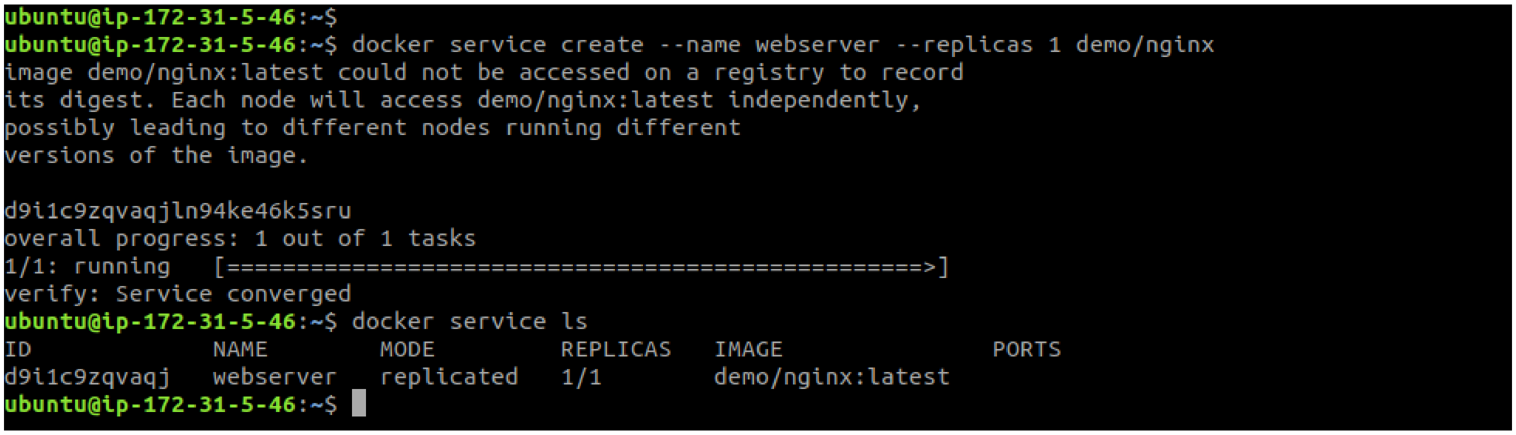
Creating, Updating and Deleting Swarm Services

Now, we will create a service, list all the services and list all the containers running under the service:

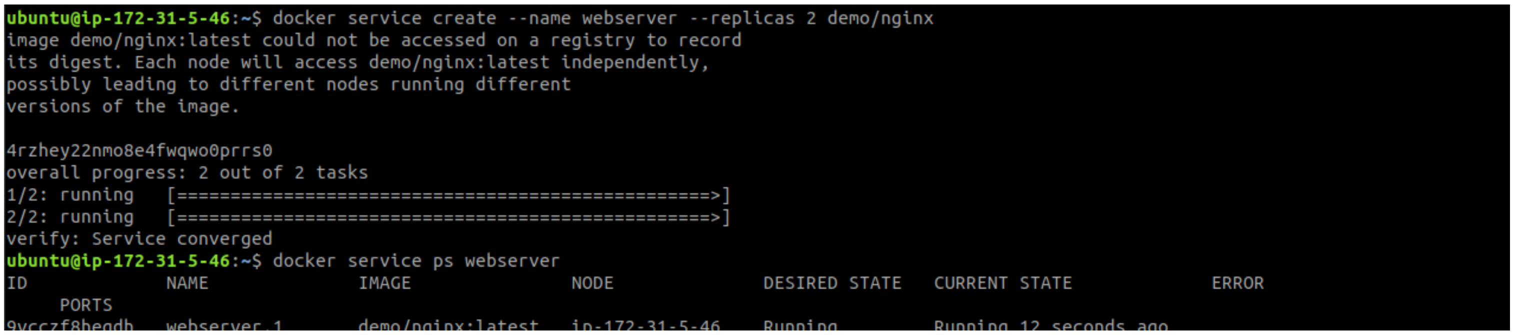
1. List all the services deployed on the cluster **docker service ls.**
2. Create a service (replica=1) with the name "webserver" and image 'nginx' using following command:
   docker service create --name webserver --replicas 1 nginx
3. Again, list all the services deployed on the cluster **docker service ls.**
4. Check information about 'webserver' service using following command:
   docker service ps webserver #Info about specific service
5. Now, remove the service using the following command:
   docker service rm webserver #remove service

Now, let's deploy the **demo/nginx** image, which we built previously in the demo
docker service create --name webserver --replicas 2 demo/nginx.

6. Swarm will not be able to place the container on the second node because the image is only available locally at the first node. This highlights the importance of image registries such as Dockerhub or any local image registry available to all the nodes.



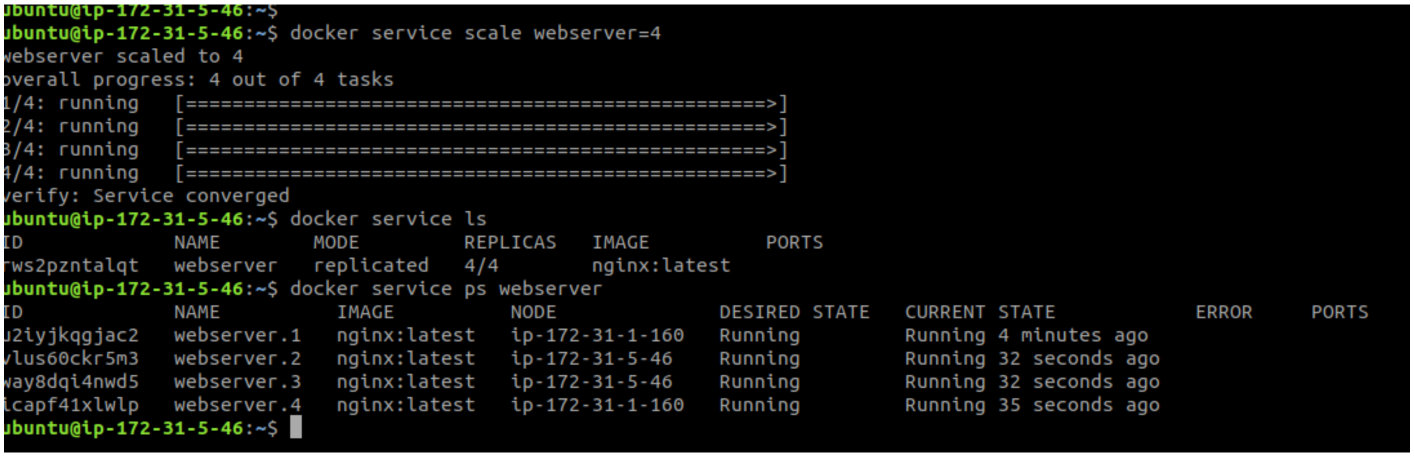
```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service create --name webserver --replicas 1 demo/nginx
image demo/nginx:latest could not be accessed on a registry to record
its digest. Each node will access demo/nginx:latest independently,
possibly leading to different nodes running different
versions of the image.

d9i1c9zqvaqjln94ke46k5sru
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-46:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE                PORTS
d9i1c9zqvaqj   webserver   replicated   1/1        demo/nginx:latest
ubuntu@ip-172-31-5-46:~$
```



```
ubuntu@ip-172-31-5-46:~$ docker service create --name webserver --replicas 2 demo/nginx
image demo/nginx:latest could not be accessed on a registry to record
its digest. Each node will access demo/nginx:latest independently,
possibly leading to different nodes running different
versions of the image.

4rzhey22nmo8e4fwqwo0prrs0
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID             NAME              IMAGE               NODE               DESIRED STATE   CURRENT STATE            ERROR
     PORTS
9vcczf8heqdh   webserver.1       demo/nginx:latest   ip-172-31-5-46     Running         Running 12 seconds ago
```

7. Now, let’s pull the nginx image from the docker hub and then run the following command again in the terminal of the manager node using the command -**docker service create --name webserver --replicas 2 nginx.**
8. Run the following command to get more details about the ‘webserver’ service: **docker service ps webserver**
9. Let’s now manually stop one of the containers of the ‘webserver’ service using the command **docker stop <id>.** Now, check the number of instances of ‘webserver’ by running the command **docker service ps webserver**. You will observe that the swarm service maintains availability by deploying new containers.
10. Run **systemctl stop docker** in one worker node to stop the docker service on that node and again run the command **docker service ps webserver**. You will find that the swarm service will maintain the number of tasks by redistributing containers on the remaining nodes.
11. Let’s now scale up the ‘webserver’ service using the command **docker service scale webserver=4.**

12. Let's look at the details of 'webserver' service using following commands:
    docker service ps webserver



```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service scale webserver=4
webserver 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
ubuntu@ip-172-31-5-46:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE          PORTS
rws2pzntalqt   webserver   replicated   4/4        nginx:latest
ubuntu@ip-172-31-5-46:~$ docker service ps webserver
ID             NAME          IMAGE          NODE            DESIRED STATE   CURRENT STATE            ERROR     PORTS
u2iyjkqgjac2   webserver.1   nginx:latest   ip-172-31-1-160   Running         Running 4 minutes ago
vlus60ckr5m3   webserver.2   nginx:latest   ip-172-31-5-46    Running         Running 32 seconds ago
way8dqi4nwd5   webserver.3   nginx:latest   ip-172-31-5-46    Running         Running 32 seconds ago
icapf41xlwlp   webserver.4   nginx:latest   ip-172-31-1-160   Running         Running 35 seconds ago
ubuntu@ip-172-31-5-46:~$
```

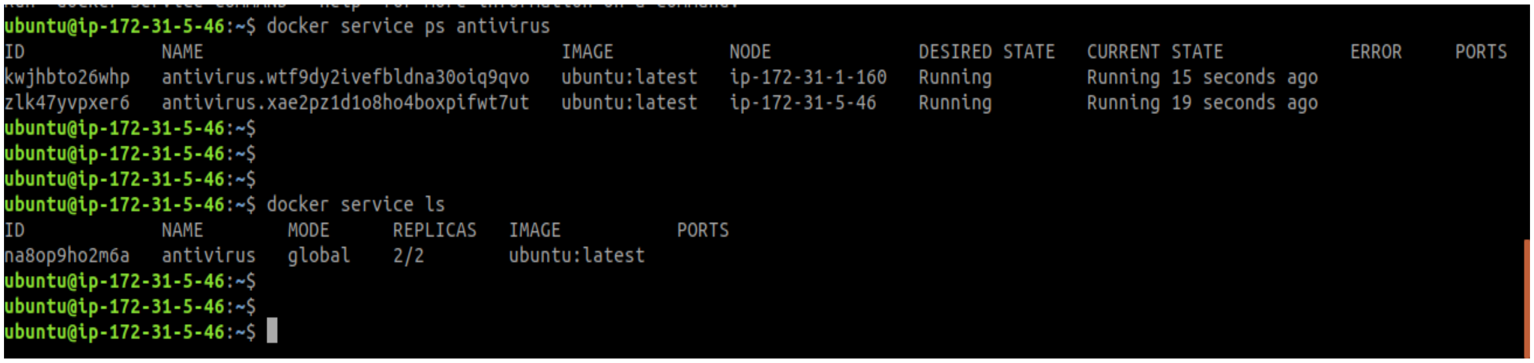
Creating Replica and Global Services

We will demonstrate different modes of services. Deploy the nginx service with one replica. Also, deploy another service with the name 'antivirus' and image 'ubuntu' in the global mode. You will find that one replica of 'antivirus' will get deployed in each node.

Run the following commands to create two services in replica mode and global mode respectively:

1. docker service create replica --replicas 1 nginx
2. docker service create --name antivirus --mode global -dt ubuntu
3. docker service ps antivirus

(The global mode will deploy one container in each node.)



```
ubuntu@ip-172-31-5-46:~$ docker service ps antivirus
ID             NAME                                     IMAGE           NODE              DESIRED STATE   CURRENT STATE            ERROR     PORTS
kwjhbto26whp   antivirus.wtf9dy2ivefbldna30oiq9qvo   ubuntu:latest   ip-172-31-1-160   Running         Running 15 seconds ago
zlk47yvpxer6   antivirus.xae2pz1d1o8ho4boxpifwt7ut   ubuntu:latest   ip-172-31-5-46    Running         Running 19 seconds ago
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service ls
ID             NAME        MODE     REPLICAS   IMAGE           PORTS
na8op9ho2m6a   antivirus   global   2/2        ubuntu:latest
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
```

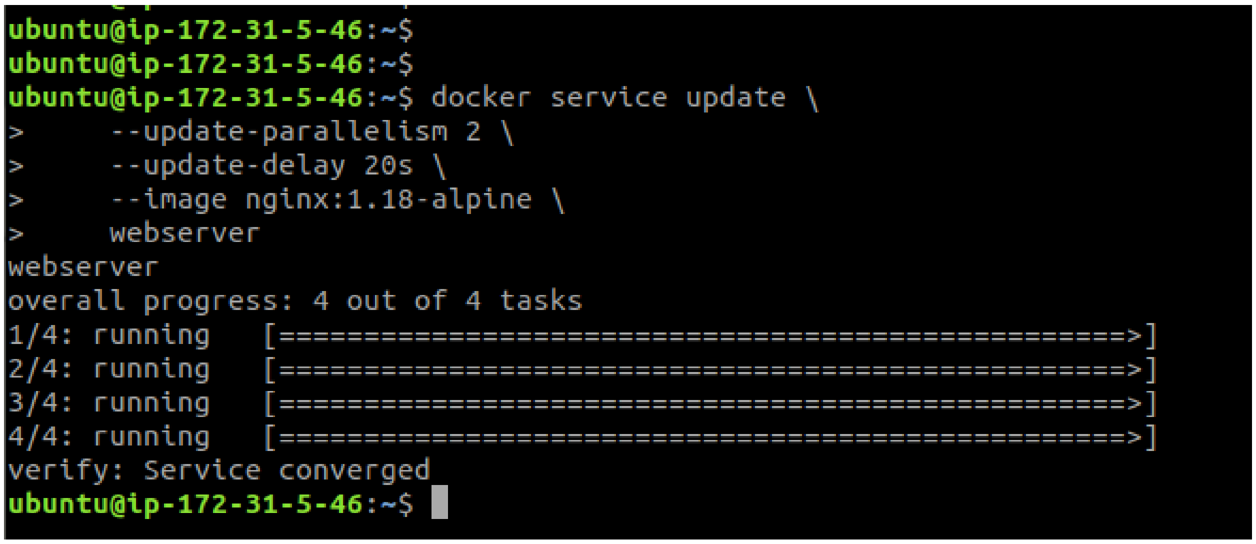
Updating a Service

We will first create a service and then scale it. Then, we will update the image of an already deployed service. Some important points to note here are as follows:

- Parallelism sets the number of containers to be updated simultaneously.
- Update delay sets the time duration after which the next set of containers will get updated.

Now, let's take a look at the commands to be used to update the swarm service. The following commands have to be run in the terminal of the manager node:

1. docker service create --name webserver --replicas 1 nginx
2. docker service scale webserver=4
3. docker service update \
   --update-parallelism 2 \
   --update-delay 20s \
   --image nginx:1.18-alpine \
   webserver



```
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service update \
>     --update-parallelism 2 \
>     --update-delay 20s \
>     --image nginx:1.18-alpine \
>     webserver
webserver
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-46:~$
```

Overlay Network

We will create a custom overlay network and attach services to the network. We will also demonstrate using the 'curl' command so that containers connected over the same network can communicate with other containers in the cluster using service names.

1. Docker swarms, by default, create an overlay network across the nodes. Swarm service containers can communicate across nodes. Run the command **docker network ls** to view the networks.
2. You can create your own network using the following command:

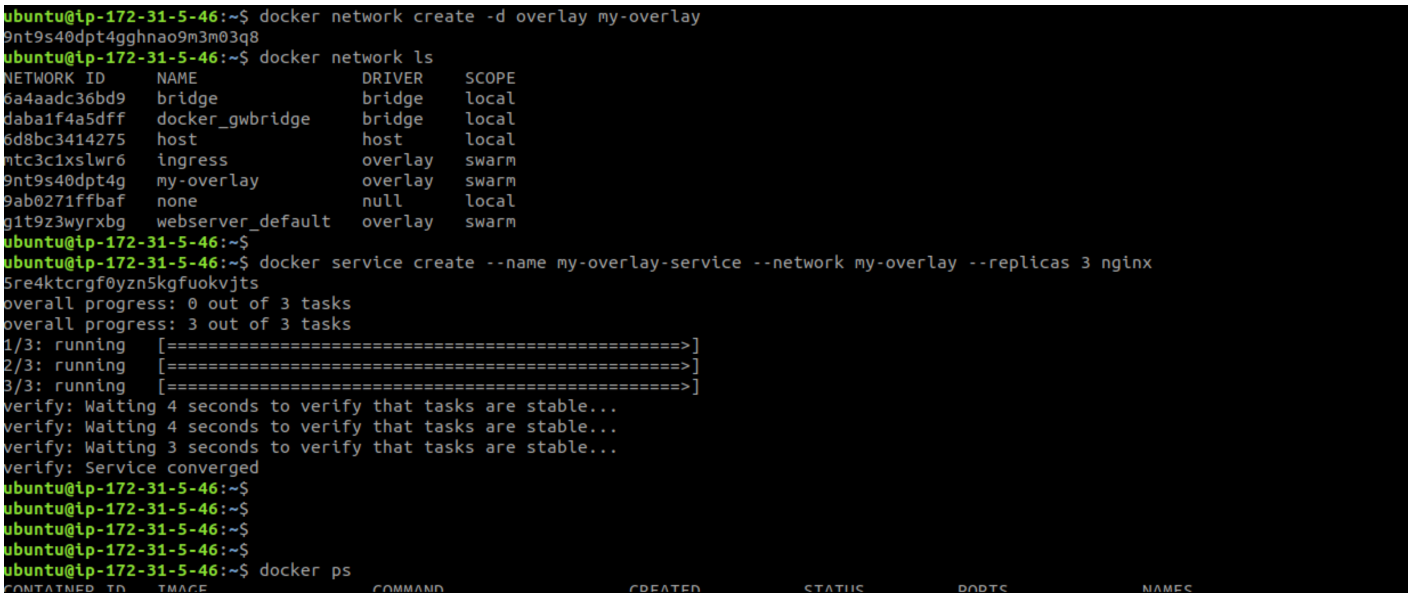
   docker network create -d overlay my-overlay

3. You can create a service and attach it to a network.

   docker service create --name my-overlay-service --network my-overlay --replicas 3 nginx

4. Now, let's find the IP of any container.

   Run **docker ps** on the first node and get the container id, and then, run **docker inspect id** and then find the container IP.

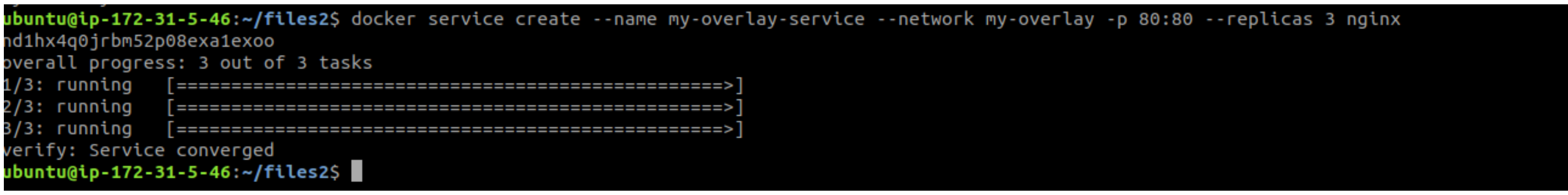


```
ubuntu@ip-172-31-5-46:~$ docker network create -d overlay my-overlay
9nt9s40dpt4gghnao9m3m03q8
ubuntu@ip-172-31-5-46:~$ docker network ls
NETWORK ID     NAME                DRIVER    SCOPE
6a4aadc36bd9   bridge              bridge    local
daba1f4a5dff   docker_gwbridge     bridge    local
6d8bc3414275   host                host      local
mtc3c1xslwr6   ingress             overlay   swarm
9nt9s40dpt4g   my-overlay          overlay   swarm
9ab0271ffbaf   none                null      local
g1t9z3wyrxbg   webserver_default   overlay   swarm
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker service create --name my-overlay-service --network my-overlay --replicas 3 nginx
5re4ktcrgf0yzn5kgfuokvjts
overall progress: 0 out of 3 tasks
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Waiting 4 seconds to verify that tasks are stable...
verify: Waiting 4 seconds to verify that tasks are stable...
verify: Waiting 3 seconds to verify that tasks are stable...
verify: Service converged
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$
ubuntu@ip-172-31-5-46:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                   CREATED           STATUS          PORTS           NAMES
```

5. On the second node, get into the container and run the command **ping <container ip>** (first run **apt-get install iputils-ping**).
6. Create a nginx service with three replicas, attach the service to **my-overlay-network** and publish the service’s port 80 outside the swarm as 8080.

   docker service create --name my-overlay-service --network my-overlay -p 8080:80 --replicas 3 nginx

7. The three tasks of the nginx service get distributed between the two nodes. You do not need to know which nodes are running which tasks; connecting to port 8080 on any of the two nodes connects you to one of the three nginx tasks. You can test this using curl.



```
ubuntu@ip-172-31-5-46:~/files2$ docker service create --name my-overlay-service --network my-overlay -p 80:80 --replicas 3 nginx
nd1hx4q0jrbm52p08exa1exoo
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
ubuntu@ip-172-31-5-46:~/files2$
```

8. Hit <any node’s public ip>:8080 from your browser; you will get the Nginx welcome page from any of the tasks running on the cluster, irrespective of the node you hit.

Deploying the Docker Voting Application on a Docker Swarm

Now, we will deploy a voting application that contains five microservices using the docker swarm stack file. We will create two overlay networks and attach respective services to those networks, and also expose the only frontend services to the host port. We will set environment variables, and restart and update policies in the stack file itself.

Use this stack file to deploy the application using a docker swarm. Remember that a docker swarm does not build the images, unlike docker compose, so your nodes will be pulling images that are already present in the docker hub repository.

You can deploy all the docker swarm services using a single command as the one given below.

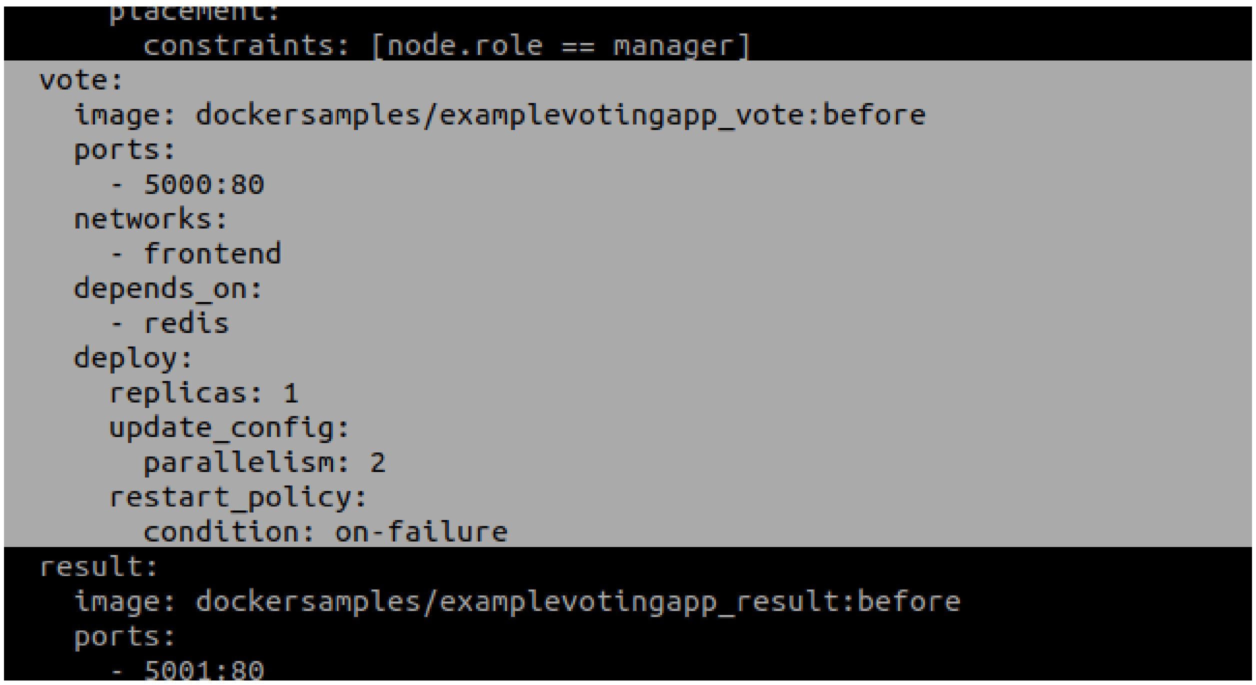
docker stack deploy --compose-file docker-compose.yml <Stack Name>

docker-compose.yml is a stack file written in YAML. It is similar to a compose file, but some features are not supported while deploying using the docker-compose command (such as the "deploy" keyword). Similarly, the "built" keyword is not supported by a docker swarm and is ignored if present in a YAMLfile.

docker-stack-simple.yml

```
version: "3"
services:

  redis:
    image: redis:alpine
    ports:
      - "6379"
    networks:
      - frontend
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
        delay: 10s
      restart_policy:
        condition: on-failure
  db:
    image: postgres:9.4
    environment:
      POSTGRES_USER: "postgres"
      POSTGRES_PASSWORD: "postgres"
  volumes:
    - db-data:/var/lib/postgresql/data
  networks:
    - backend
  deploy:

vote:
  image: dockersamples/examplevotingapp_vote:before
  ports:
    - 5000:80
  networks:
    - frontend
  depends_on:
    - redis
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
    restart_policy:
      condition: on-failure
result:
  image: dockersamples/examplevotingapp_result:before
  ports:
    - 5001:80
  networks:
    - backend
  depends_on:
    - db
  deploy:
    replicas: 1
    update_config:
      parallelism: 2
      delay: 10s
    restart_policy:
      condition: on-failure

worker:
  image: dockersamples/examplevotingapp_worker
  networks:
    - frontend
    - backend
  depends_on:
    - db
```

```
      - redis
    deploy:
      mode: replicated
      replicas: 1
      labels: [APP=VOTING]
      restart_policy:
        condition: on-failure

networks:
  frontend:
  backend:

volumes:
  db-data:
```



```
      placement:
        constraints: [node.role == manager]
  vote:
    image: dockersamples/examplevotingapp_vote:before
    ports:
      - 5000:80
    networks:
      - frontend
    depends_on:
      - redis
    deploy:
      replicas: 1
      update_config:
        parallelism: 2
      restart_policy:
        condition: on-failure
  result:
    image: dockersamples/examplevotingapp_result:before
    ports:
      - 5001:80
```