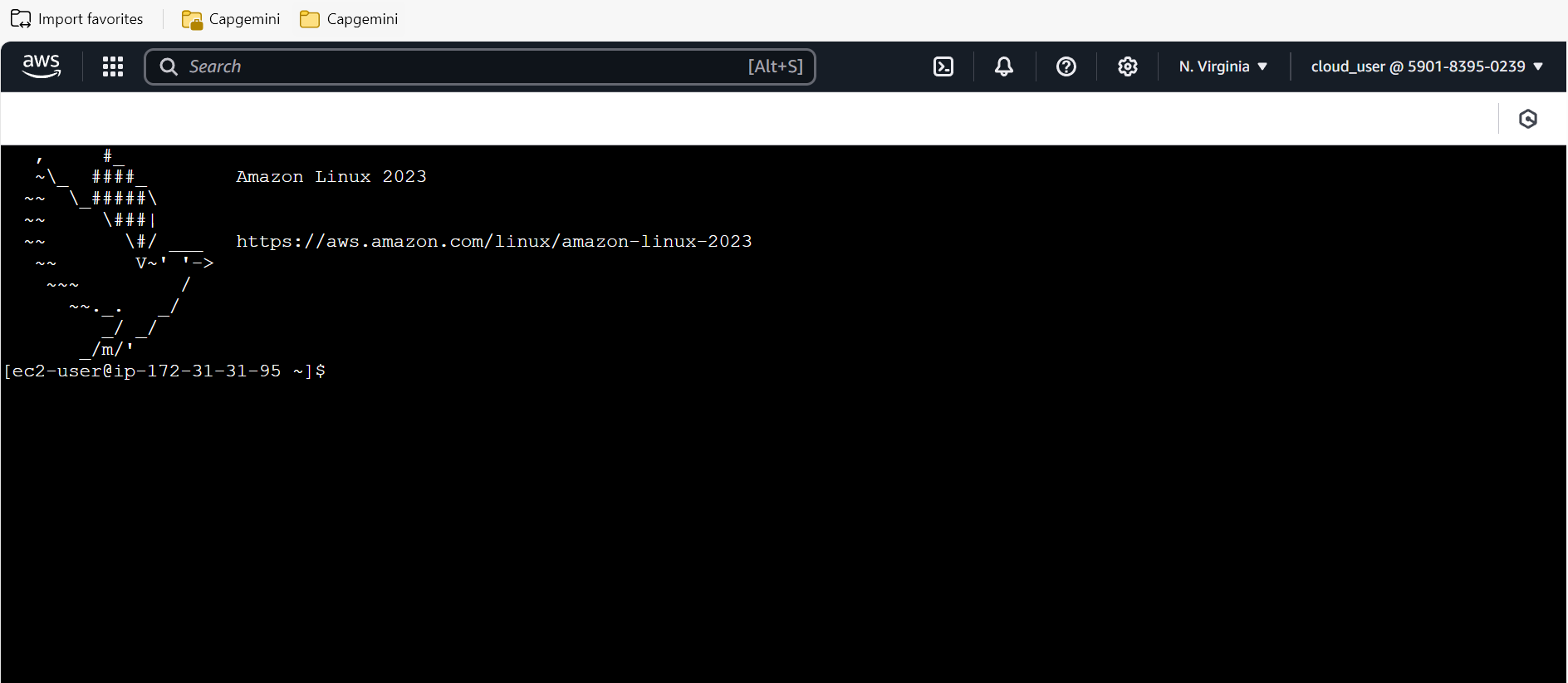
**Importance of Devops using – Install docker on AWS EC2:**

**Step – 1**: Launch and create a EC2 instance -> name it as **docker-server** -> key pair (login) proceed without key pair need to select -> Select create security groups modify the name -> Click on edit -> name of the security group is **docker-sg** -> click on launch instance -> The image we can use 8080 -> Click on the instance id -> Change the outbound rule to inbound rule -> click on the add rule add 8080 select anywhere -> save rules -> this rules is enable now go back to the instances -> select the instance and connect.

Once it connected, we can see this type of CMD page.



**Step – 2**: Docker commands with Aws EC2

1. **sudo yum install docker** – This is the command is used to run and install the docker.
2. **sudo systemctl  start docker** – This command is used to start the docker
3. **sudo docker –version** – To check the docker version.
4. **sudo docker run hello-world** – It says that docker is running successful.
5. **sudo docker pull nginx** – It is downloading the image of nginx from the docker hub
6. **sudo docker images** – It shows the images.
7. **sudo docker container ls** – I don’t have any container when we use this command Note: Running instance is one container.
8. **sudo docker rm -f image name** – Using this command we can delete our containers.
9. **sudo docker run -d --p 80:80 nginx** – Using this command create a container with port number
10. **sudo docker run -d -p 40:80 --name=mycon4 nginx** – Using this command name the container name.

**Install docker on windows and Aws difference:**

* We have gone for the **.exe** file for windows, for installing the docker desktop.

Ex: We are using the commands are: docker run, docker push etc.

* But in Aws EC2 we are creating an instance connected it and in the CLI using **sudo yum install docker** this command.

Ex: we are using the commands are above.

**Difference between docker image and container:**

**Image:**

* A Docker image is a lightweight, immutable, and read-only template that contains all the instructions and dependencies needed to run an application. It includes the application code, runtime, libraries, environment variables, and configuration files.
* It serves as a blueprint for creating Docker containers.
* A single image can be used to create multiple containers.

Ex: docker build -t my-app-image .

docker pull nginx

**Docker Container:**

* A Docker container is a running instance of a Docker image. It includes a writable layer on top of the image, allowing it to have a dynamic state and perform operations.
* It runs the actual application, encapsulating it along with its environment.
* Multiple containers can be created from the same image, each with its own unique state.

Ex: docker run -d -p 8080:80 nginx

docker run --name my-app-container my-app-image

**Image**: The static blueprint for an application.

**Container**: The dynamic runtime instance of that blueprint.

This is for default message coming from nginx

A screenshot of a computer error

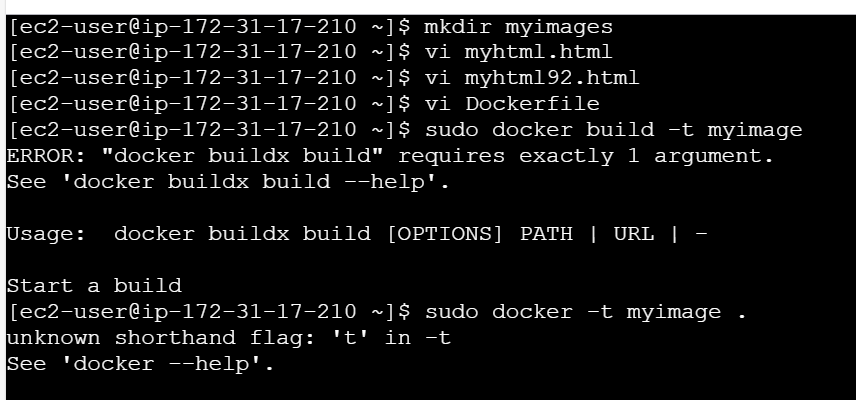
Description automatically generated

**For creating our own image:**

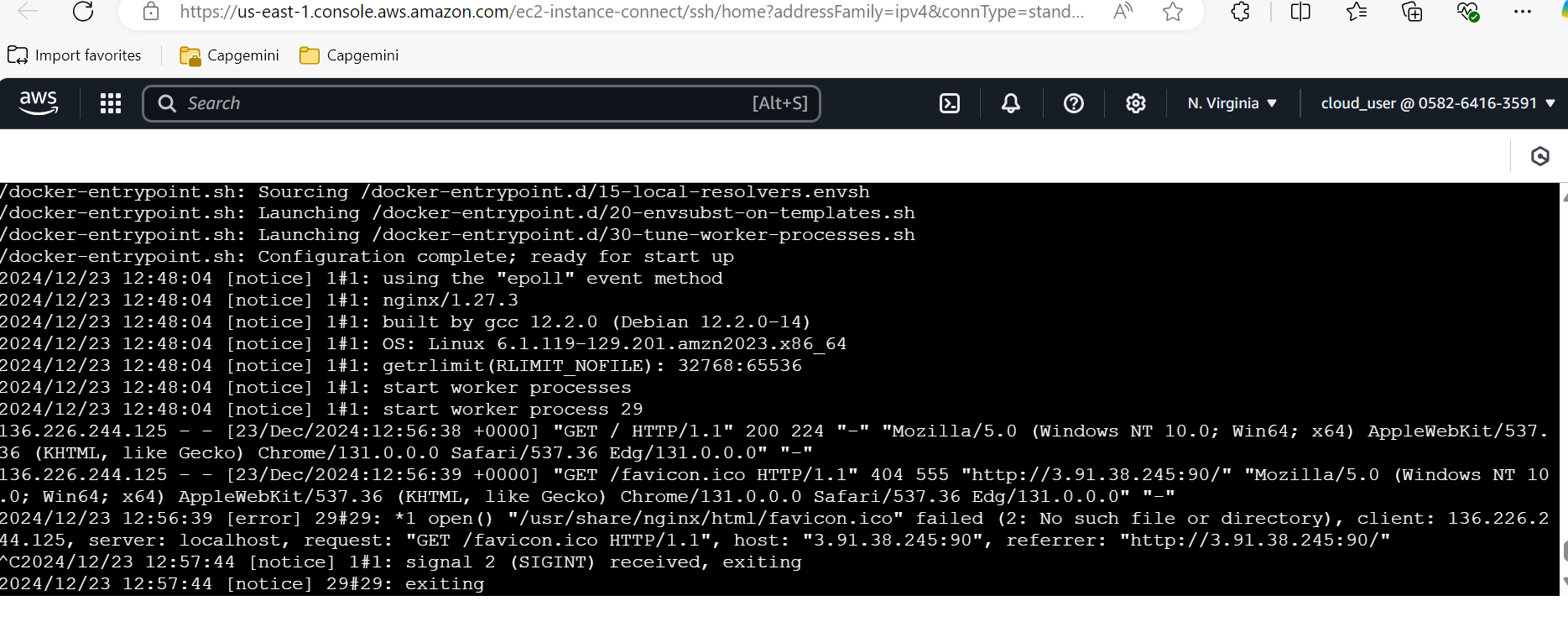
* **mkdir myimages** – Creating a directory, **cd myimages** gone to this directory.
* **vi myhtml.html** – command for creating a html file
* The below images are the commands to create a new image.
* For adding a content in the file use **I** keyword for inserting use **esc** once add the content to the file and finally **:wq** again back to the cmd.

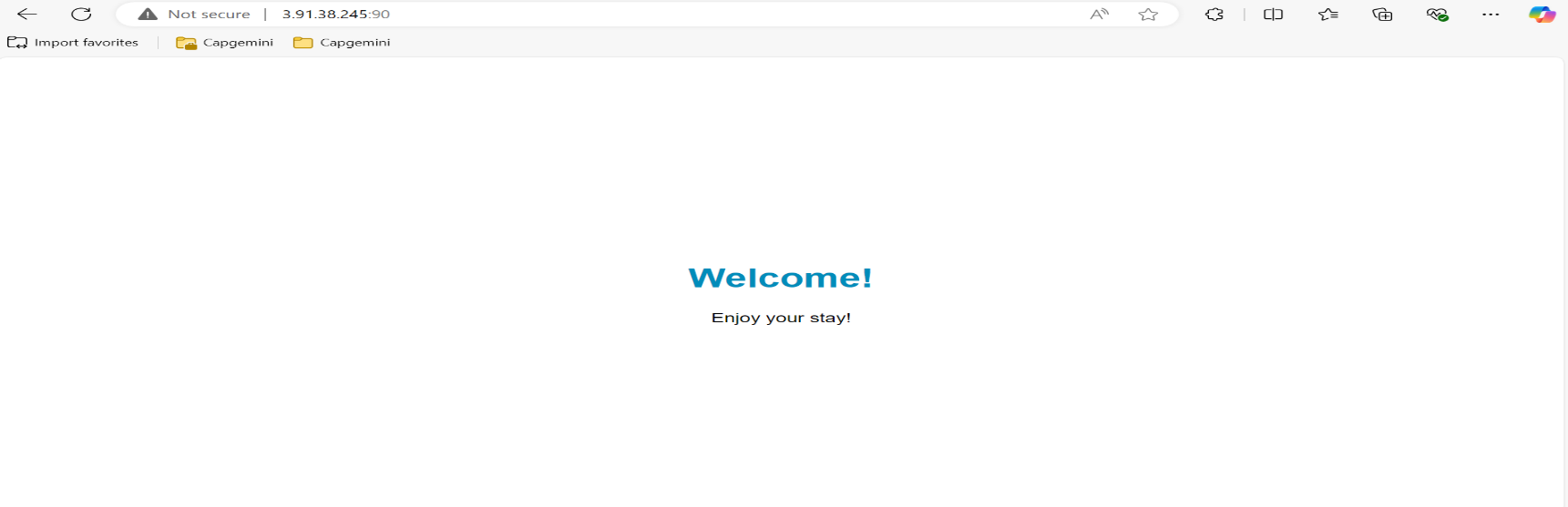
This we must add docker file.

* **FROM nginx**
* **COPY myhtml2.html /usr/share/nginx/html/index.html**
* **EXPOSE 90**

****

* We have used the proper port number in the security groups which port we are going to use

****

****

* **Now we have push image to the docker hub.**

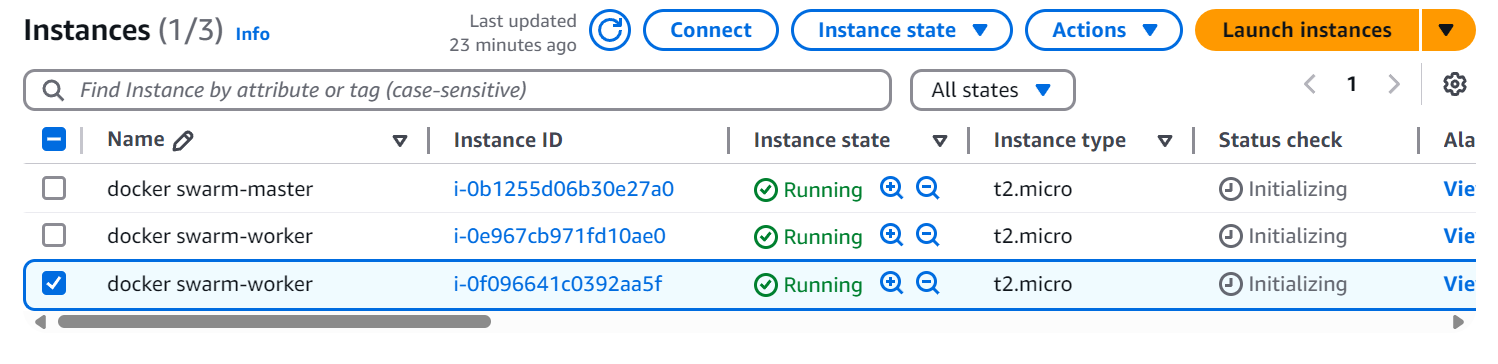
1. sudo docker tag imagename usernameofdockerrepository/imagename:latest
2. Login to the repository – sudo docker login
3. Enter my email address place of username and password enter my old password.
4. Created a repository name as **dockerconfig.**
5. sudo docker push varunkumarkm43/ myimage:latest

**Docker Swarm: Overview and Features:**

**Docker Swarm** is Docker’s native container orchestration and clustering tool. It allows you to manage a group of Docker hosts (nodes) as a single virtual system. With Docker Swarm, you can deploy, scale, and manage containers across multiple nodes seamlessly.

Basically, docker swarm we are creating a cluster of machines and then each of the machines we need to install docker then make 1 machine as a master node and remaining are worker mode.

**Step – 1** – create 3 EC2 instance -> name as **docker swarm** in summary create 3 instance -> edit security group select create security group name it has **swarm-sg** -> Proceed without key pair -> Click on launch instance.



Instance got created -> modified name -> first instance is **docker swarm-master** -> second and third instance are **docker swarm-worker**.

**Step - 2**

[Open protocols and ports between the hosts](https://docs.docker.com/engine/swarm/swarm-tutorial/#open-protocols-and-ports-between-the-hosts)

The following ports must be available. On some systems, these ports are open by default.

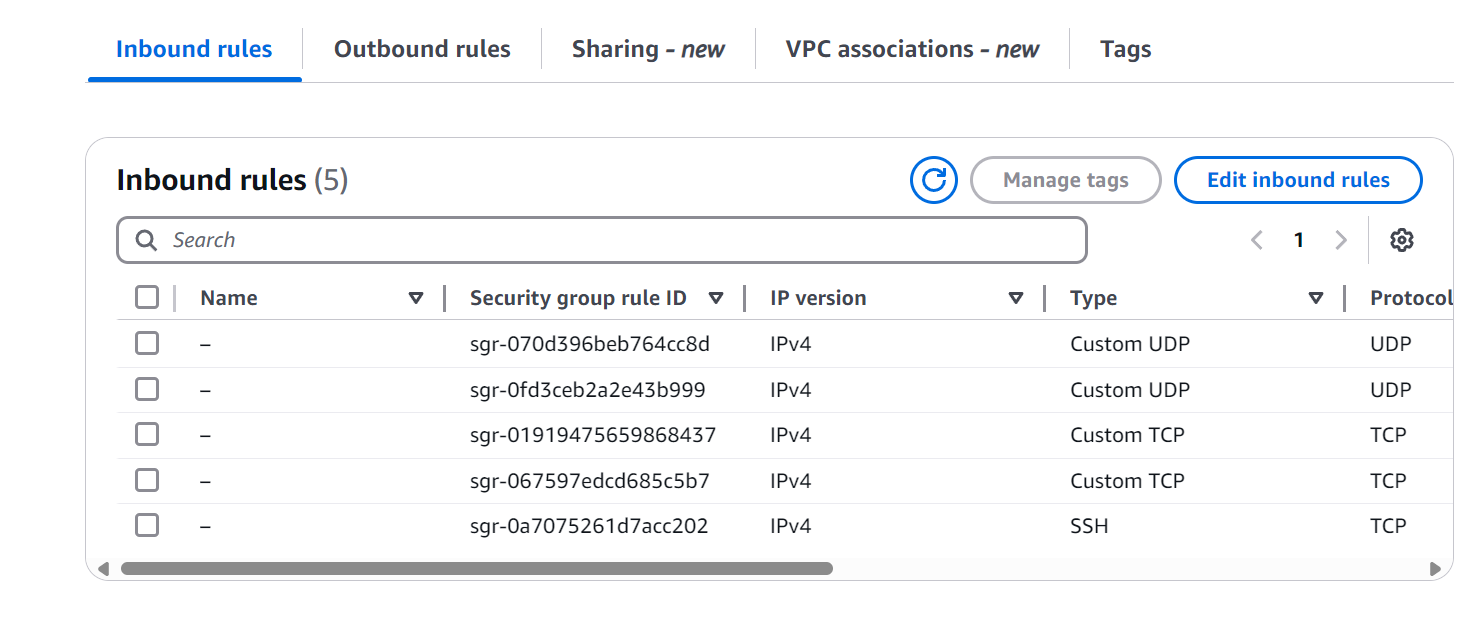
* Port 2377 TCP for communication with and between manager nodes
* Port 7946 TCP/UDP for overlay network node discovery
* Port 4789 UDP (configurable) for overlay network traffic

We need to open this 3 ports TCP and UDP ports, for that we need to go for security group in Aws -> click on **Security group ID** swarm-sg id -> click on edit inbound rules:

Add rule -> select custom TCP -> add 2377 port number and select Anywhere-Ipv4  
add one more rule -> select custom TCP -> add 7946 port number and select Anywhere-Ipv4  
add one more rule -> select custom UDP -> add 7946 port number and select Anywhere-Ipv4  
add one more rule -> select custom UDP -> add 4789 port number and select Anywhere-Ipv4

add one more rule -> select custom TCP -> add 80 port number and select Anywhere-Ipv4  
add one more rule -> select custom TCP -> add 8080 port number and select Anywhere-Ipv4

Add this security groups and click on save rules.



**Step - 3**

Open this one-by-one node and install the docker

1. In instance -> select master instance -> click on connect. -> Use the below commands to install and start the docker.

* sudo yum install docker
* sudo systemctl start docker

1. In instance -> select one worker instance -> click on connect. -> Use the below commands to install and start the docker

* sudo yum install docker
* sudo systemctl start docker

1. In instance -> select another worker instance -> click on connect. -> Use the below commands to install and start the docker

* sudo yum install docker
* sudo systemctl start docker

**Step – 4:** In the master node we need to initialize the docker swarm. Use **sudo docker swarm init** command to initialize the docker swarm.

A black screen with white text

Description automatically generated

docker swarm join --token SWMTKN-1-2mm7aocdayzi893uw1gu37ltso4t6pmxqgprbx0j1qci9zl8j6-5t7t1x70n6twq2ekcq9uhaxkl 172.31.19.47:2377

Use this above command to every worker node.

Use the command like below for every worker node

sudo docker swarm join --token SWMTKN-1-2mm7aocdayzi893uw1gu37ltso4t6pmxqgprbx0j1qci9zl8j6-5t7t1x70n6twq2ekcq9uhaxkl 172.31.19.47:2377

This is the way create a docker swarm architecture using Aws EC2

**Step – 5:** Run the image.

Goto the master node **sudo docker service create - -name mysvc - -replicas 3 varunkumar43/myimage** Once use this command all 3 tasks are downloading

Check the node is running or not using this command **sudo docker container ls** so master node is running here the check worker node -> one container is working here -> and worker node another container is working.

This way we can create any number of instance **Ex**: we want to create 20 instances use this below command 17 new instane get created.

**sudo docker service scale mysvc=20**

**sudo docker service create - -name mysvc2 - -replicas 10 - -public 80:80 varunkumar43/myimage** using this command to run our custom image and check this once execution is completed. Using URL **Ex**: 54.225.10.251:8080

Now we can access this application from the worker node. We can also access from the master node using 54.225.10.251:80

This is how we are going to create docker swarm we can deploy application on a docker swarm and as per our requirement we can scale up and scale down our application.

We are accessing our application using based on the IP number

**Sept – 6**: We can configure application load balancer in a EC2 instance -> Goto the instance page -> select load balancers and click -> Create a load balancer of type ALB name it as docker-alb -> Select VPS means all the availability zones -> security groups select swarm-sg -> we need to create a target group basically it is a group of EC2 instances name it has **mytgt**. -> click on next -> select all master and worker nodes -> click include pending below -> click create target group. -> this process with take 5 mins to create.

Once this target group is ready -> once all showing become healthy node, we will get some provisioning here then we can use this DNS name to access the services instead of individual Ip address.

Suppose in this example once node not working also, we can access our application with another worker node this this load balancing happening. If the master node also terminates, we can access worker node.

Once master node shot down, we cannot scaleup and scale down our application. Once all the nodes are shot down, we cannot access any node we will get the 502 bad Gateway error response it means we doesn’t have any healthy nodes.