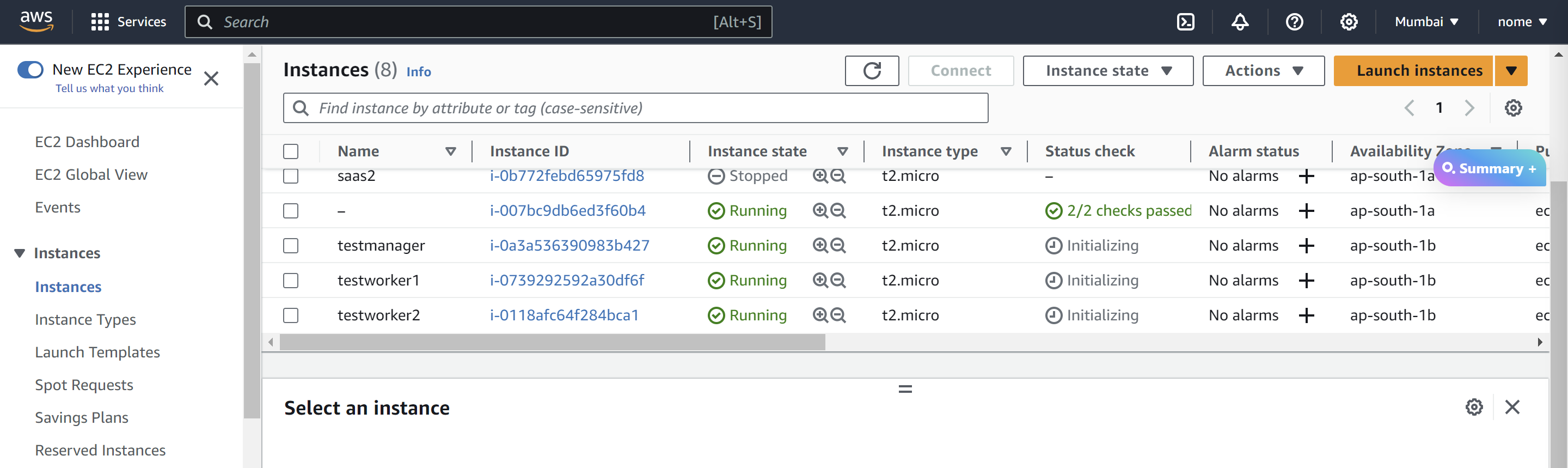
**Docker Swarm**

Ques-1: Do the following:

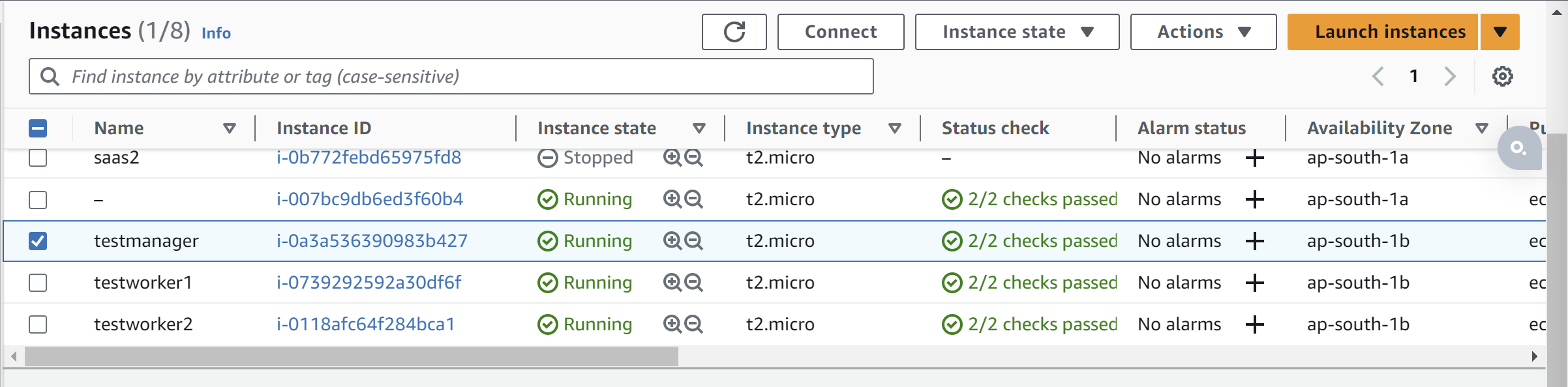
1. Create 3 linux machines on AWS.
2. Install docker and check swarm is working on it or not.
3. Make first instance as Manager.
4. Make second and third machine as worker machines.
5. Check the current status of the swarm.
6. View the information of all the nodes.

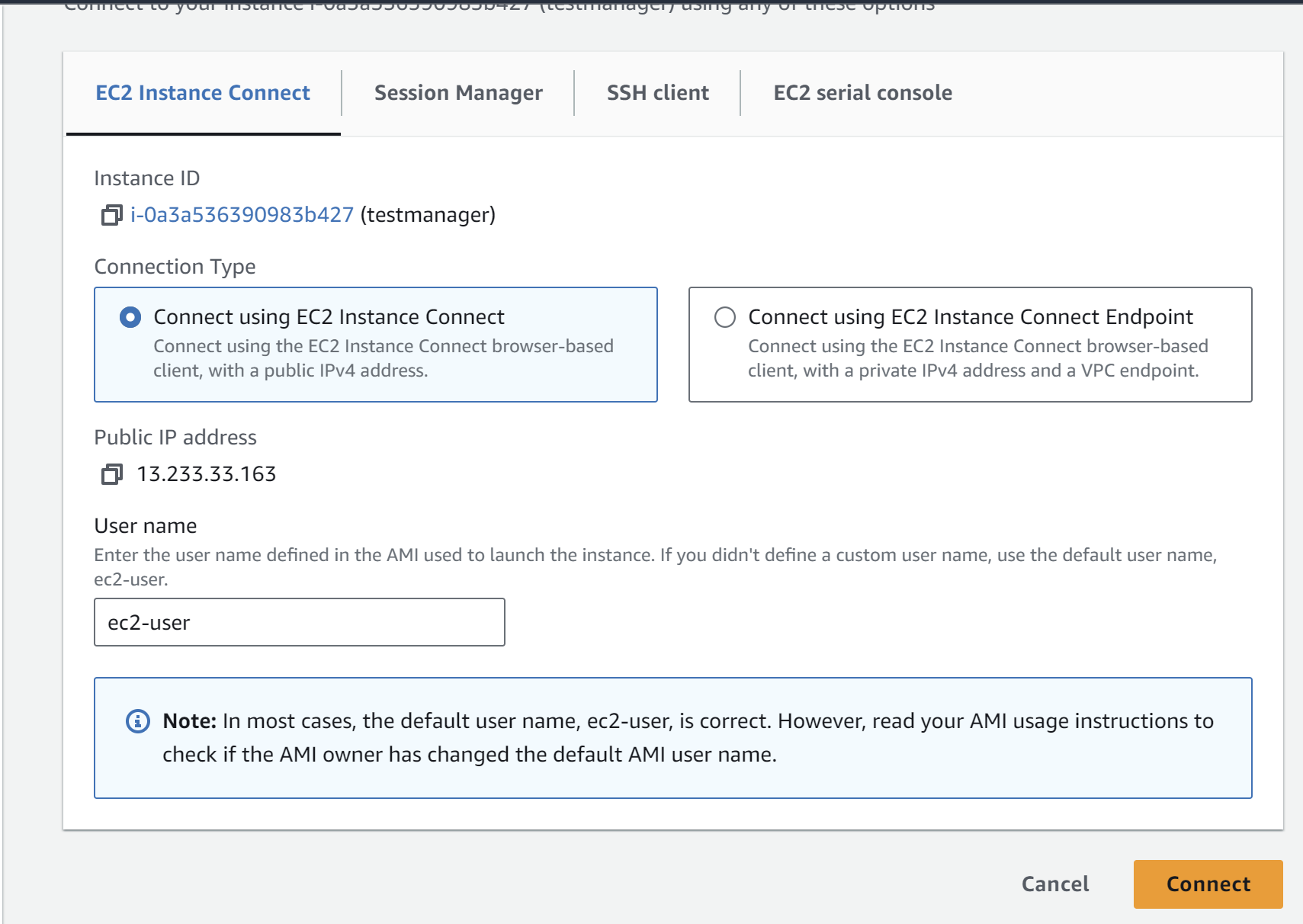
**Step-1:** Go to aws management console and create three EC2 linux instances.



**Note:** Make sure you add “All TCP” and “All UDP” in the security groups of all the three instances.

**Step-2:** Now select the “testmanager” instance and click on connect.

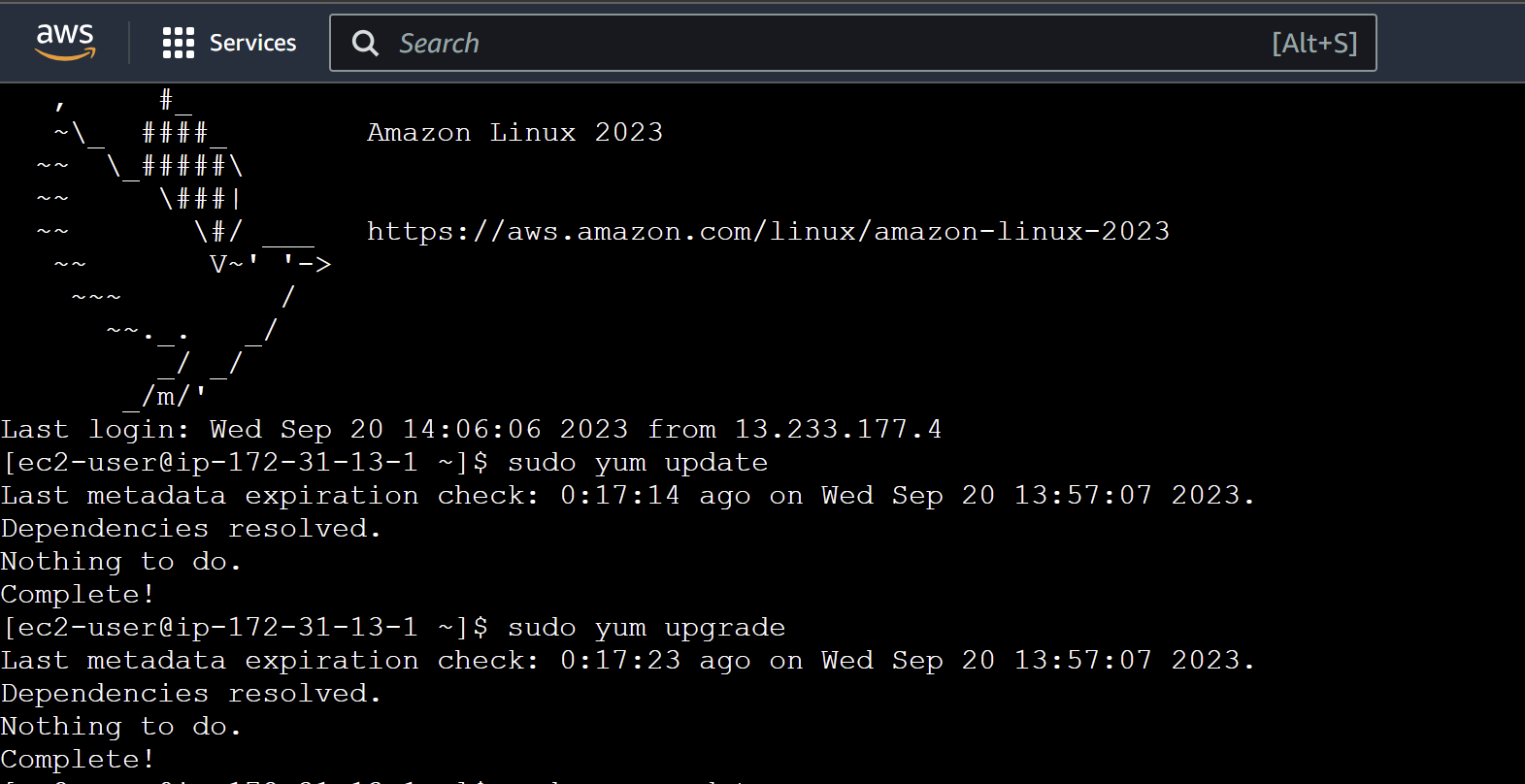
****

****

**Step-3:** After connecting to it successfully, update and upgrade the linux instance

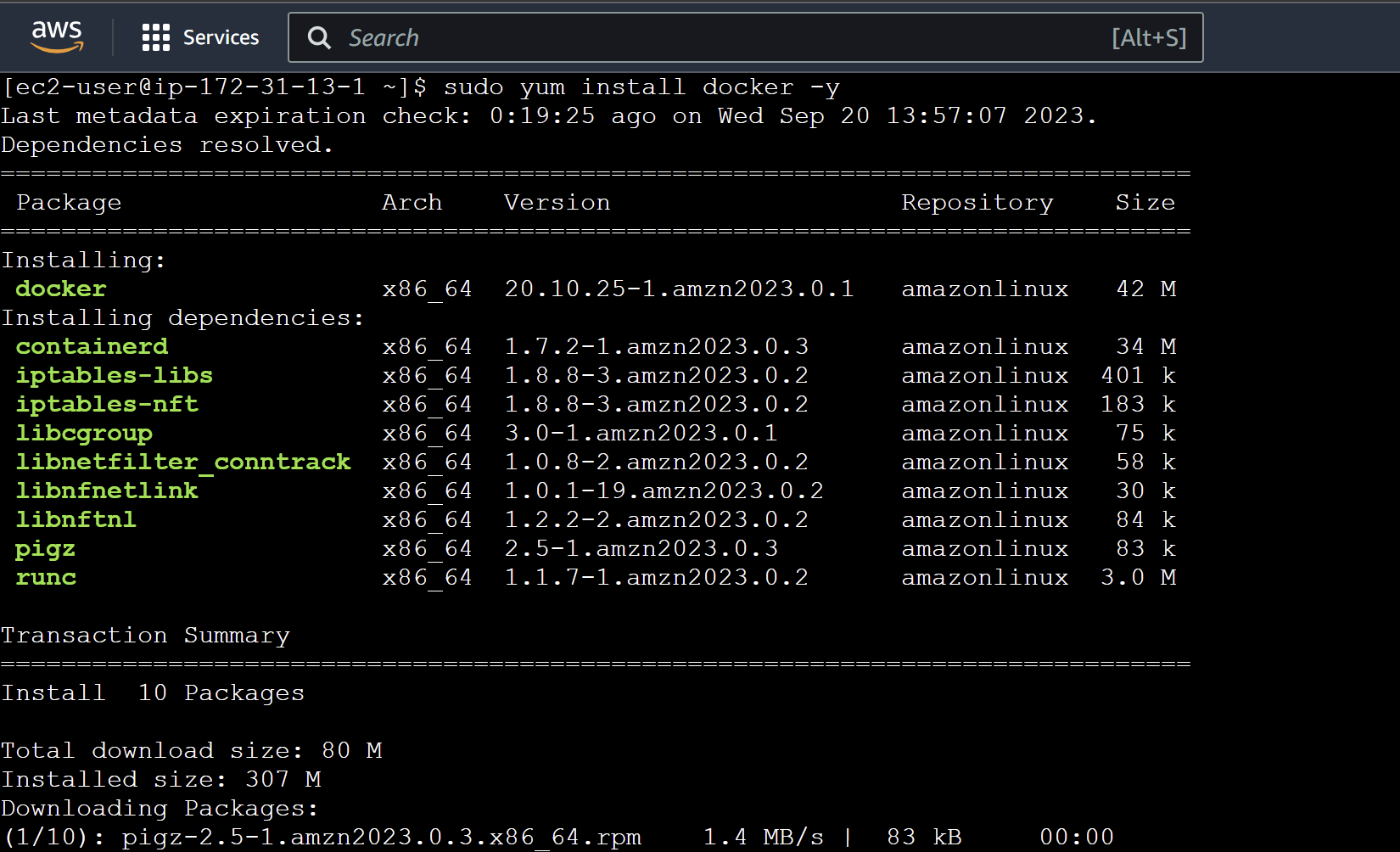
sudo yum update

sudo yum upgrade



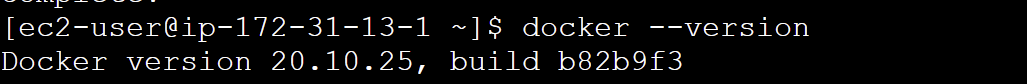
**Step-4:** Install docker in the instance.

sudo yum install docker -y



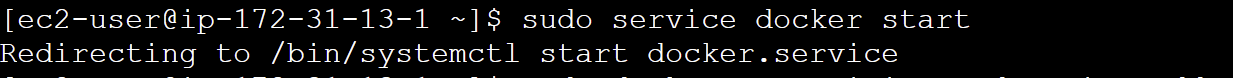
**Step-5:** Check the docker version and check whether it is installed successfully or not.

docker --version

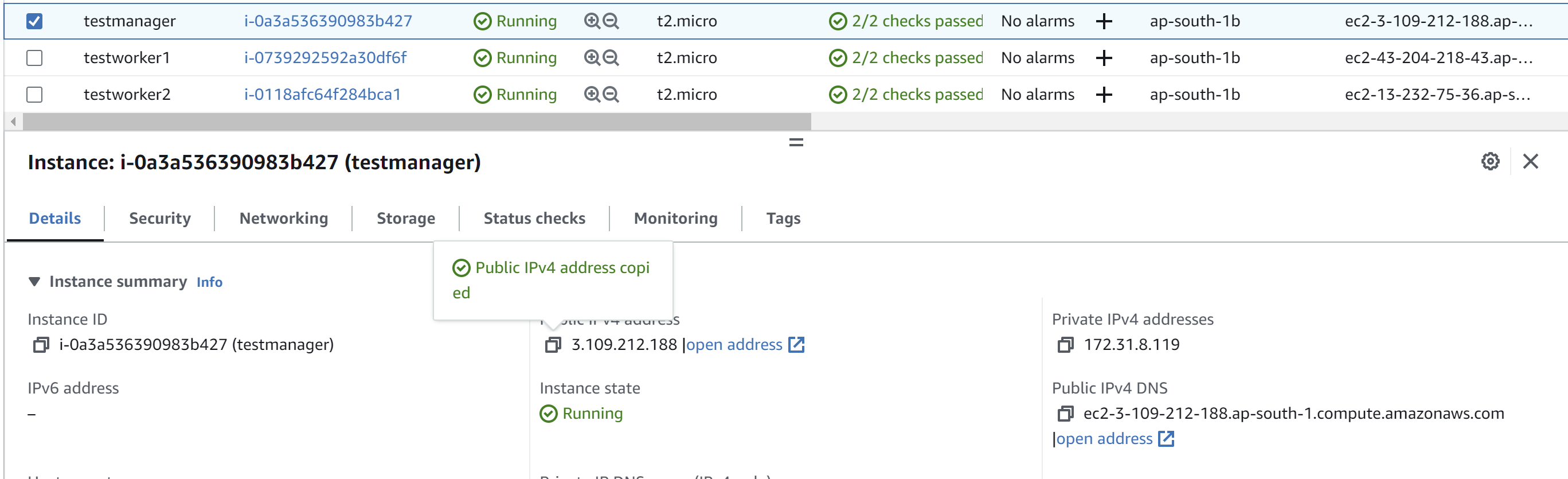


**Step-6:** Start the docker service.

sudo service docker start

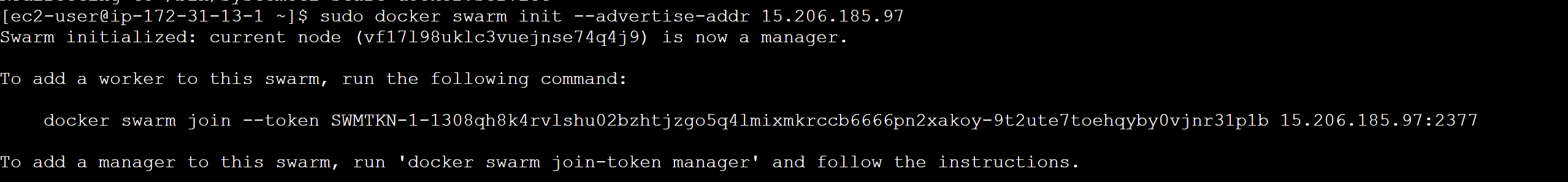


**Step-7:** Select the testmanager and then copy its “public IP”.



**Step-8:** Now initialise swarm and paste the IP address of the manager node that you copied in the following command:

sudo docker swarm init - -advertise-addr <public-ip-of-same-machine>

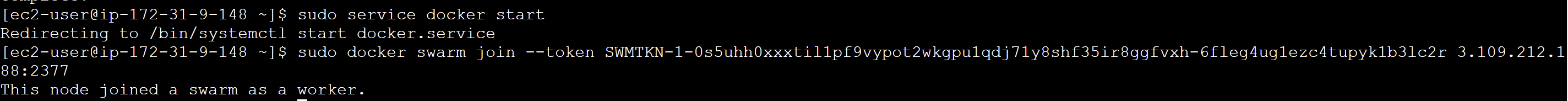


Here copy the docker swarm join command.

**Step-9:** Go to the worker instances and repeat the steps from 1-6 in the instances.

**Step-10:** Now after performing the above steps, paste the “docker swarm join …” command-**(testworker1)**

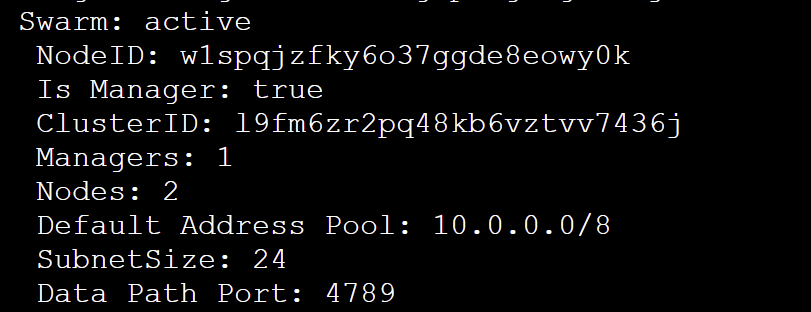
sudo service docker start



**Step-11:** Check the swarm status and the number of nodes in the testmanager.

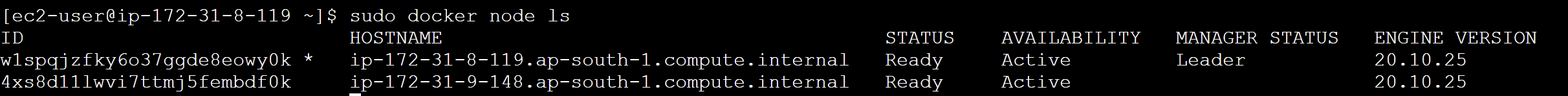
sudo docker info





**Step-11: (testmanager)**Check the available nodes.

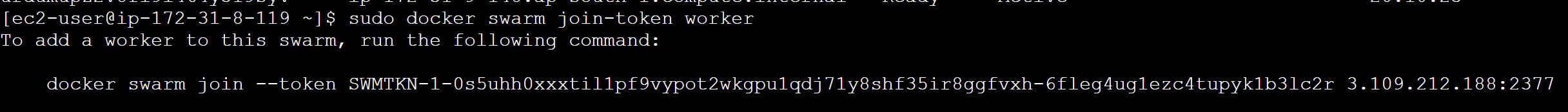
sudo docker node ls



**Docker swarm services**

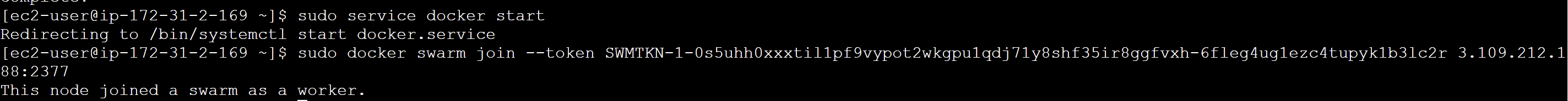
**Step-1: (testmanager)**Firstly join another instance to the swarm service**.** For getting the join token again, type the following command:

sudo docker swarm join-token worker



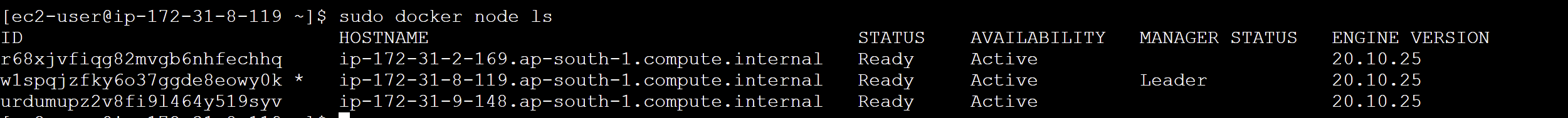
**Step-2:** Start the service and type the docker swarm join command copied from the above **.(testworker2)**

sudo service docker start



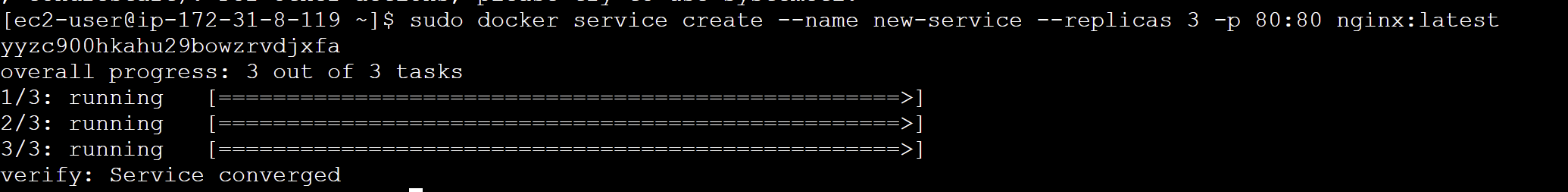
**Step-3:** **(testmanager)**Go to the manager instance and check the available nodes. Here, both the worker nodes are added to the swarm service.

sudo docker node ls



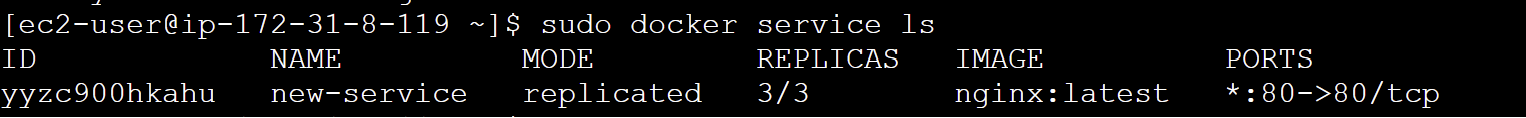
**Step-4:** **(testmanager)** Run the below command. This command creates a Docker service named "new-service" with 3 replicas running the "nginx:latest" image, mapping port 80 from the host to port 80 in the container.

sudo docker service create - -name new-service - -replicas 3 -p 80:80 nginx:latest



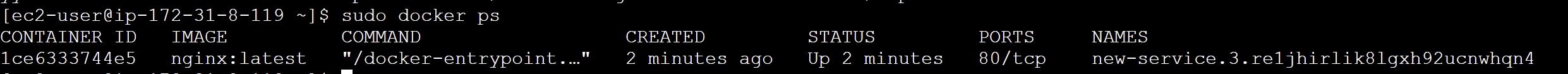
**Step-5:** **(testmanager)**Check whether the service is created or not.

sudo docker service ls



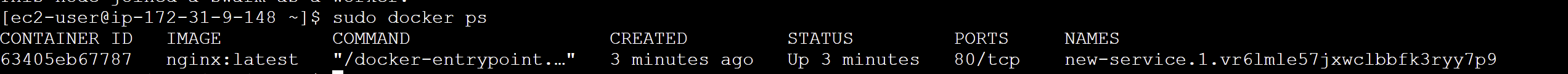
**Step-6:** **(testmanager)**Check whether the container is created or not.

sudo docker ps



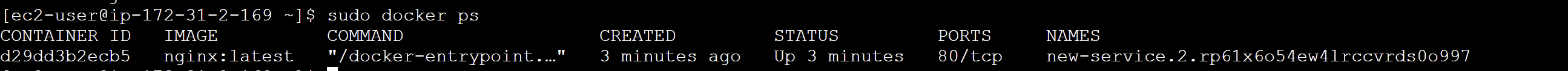
**Step-7:** **(testworker1)**Check whether the container is created or not on worker nodes as well.

sudo docker ps

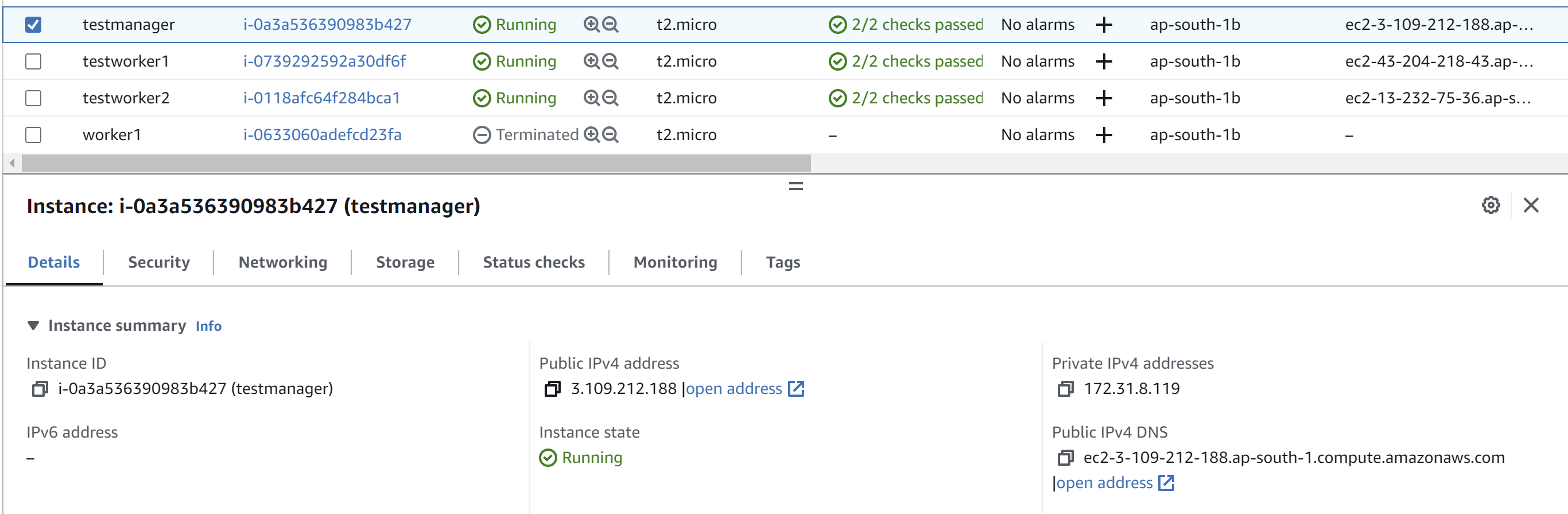


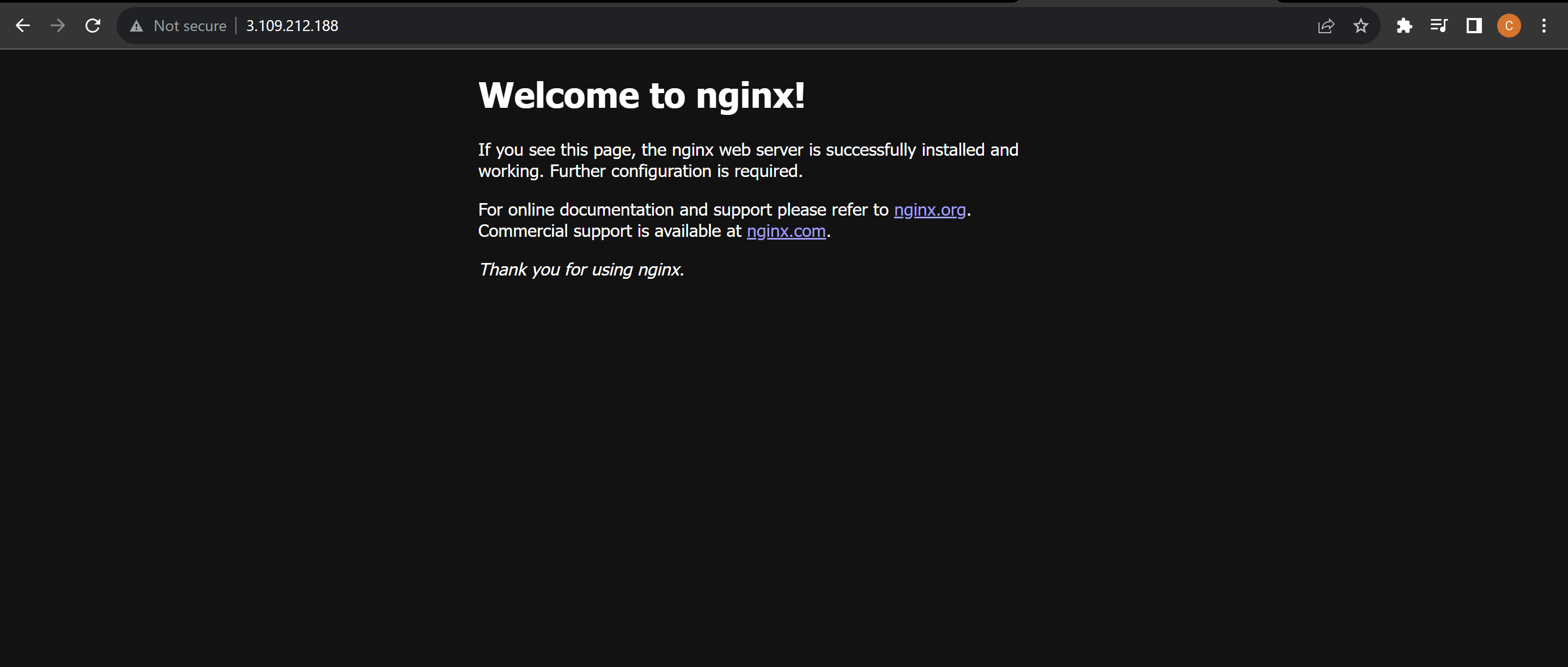
**Step-8:** **(testworker2)**Check whether the container is created or not.

sudo docker ps



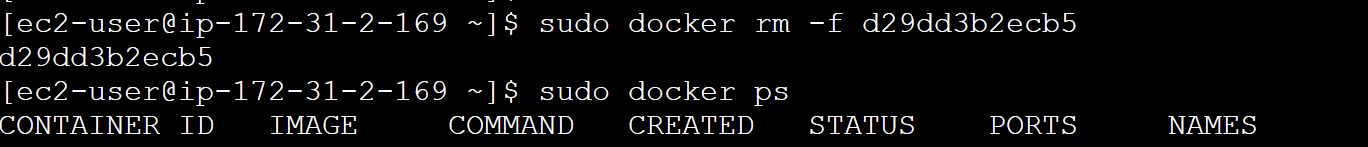
**Note:** Now you can copy the public IPs of the instances and check them on browser whether they are accessible or not.





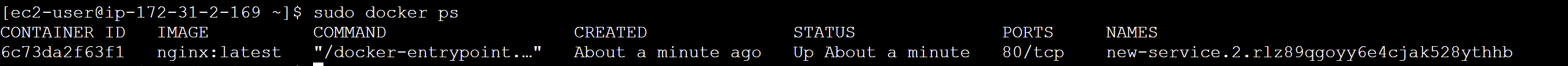
**Step-9**: **(testworker2)** Now let’s remove the container that is created on the worker2 node.

sudo docker rm -f <container-id>

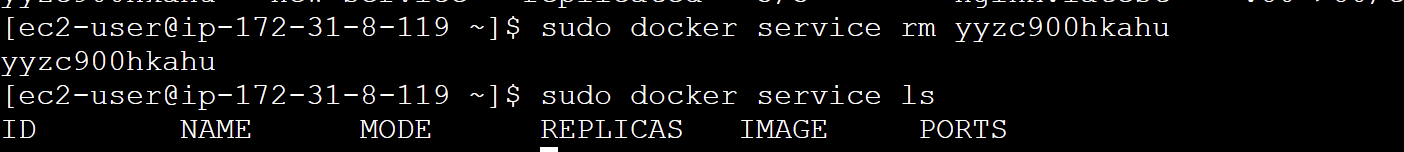


**Step-10:(testworker2)** After a few seconds, the manager detects that the container is stopped and not working on the worker node so it relaunches the container on that node.

sudo docker ps

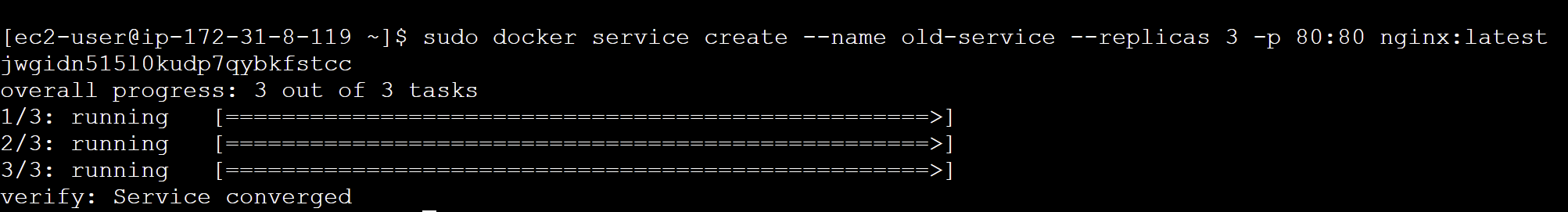


**Step-11:(testmanager)** Now if we remove the service from the manager node, the container will be removed from all the nodes as well.

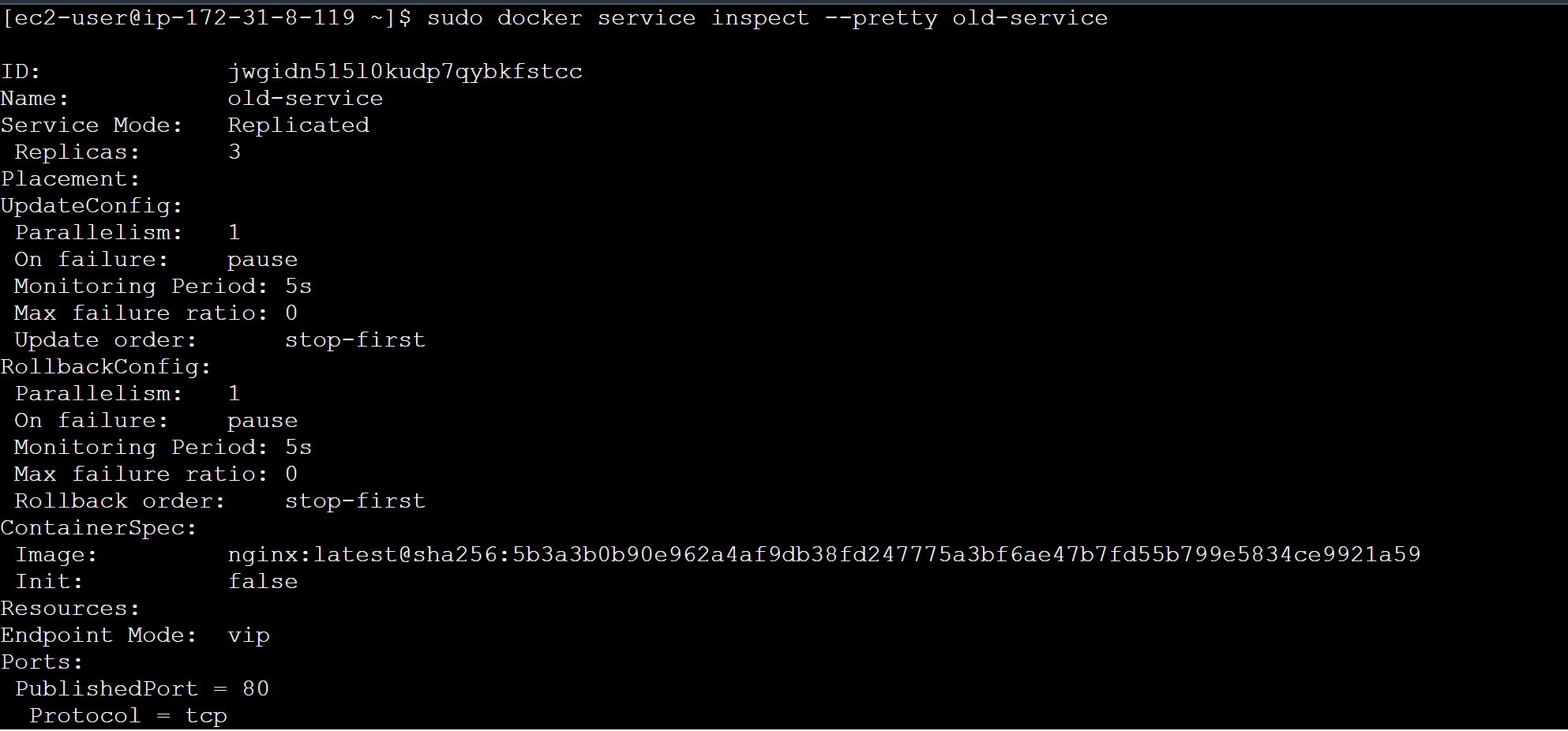


**Step-12:(testmanager)** Recreate the service with the same configuration because we have removed the previous one and inspect it.

sudo docker service create - -name old-service - - replicas 3 -p 80:80 nginx:latest

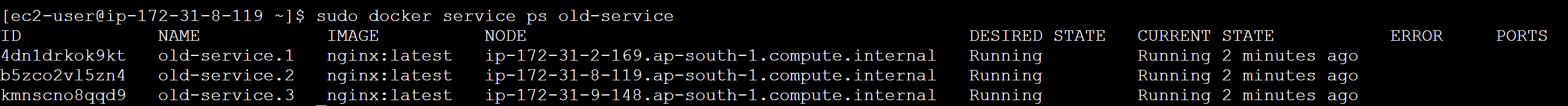


sudo docker service inspect - -pretty old-service



**Step-12:(testmanager)** Instead of going to each node and checking the service, we can do this:

sudo docker service ps old-service



**Deploy**

**Step-1:** In order to deploy services, make a yaml file listing all the services inside it. Open any text editor and enter the following code that includes two services: nginx and ubuntu.

sudo vim practice.yml



version: '3.3'

services:

sample1:

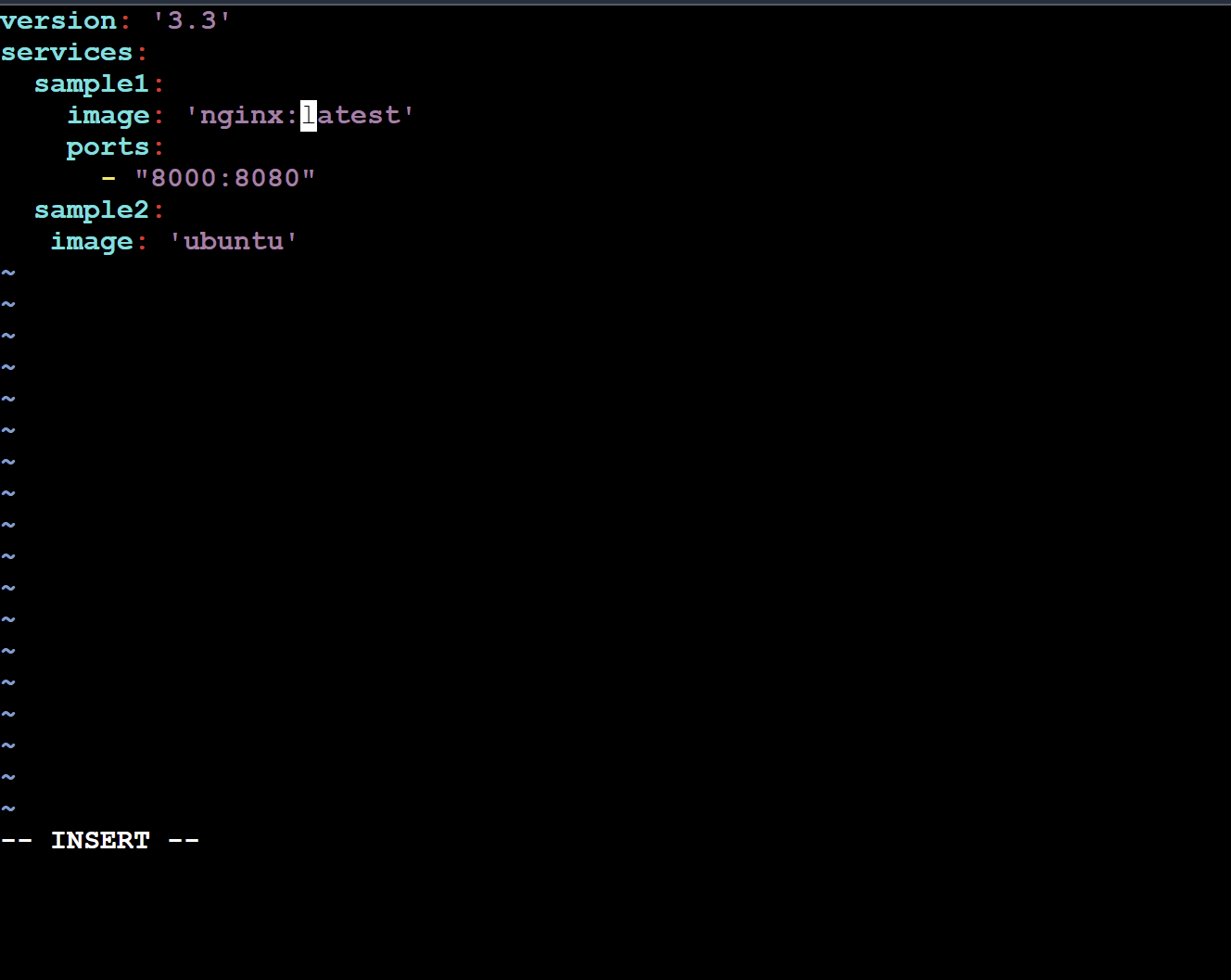
image: 'nginx:latest'

ports:

- "8000:8080"

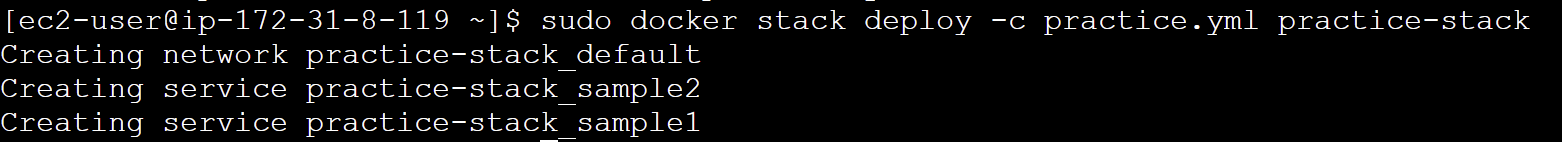
sample2:

 image: 'ubuntu'

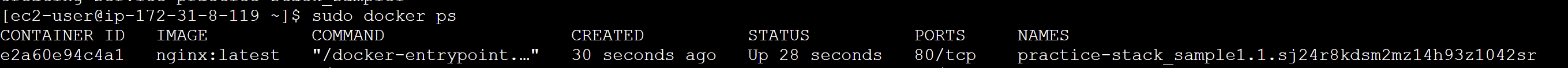


**Step-2: (testmanager)** Run the following command. This command deploys a Docker stack named "practice-stack" using the configuration defined in the "practice.yml" file.

sudo docker stack deploy -c practice.yml practice-stack



sudo docker ps



**Note:** In the yaml file, we didn’t mention the number of replicas and thus the default value for replica is 1 and hence the container are not created on the worker nodes.

**testworker1:** sudo docker ps

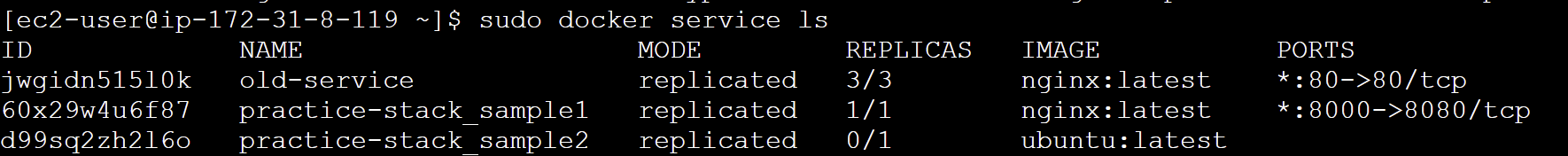


**testworker2:** sudo docker ps



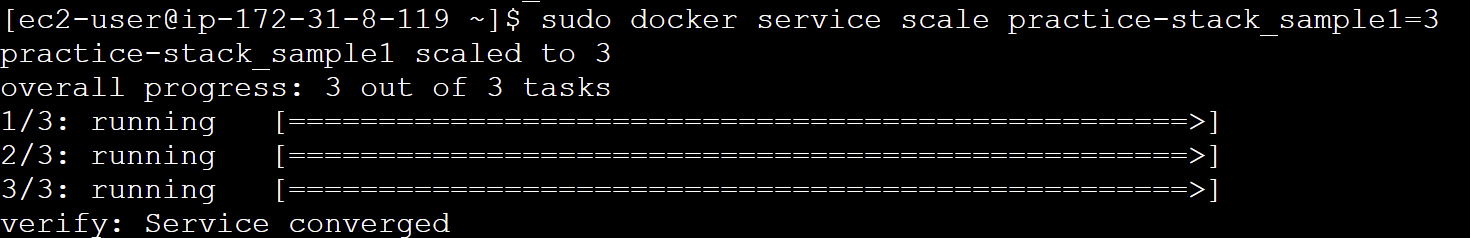
**Step-3: (testmanager)** Here two services are created: one for ubuntu and one for nginx.

sudo docker service ls

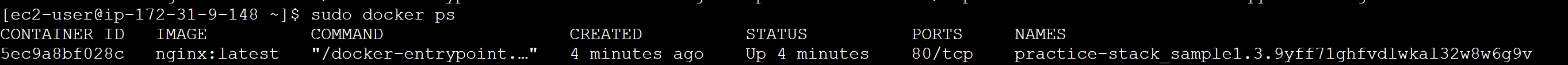


**Step-4: (testmanager)** Using the scale command, we can increase the number of replicas for a particular service.

sudo docker service scale practice-stack\_sample1=3



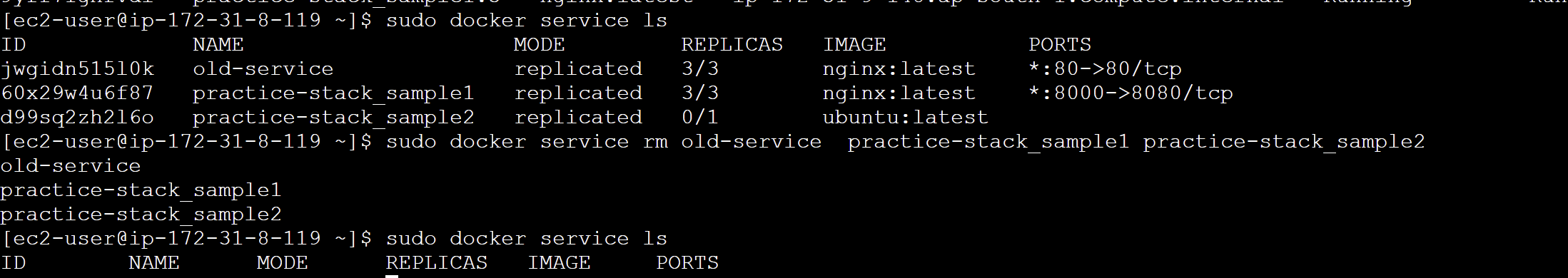
sudo docker ps



**Rolling updates**

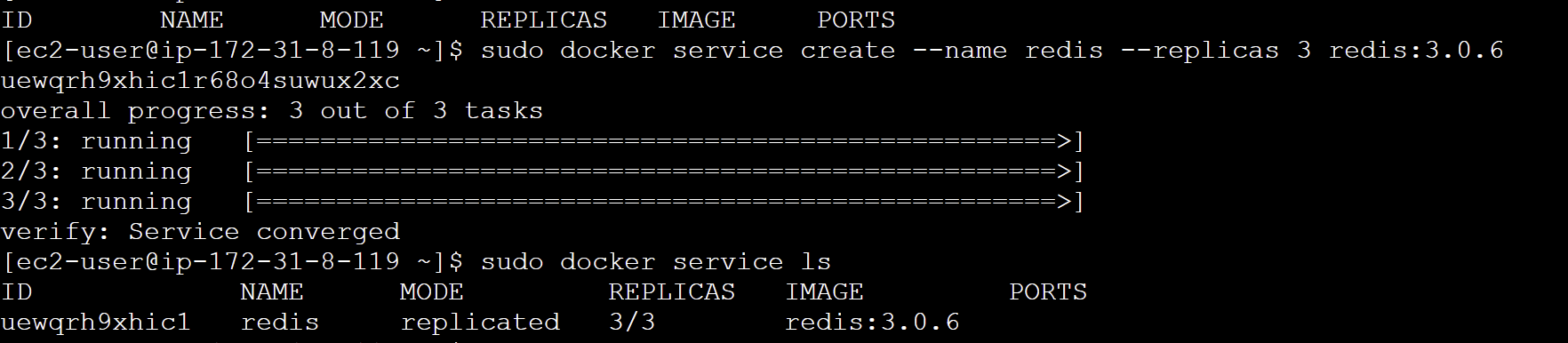
**Step-1: (testmanager)** Firstly, remove the existing services using the following command:

sudo docker service ls



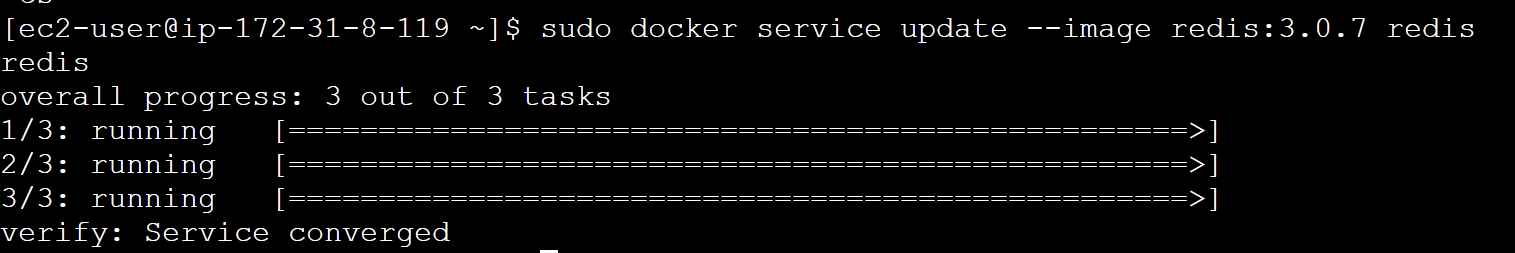
**Step-2: (testmanager)** Create a Docker Swarm service named "redis" with three replicas, using the "redis:3.0.6" container image.

sudo docker service create - -name redis - - replicas 3 redis:3.0.6

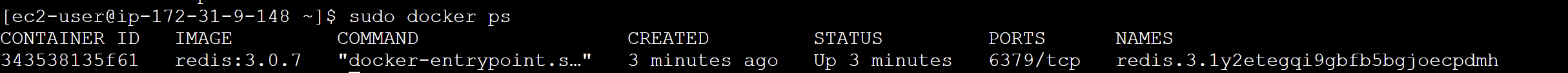


**Step-3: (testmanager)** Using the update command we can update the version of the redis service from 3.0.6 to 3.0.7

sudo docker service update - -image redis:3.0.7 redis



sudo docker ps

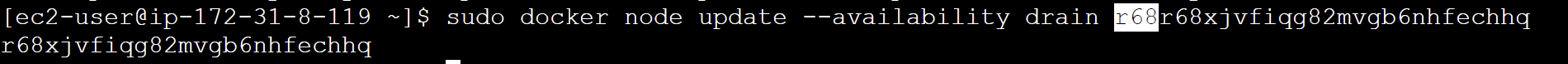


**Note:** It will automatically get updated on the nodes without stopping the service.

**Draining nodes**

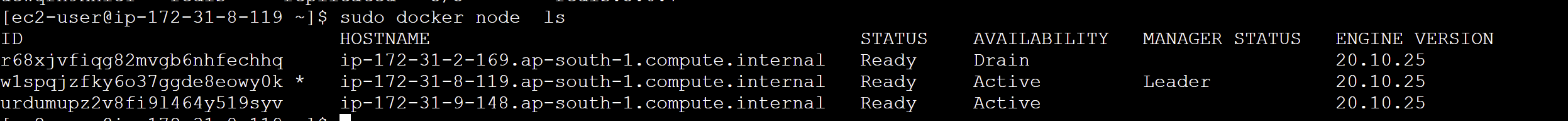
**Step-1: (testmanager)** The below command sets a worker node's availability to "drain" in a Docker Swarm cluster, meaning it stops and reschedules containers on that node to prepare it for maintenance or removal.

sudo docker node update - - availability drain <workernode-id>



**Step-2: (testworker2)** Check the status of the node. It shows “Drain”

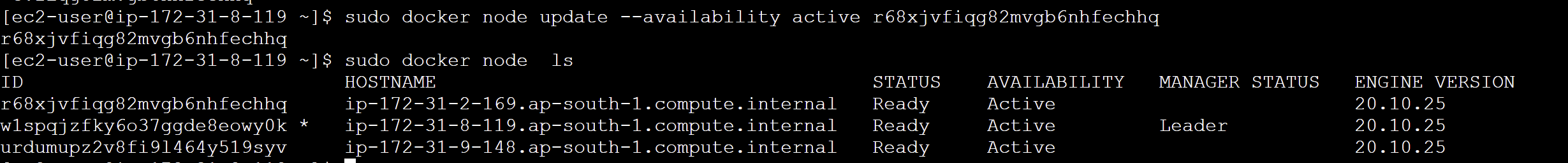
sudo docker node ls



**Note:** At this time no container will be running on the testworker1 because we have drained it.

**Step-3: (testmanager)** In the same way, we can set the status to “active” again.

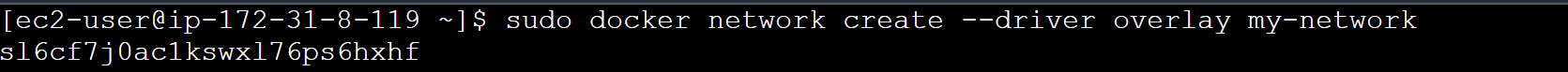
sudo docker node update - - availability active <workernode-id>



**Connecting to a network**

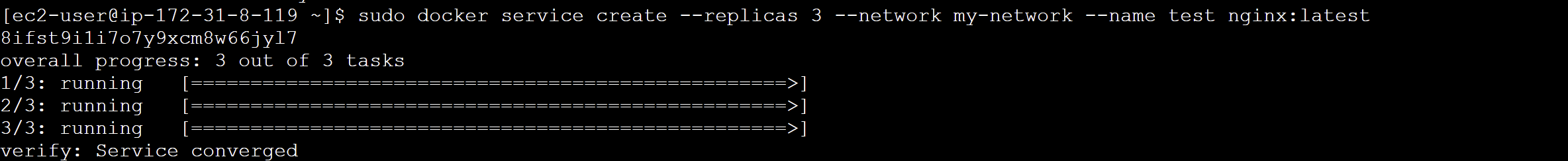
**Step-1:** Create an overlay network.

sudo docker network create - -driver overlay my-network

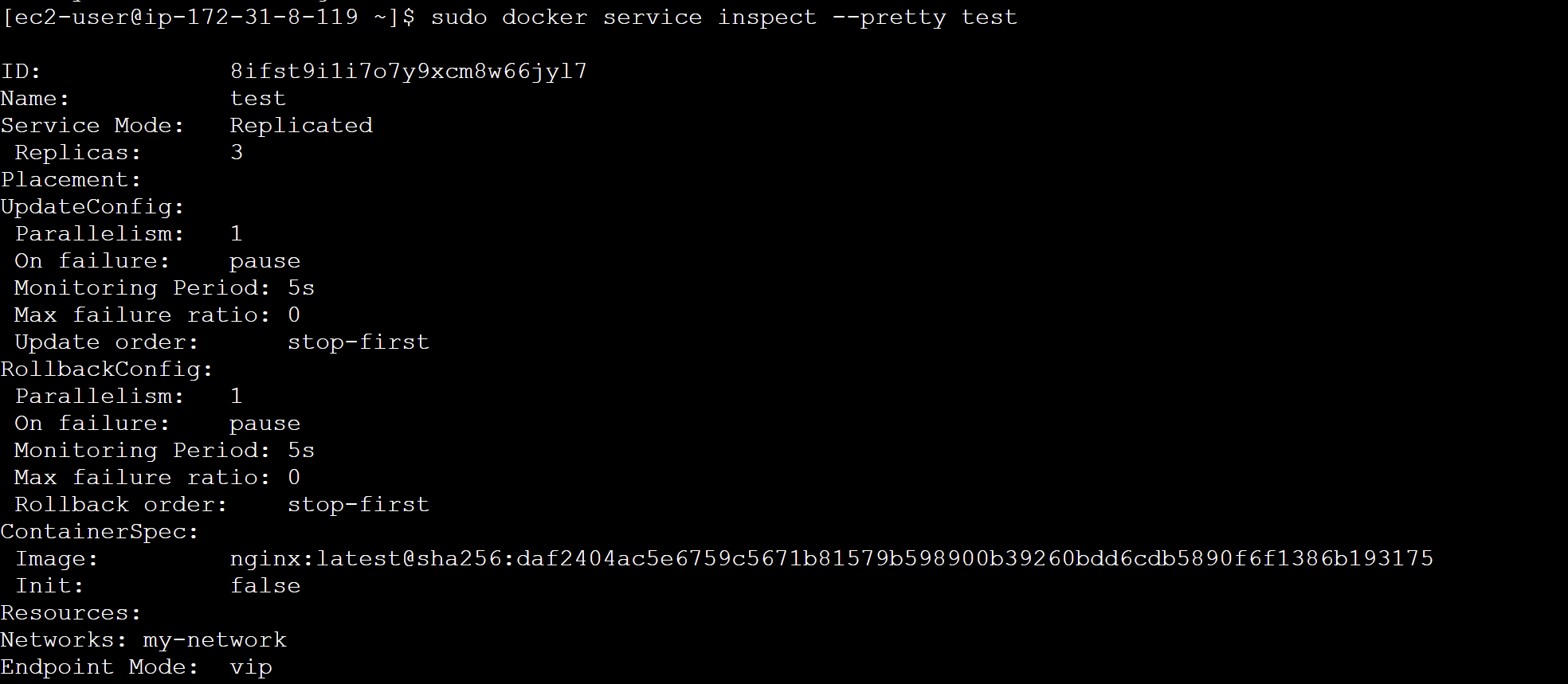


**Step-2:** While creating the service, we can specify the user-defined network using the “- - network” flag.

sudo docker service create - - replicas 3 - -network my-network - -name test nginx:latest

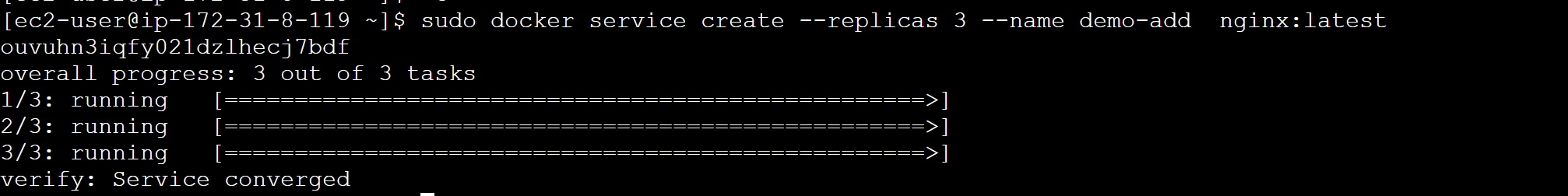


sudo docker service inspect - -pretty test

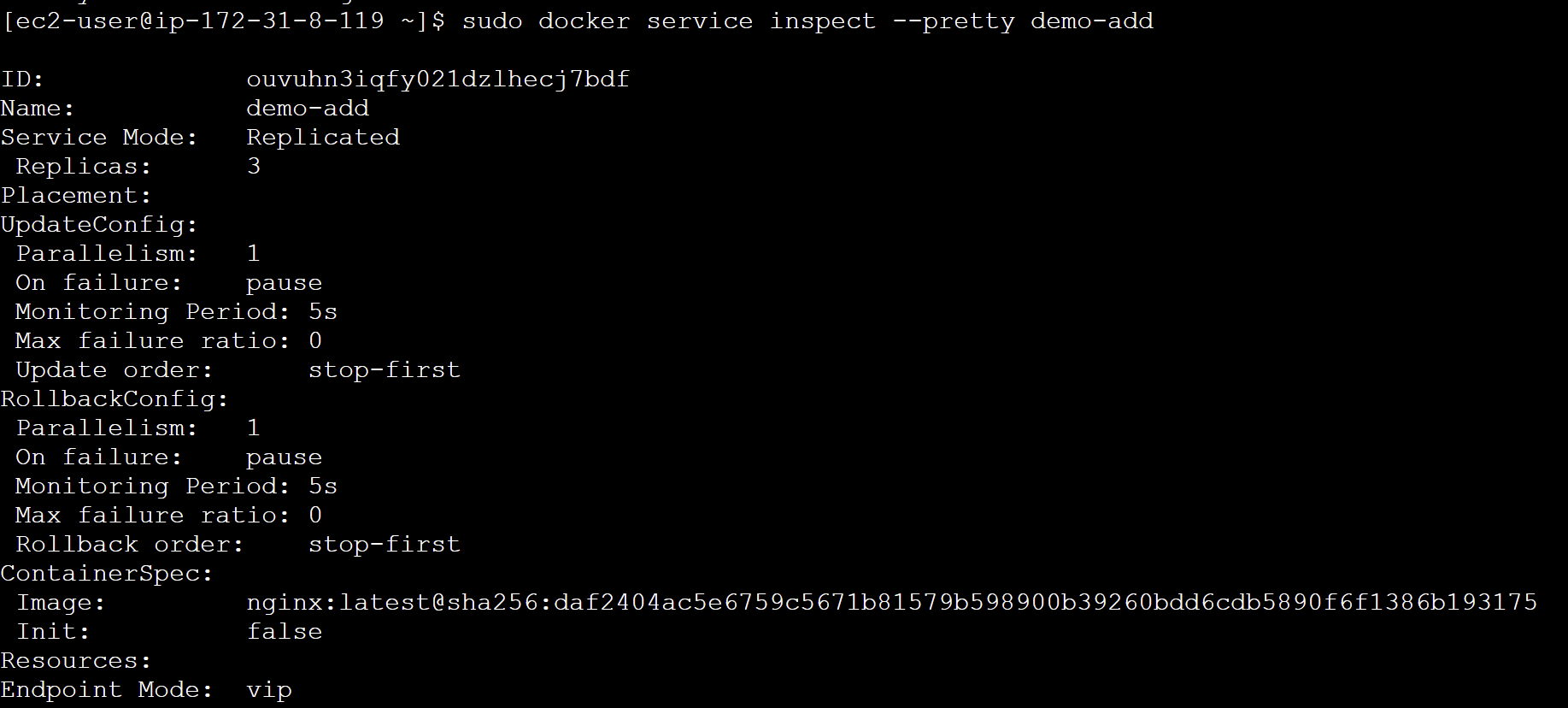


**Step-3:** While creating the below service, no network is attached to it by default.

sudo docker service create - - replicas 3 - -name demo-add nginx:latest

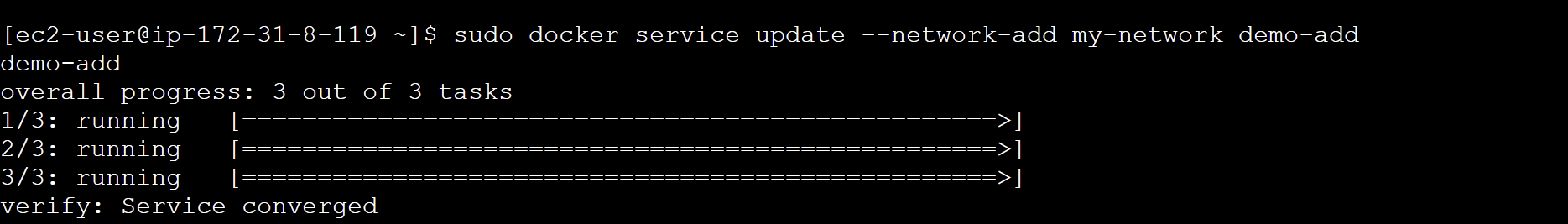


sudo docker service inspect - -pretty demo-add

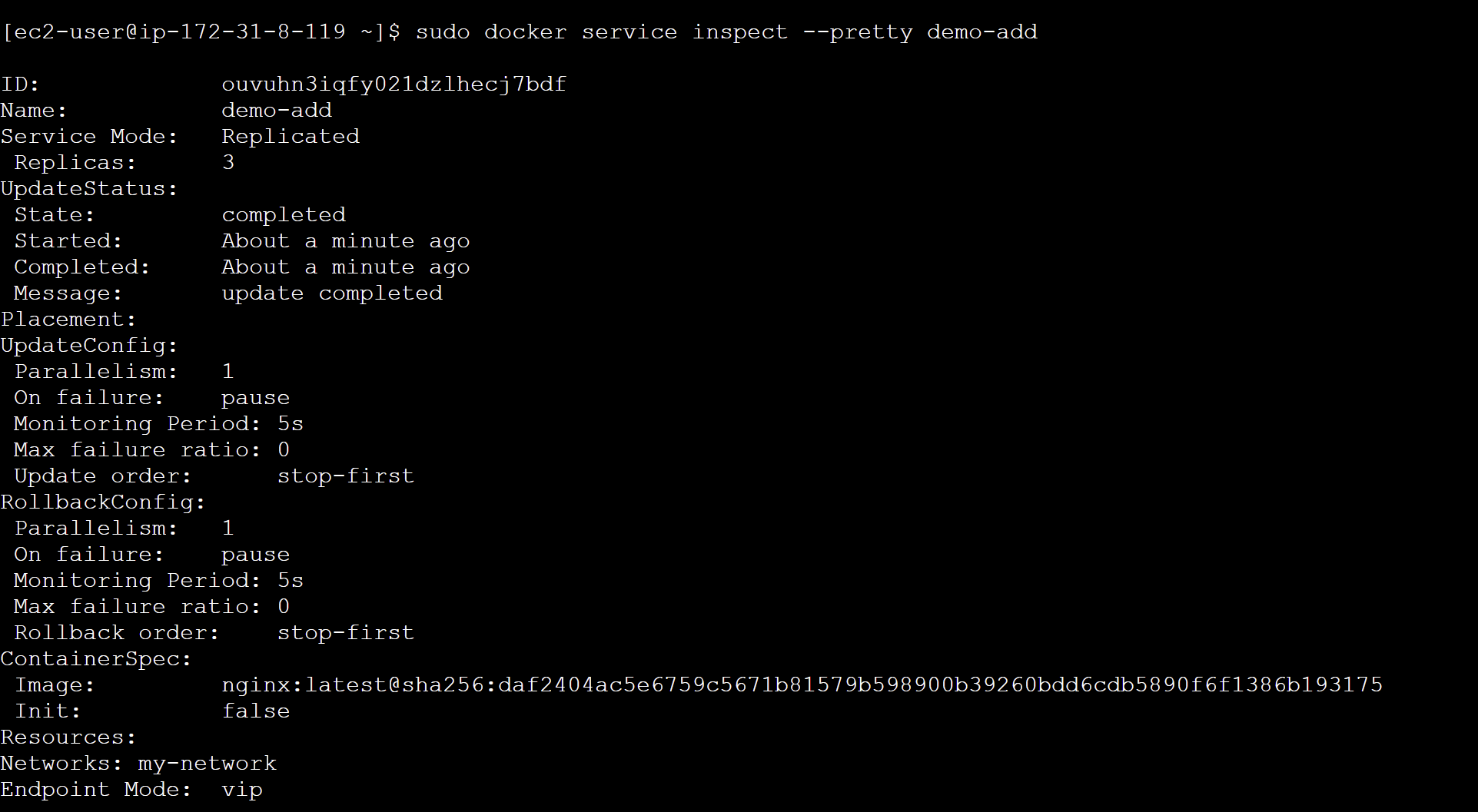


**Step-4:** We can attach a network to any existing service by using the “update” command

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

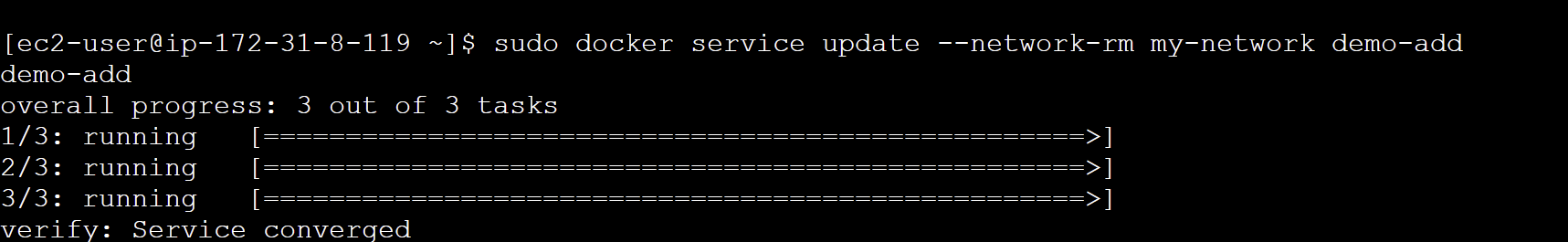


sudo docker service inspect - -pretty demo-add

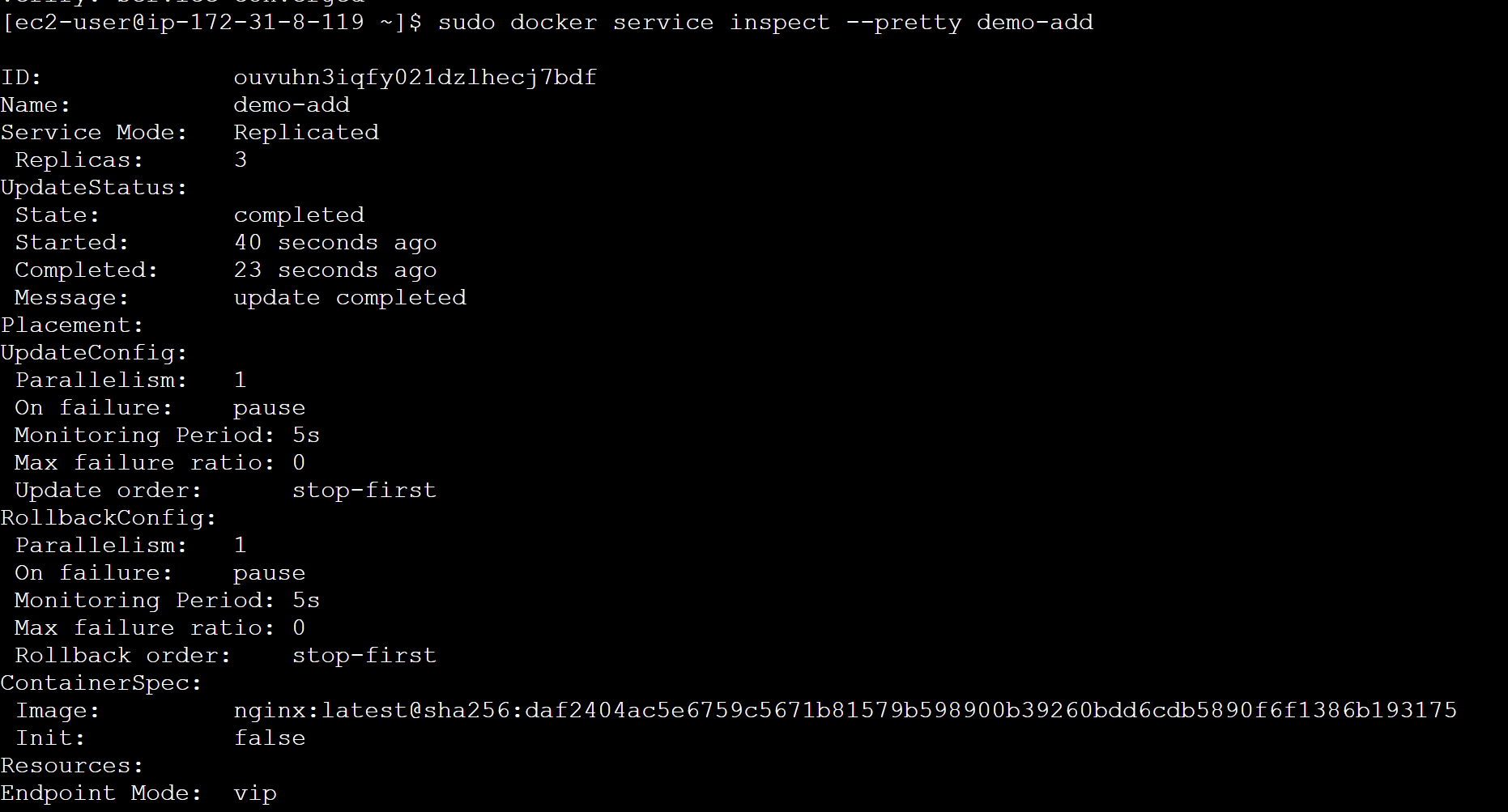


**Step-5:** We can also remove the network in the same way.

sudo docker service update - -network-rm my-network demo-add

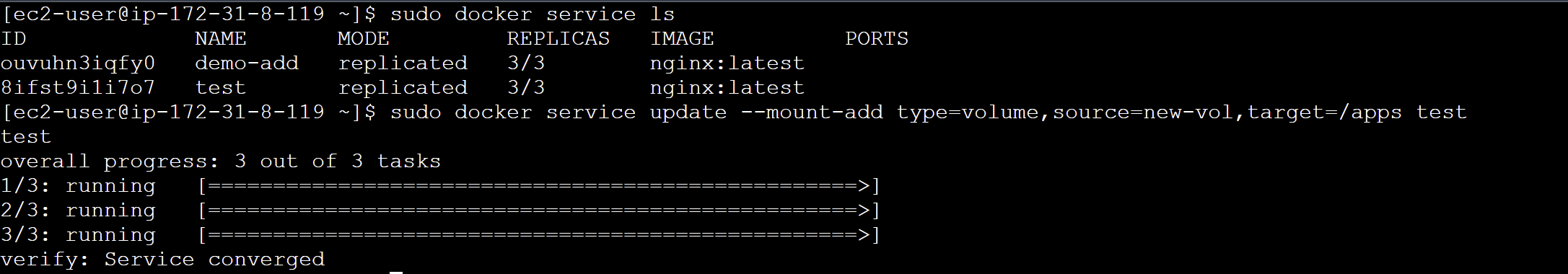


sudo docker service inspect - -pretty demo-add

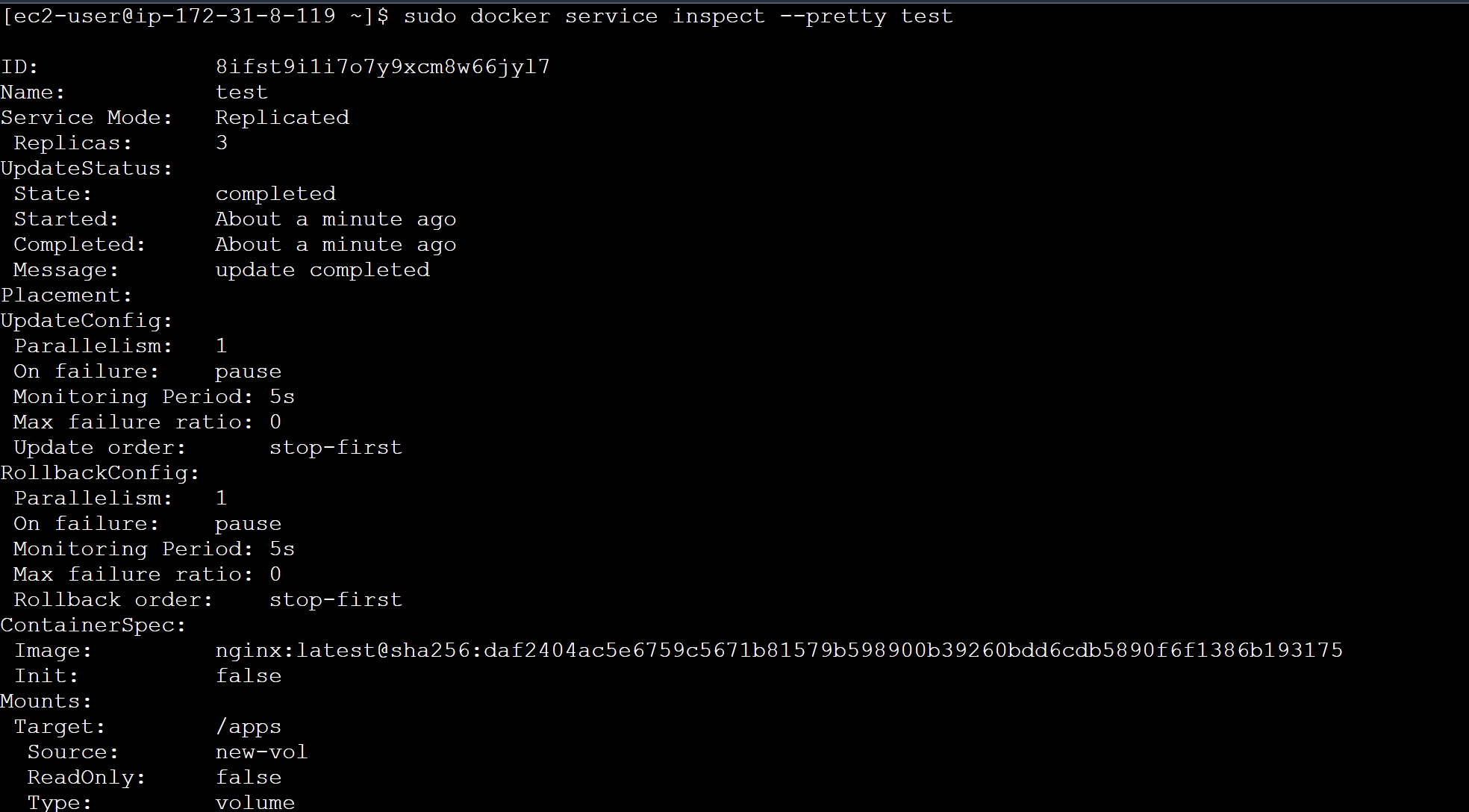


**Volume**

sudo docker service update --mount-add type=volume,source=new-vol,target=/apps test



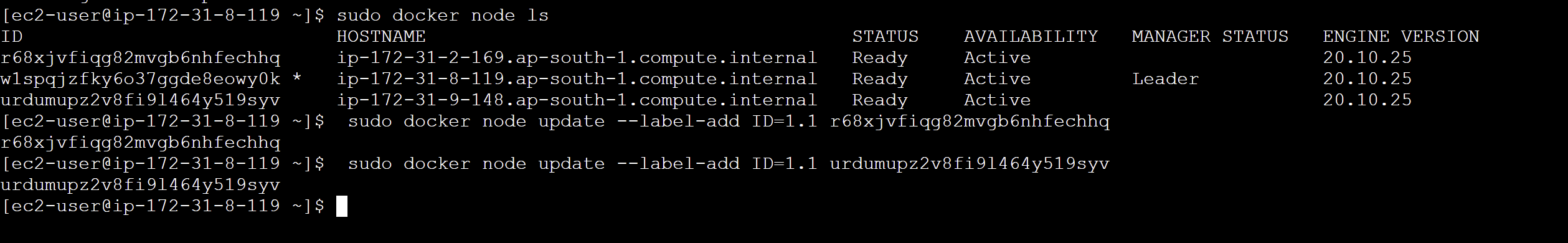
sudo docker service inspect --pretty test



**Constraints**

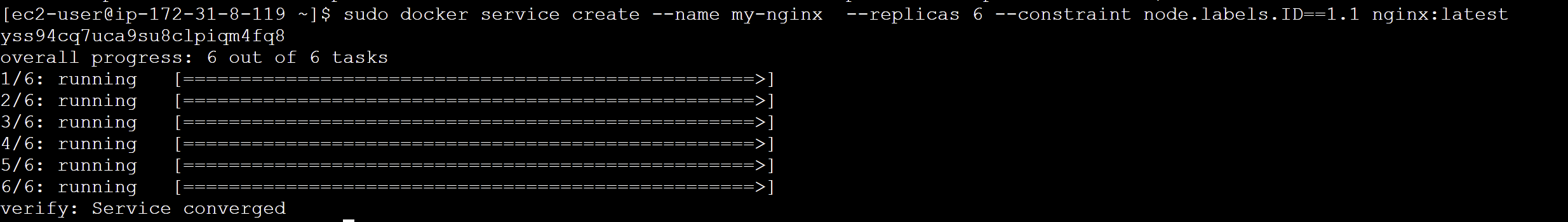
**Step-1:** By using the update command, we can add constraints to the nodes. This command adds a label with the key "ID" and the value "1.1" to a Docker Swarm node specified by "<node\_id>."

sudo docker node update --label-add ID=1.1 <node\_id>



**Step-2:** This command creates a Docker service named "my-nginx" with 6 replicas, and it restricts these replicas to nodes labeled with "ID" equal to "1.1," using the "nginx:latest" image.

sudo docker service create --name my-nginx --replicas 6 --constraint node.labels.ID==1.1 nginx:latest



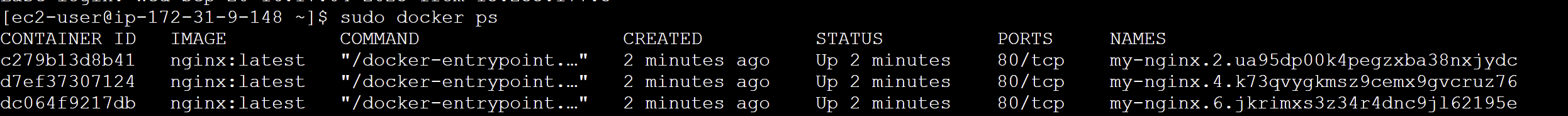
**Note:** No containers running in the manager node because its id is not 1.1



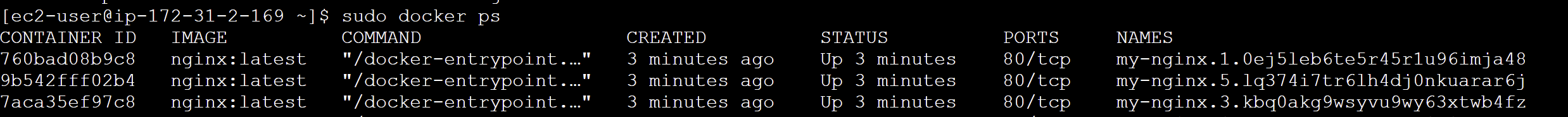
If we go to worker1 node and run “docker ps” we can see that the 6 replicas defined in the service are divided between the 2 nodes with the specified ID:

sudo docker ps

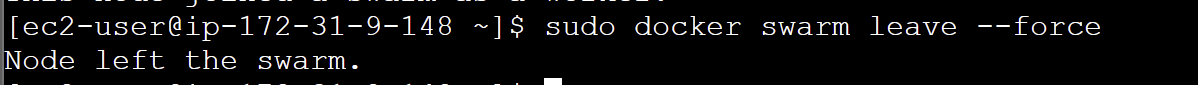
**(testworker1)**



**(testworker2)**



**Step-3:** (testworker) Using the swarm leave command, nodes can be forced to leave the swarm.

sudo docker swarm leave --force

**Step-4:** (testmanager) To remove a node from the swarm, the below command is used:

sudo docker node rm <node-id>

