Docker 4:

What’s alternative for docker-compose? it is kubernetes

[ec2-user@ip-172-31-19-227 ~]$ git clone https://github.com/Haider7214/springboot-mysql-docker-compose.git

[ec2-user@ip-172-31-19-227 ~]$ ls -l

total 0

drwxr-xr-x. 3 ec2-user ec2-user 32 May 7 00:13 demo-webapp

drwxr-xr-x. 5 ec2-user ec2-user 83 May 6 02:43 my-webapp

drwxr-xr-x. 4 ec2-user ec2-user 40 May 11 20:32 SpringBootApp

drwxr-xr-x. 4 ec2-user ec2-user 58 May 13 23:55 springboot-mysql-docker-compose

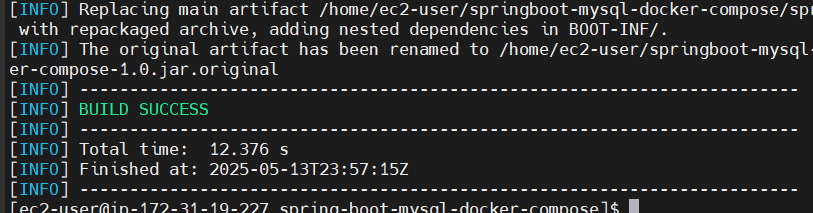
drwxr-xr-x. 5 ec2-user ec2-user 118 May 7 00:48 SpringSecurity\_JWT

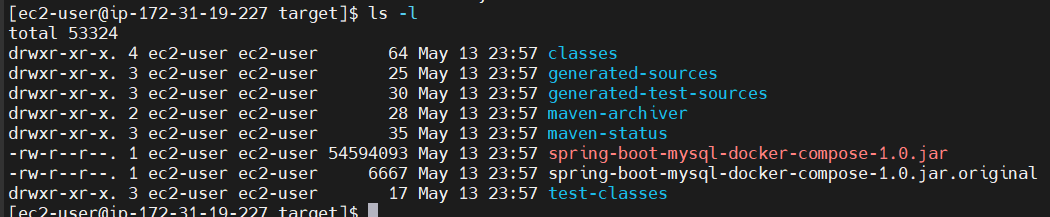
drwxr-xr-x. 4 ec2-user ec2-user 64 May 11 17:54 try-webapp

drwxr-xr-x. 4 ec2-user ec2-user 36 May 13 02:30 WebappCRM

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ mvn clean package

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ mvn clean package -DskipTests





[ec2-user@ip-172-31-19-227 target]$ vi Dockerfile

[ec2-user@ip-172-31-19-227 target]$ cat Dockerfile

FROM openjdk:17

EXPOSE 8080

COPY target/spring-boot-mysql-docker-compose-1.0.jar spring-boot-mysql-docker-compose-1.0.jar

ENTRYPOINT ["java", "-jar", "/spring-boot-mysql-docker-compose-1.0.jar"]

[ec2-user@ip-172-31-19-227 target]$ vi docker-compose.yml

[ec2-user@ip-172-31-19-227 target]$ cat docker-compose.yml

version: "3.8"

services:

musqldb:

image: mysql:8.0

ports:

- "3306:3306"

environments:

- MYSQL\_ROOT\_PASSWORD: root

- MYSQL\_DATABASE: sbm

networks:

- springboot-db-net

application:

build: .

depends\_on:

mysqldb:

ports:

- "8080:8080"

networks:

- springboot-db-net

volumes:

- /data/springboot-app

networks:

springboot-db-net:

[ec2-user@ip-172-31-19-227 springboot-mysql-docker-compose]$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

try-webapp latest fcd71f541cea 2 days ago 468MB

springbootapp latest f06e88a7ddef 2 days ago 497MB

[ec2-user@ip-172-31-19-227 springboot-mysql-docker-compose]$ docker system prune -a

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ ls -l

total 32

-rw-r--r--. 1 ec2-user ec2-user 503 May 14 00:17 docker-compose.yml

-rw-r--r--. 1 ec2-user ec2-user 10665 May 13 23:55 mvnw

-rw-r--r--. 1 ec2-user ec2-user 7061 May 13 23:55 mvnw.cmd

-rw-r--r--. 1 ec2-user ec2-user 2051 May 13 23:55 pom.xml

drwxr-xr-x. 4 ec2-user ec2-user 30 May 13 23:55 src

drwxr-xr-x. 8 ec2-user ec2-user 4096 May 14 00:08 target

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ docker-compose up -d

Updated docker-compose.yml

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ cat docker-compose.yml

version: "3.8"

services:

mysqldb:

image: mysql:8.0

ports:

- "3306:3306"

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: sbm

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

interval: 10s

timeout: 5s

retries: 5

networks:

- springboot-db-net

application:

build: .

depends\_on:

mysqldb:

condition: service\_healthy

ports:

- "8080:8080"

networks:

- springboot-db-net

volumes:

- /data/springboot-app

networks:

springboot-db-net:

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ docker-compose up -d

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ docker-compose ps

Name Command State Ports

---------------------------------------------------------------------------------------------------------------------------------------------

spring-boot-mysql-docker-compose\_mysqldb\_1 docker-entrypoint.sh mysqld Up (healthy) 0.0.0.0:3306->3306/tcp,:::3306->3306/tcp, 33060/tcp

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ docker-compose up -d

Starting spring-boot-mysql-docker-compose\_mysqldb\_1 ... done

Starting spring-boot-mysql-docker-compose\_application\_1 ... done

Docker compose file

Clone the project from Github repo

git clone <https://github.com/Haider7214/springboot-mysql-docker-compose.git>

cd springboot-mysql-docker-compose

Mvn clean package -DskipTest

ls -l target (.jar file will be available)

Create one Dockerfile

vi Dockerfile

FROM openjdk:17

EXPOSE 8080

COPY target/spring-boot-mysql-docker-compose-1.0.jar spring-boot-mysql-docker-compose-1.0.jar

ENTRYPOINT ["java", "-jar", "/spring-boot-mysql-docker-compose-1.0.jar"]

docker-compose up -d ---> Create docker containers using docker-compose

docker-compose ps ---> Check docker containers running or not

docker-compose stop ---> to stop the docker containers

docker-compose start --->

docker-compose down ---> delete docker containers using docker-compose

Stateful containers: Data will be there permanently

Stateless containers: it will not recollect what has happened. Data will be deleted after container deletion

Note: Docker containers by default are stateless

Example: In our spring-boot-mysql-docker-compose app, we used MySQL as docker container to store data and when we stop the containers and re-created these containers we lost the data

To store application data permanently in this case we may have to make docker container as stateful and hence we need to use Docker volumes

Docker Volumes:

Volumes are used to persist data, which is generated by docker container and to avoid data loss.

With the help of docker volumes we can make our containers stateful

docker volume ls ---> display docker volumes

docker volume create <vol-name> ---> create new docker volume

docker volume inspect <vol-name> ---> Inspect docker volumes

docker volume rm <vol-name> ---> remove docker volume

=> Create mount directory in host machine (/home/ec2-user/)

mkdir app

Map this app directory in docker-compose.yml file

[ec2-user@ip-172-31-19-227 ~]$ docker volume ls

DRIVER VOLUME NAME

local 4de4ef265b2a67d1ccb84e0ddf1fa0fc55454dff5808c032e911a39d2935167d

local 41dbdcdaa0404f328aaa6960b467a43c46ed86f3b895aa67bb8479cfcdf11304

local 86983f6c81527b0612934d6bd4f98784b142cc0016659fc27281864ae6148dbd

local b936e5199f3e27d2657ad8010ed846a3177135105ecb49ac6d9941883f477bdc

local bd19e40d4ca00138523a0f5c98bc2f25fb47eabb73b5d895a756137c9613ba5c

local c9fb37b4b4ab19d104cb29a2bec665245d8e1fdb1534952fd6698c717a84b263

[ec2-user@ip-172-31-19-227 ~]$

[ec2-user@ip-172-31-19-227 ~]$ docker volume create demo-volume

demo-volume

[ec2-user@ip-172-31-19-227 ~]$ docker volume ls

DRIVER VOLUME NAME

local 4de4ef265b2a67d1ccb84e0ddf1fa0fc55454dff5808c032e911a39d2935167d

local 41dbdcdaa0404f328aaa6960b467a43c46ed86f3b895aa67bb8479cfcdf11304

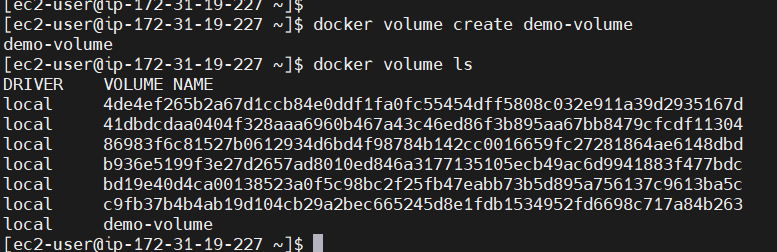
local 86983f6c81527b0612934d6bd4f98784b142cc0016659fc27281864ae6148dbd

local b936e5199f3e27d2657ad8010ed846a3177135105ecb49ac6d9941883f477bdc

local bd19e40d4ca00138523a0f5c98bc2f25fb47eabb73b5d895a756137c9613ba5c

local c9fb37b4b4ab19d104cb29a2bec665245d8e1fdb1534952fd6698c717a84b263

local demo-volume



[ec2-user@ip-172-31-19-227 ~]$ docker volume inspect demo-volume

[

{

"CreatedAt": "2025-05-14T02:57:21Z",

"Driver": "local",

"Labels": null,

"Mountpoint": "/var/lib/docker/volumes/demo-volume/\_data",

"Name": "demo-volume",

"Options": null,

"Scope": "local"

}

]

[ec2-user@ip-172-31-19-227 ~]$ docker volume ls

DRIVER VOLUME NAME

local 4de4ef265b2a67d1ccb84e0ddf1fa0fc55454dff5808c032e911a39d2935167d

local 41dbdcdaa0404f328aaa6960b467a43c46ed86f3b895aa67bb8479cfcdf11304

local 86983f6c81527b0612934d6bd4f98784b142cc0016659fc27281864ae6148dbd

local b936e5199f3e27d2657ad8010ed846a3177135105ecb49ac6d9941883f477bdc

local bd19e40d4ca00138523a0f5c98bc2f25fb47eabb73b5d895a756137c9613ba5c

local c9fb37b4b4ab19d104cb29a2bec665245d8e1fdb1534952fd6698c717a84b263

local demo-volume

[ec2-user@ip-172-31-19-227 ~]$ docker volume inspect demo-volume

[

{

"CreatedAt": "2025-05-14T02:57:21Z",

"Driver": "local",

"Labels": null,

"Mountpoint": "/var/lib/docker/volumes/demo-volume/\_data",

"Name": "demo-volume",

"Options": null,

"Scope": "local"

}

]

Docker container 2

Docker container 1

Docker volume

We have two Docker containers, we store data in Docker container 2 lets say. Whatever data is there in Docker container 2, it will be stored in Docker volume permanently. So it will become Stateful.

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ docker volume rm demo-volume

demo-volume

[ec2-user@ip-172-31-19-227 ~]$ mkdir app

drwxr-xr-x. 2 ec2-user ec2-user 6 May 15 00:53 app

-rw-r--r--. 1 ec2-user ec2-user 704 May 14 00:47 docker-compose.yml

-rw-r--r--. 1 ec2-user ec2-user 197 May 14 00:21 Dockerfile

-rw-r--r--. 1 ec2-user ec2-user 10665 May 13 23:55 mvnw

-rw-r--r--. 1 ec2-user ec2-user 7061 May 13 23:55 mvnw.cmd

-rw-r--r--. 1 ec2-user ec2-user 2051 May 13 23:55 pom.xml

drwxr-xr-x. 4 ec2-user ec2-user 30 May 13 23:55 src

drwxr-xr-x. 8 ec2-user ec2-user 4096 May 15 00:54 target

Altered docker-compose file to store data into volume

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ cat docker-compose.yml

version: "3.8"

services:

mysqldb:

image: mysql:8.0

ports:

- "3306:3306"

environment:

- MYSQL\_ROOT\_PASSWORD=root

- MYSQL\_DATABASE=sbm

volumes:

- .app:/var/lib/mysql

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

interval: 10s

timeout: 5s

retries: 5

networks:

- springboot-db-net

application:

build: .

depends\_on:

mysqldb:

condition: service\_healthy

ports:

- "8080:8080"

networks:

- springboot-db-net

volumes:

- /data/springboot-app

networks:

springboot-db-net:

Working version:

version: "3.8"

services:

mysqldb:

image: mysql:8.0

ports:

- "3306:3306"

environment:

- MYSQL\_ROOT\_PASSWORD=root

- MYSQL\_DATABASE=sbm

volumes:

- ./mysql-data:/var/lib/mysql # local folder for MySQL data

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

interval: 10s

timeout: 5s

retries: 5

networks:

- springboot-db-net

application:

build: .

depends\_on:

mysqldb:

condition: service\_healthy

ports:

- "8080:8080"

networks:

- springboot-db-net

volumes:

- ./springboot-app:/app # local folder mounted to /app in container

networks:

springboot-db-net:

For docker volume: volumes:

- ./app:/var/lib/mysql # local folder for MySQL data

[ec2-user@ip-172-31-19-227 spring-boot-mysql-docker-compose]$ cat docker-compose.yml

version: "3.8"

services:

mysqldb:

image: mysql:8.0

ports:

- "3306:3306"

environment:

- MYSQL\_ROOT\_PASSWORD=root123

- MYSQL\_DATABASE=sbm

volumes:

- ./app:/var/lib/mysql # local folder for MySQL data

healthcheck:

test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]

interval: 10s

timeout: 5s

retries: 5

networks:

- springboot-db-net

application:

build: .

depends\_on:

mysqldb:

condition: service\_healthy

ports:

- "8080:8080"

networks:

- springboot-db-net

volumes:

- ./springboot-app:/app # local folder mounted to /app in container

networks:

springboot-db-net:

Docker Swarm:

To manage your containers: It is a orchestration platform like Kubernetes

Docker swarm is used to setup docker cluster. Cluster is referring to group of servers

It is a way to run and manage many docker containers across multiple machines

It will handle where to run each container, load-balancing traffic across them and keeping everything in sync

We define a service (eg webapp) and Swarm manages its containers and we can easily also scale up/down the number of containers

Worker node1

Master node

Worker node2

Master node assigns tasks to multiple slave nodes

It will use worker machines to manage and create containers. If you want to make those containers highly available, then docker swarm comes into picture where in it will allow you to manage those containers not in one server but multiple servers (multiple machines). Docker swarm will also take care of load-balancing traffic. If you want to manage something in Docker swarm you have to define in service. I can scale up the machines as well using Docker swarm, one problem is, we have to manually scale up and down. For fully automatic scaling up or load-balancing, then Kubernetes comes into picture. Cluster is a group of machines working together

Docker Swarm cluster setup

Create 3 EC2 machines Ubuntu and install Docker in all of them

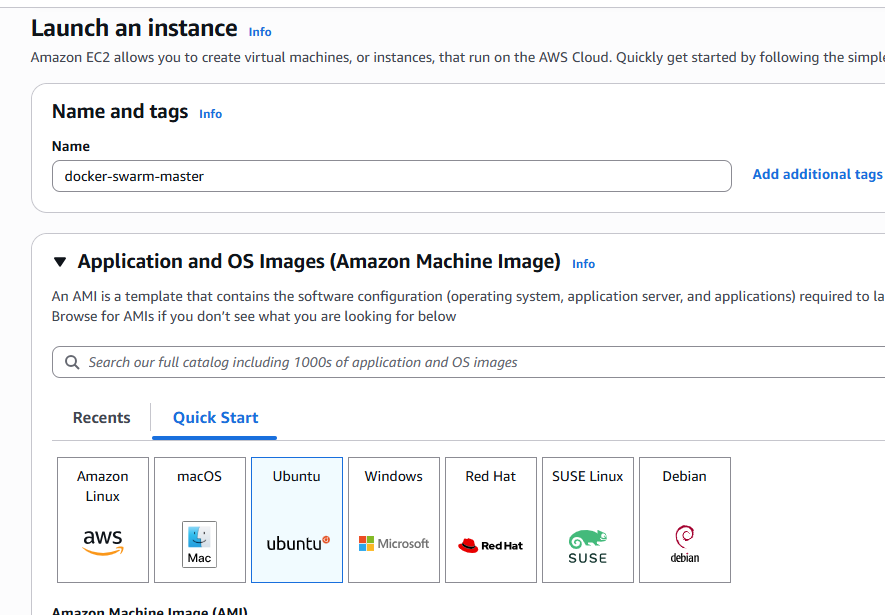
For Swarm cluster communications enable 2377 port in the security group

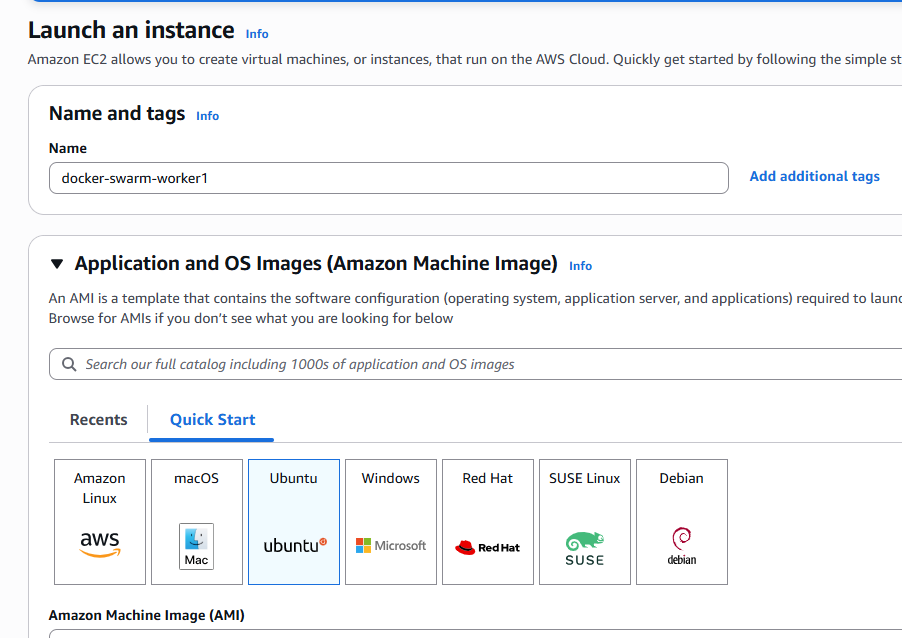
1. Master node
2. Worker node

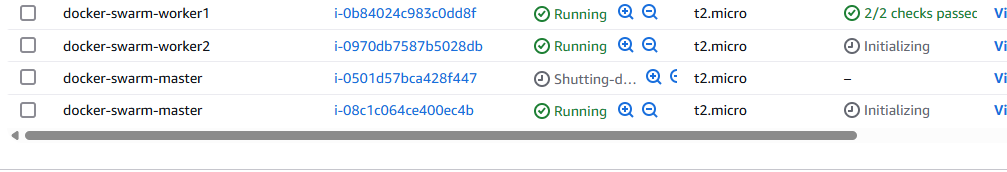
After connecting to VMs, install docker in all 3 machines

Check docker -v in the machines

Initialize docker swarm cluster







curl -fsSL https://get.docker.com -o get-docker.sh && sudo sh get-docker.sh

Install Docker in all 3 machines

Run in Master node: sudo docker swarm init --advertise-addr <PrivateIP>

Run in Worker nodes: sudo docker swarm join --token SWMTKN-1-17cpjm9mg5queqqocst1c27mmf5jcxqavkjw6hu659wia30c4h-8acpswg6zemholta1su7upx1p 172.31.7.201:2377

We deploy our application as a service in Docker swarm

Service is a collection of one or more containers of same image

Replica is a type of service in Docker swarm, which is default

sudo docker service create --name <ServiceName> -<hostport>:<containerport> <imagename:tag>

By default 1 replica is created

We can check the service created

Sudo docker service ls

We can scale docker service --> docker service scale <service-name>=<no of replicas>

To see Service details:

sudo docker service ps <service-name>

sudo docker service rm <service-name> ---> to remove the docker service

By Docker swarm, we can make sure our container is available in multiple machines

**Go to Master node**

ubuntu@ip-172-31-7-201:~$ mkdir master

ubuntu@ip-172-31-7-201:~$ ls -l

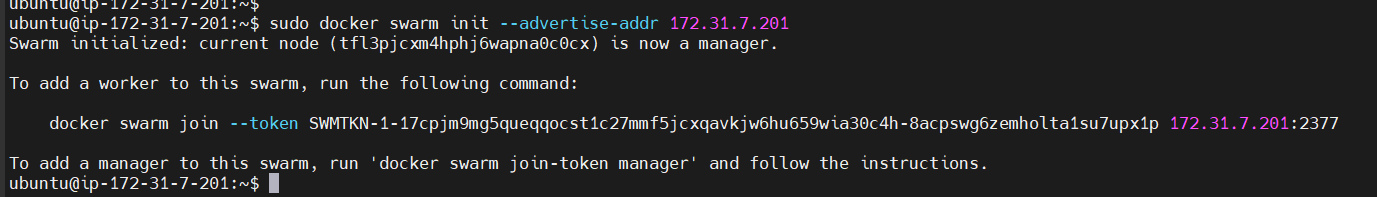
total 24

-rw-rw-r-- 1 ubuntu ubuntu 20443 May 15 02:57 get-docker.sh

drwxrwxr-x 2 ubuntu ubuntu 4096 May 16 00:35 master

ubuntu@ip-172-31-7-201:~$ sudo docker swarm init --advertise-addr 172.31.7.201

ubuntu@ip-172-31-7-201:~$ sudo docker swarm init --advertise-addr <PrivateIP>



Current node is now a manager next we need to add workers to this node

ubuntu@ip-172-31-7-201:~$ sudo docker swarm init --advertise-addr 172.31.7.201

Swarm initialized: current node (tfl3pjcxm4hphj6wapna0c0cx) is now a manager.

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

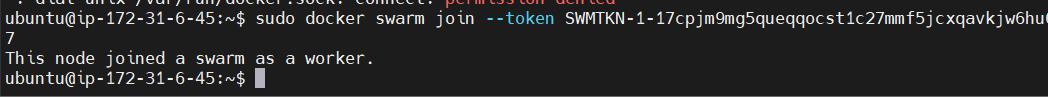
docker swarm join --token SWMTKN-1-17cpjm9mg5queqqocst1c27mmf5jcxqavkjw6hu659wia30c4h-8acpswg6zemholta1su7upx1p 172.31.7.201:2377

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

Go to worker nodes

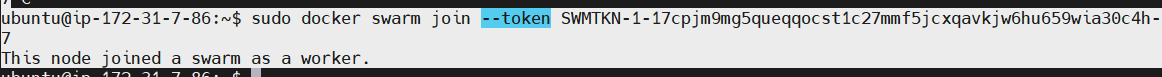
ubuntu@ip-172-31-6-45:~$ sudo docker swarm join --token SWMTKN-1-17cpjm9mg5queqqocst1c27mmf5jcxqavkjw6hu659wia30c4h-8acpswg6zemholta1su7upx1p 172.31.7.201:2377

This node joined a swarm as a worker.



ubuntu@ip-172-31-7-86:~$ sudo docker swarm join --token SWMTKN-1-17cpjm9mg5queqqocst1c27mmf5jcxqavkjw6hu659wia30c4h-8acpswg6zemholta1su7upx1p 172.31.7.201:2377

This node joined a swarm as a worker.



Go to Master node

ubuntu@ip-172-31-7-201:~$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu@ip-172-31-7-201:~$ sudo docker pull edydockers/sms-frontend:dev-31

ubuntu@ip-172-31-7-201:~$ sudo docker pull edydockers/sms-frontend:dev-31

dev-31: Pulling from edydockers/sms-frontend

f18232174bc9: Pull complete

61ca4f733c80: Pull complete

b464cfdf2a63: Pull complete

d7e507024086: Pull complete

81bd8ed7ec67: Pull complete

197eb75867ef: Pull complete

34a64644b756: Pull complete

39c2ddfd6010: Pull complete

6dfec665e776: Pull complete

12564a4dfdde: Pull complete

Digest: sha256:dc6b4833d144930b1c5dabda66f37ecbdcd7820d44980ed5fcb9ea227d114e25

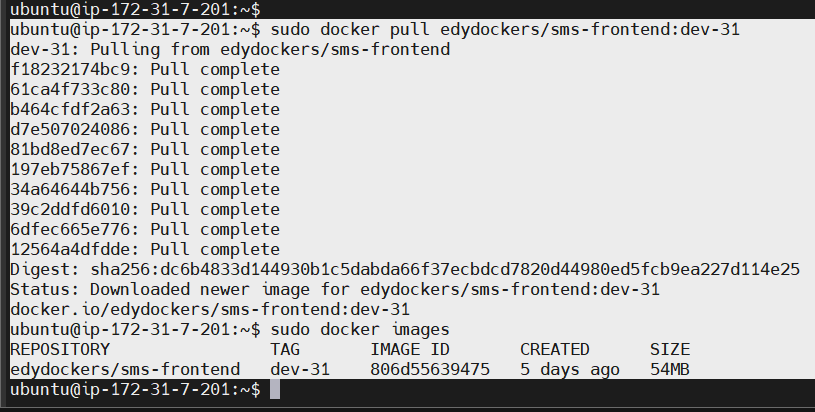
Status: Downloaded newer image for edydockers/sms-frontend:dev-31

docker.io/edydockers/sms-frontend:dev-31

ubuntu@ip-172-31-7-201:~$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

edydockers/sms-frontend dev-31 806d55639475 5 days ago 54MB



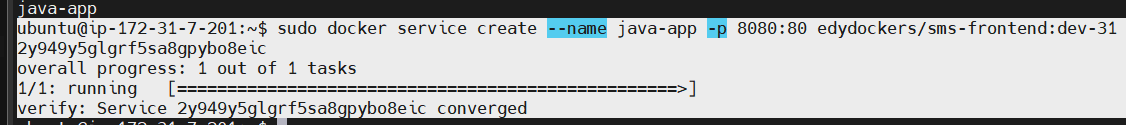
ubuntu@ip-172-31-7-201:~$ sudo docker service create --name java-app -p 8080:80 edydockers/sms-frontend:dev-31

2y949y5glgrf5sa8gpybo8eic

overall progress: 1 out of 1 tasks

1/1: running [==================================================>]

verify: Service 2y949y5glgrf5sa8gpybo8eic converged

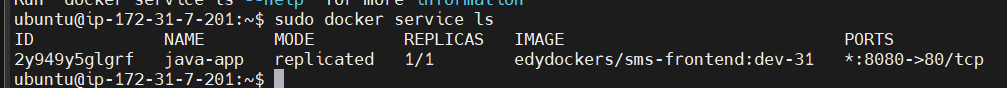


Master node:

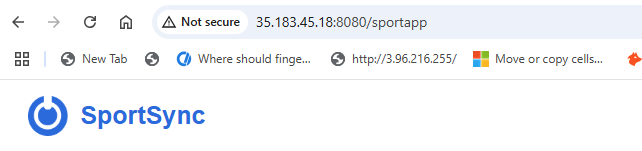
ubuntu@ip-172-31-7-201:~$ sudo docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

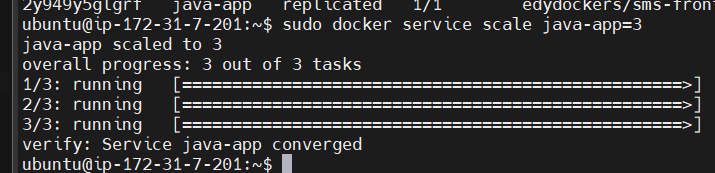
2y949y5glgrf java-app replicated 1/1 edydockers/sms-frontend:dev-31 \*:8080->80/tcp



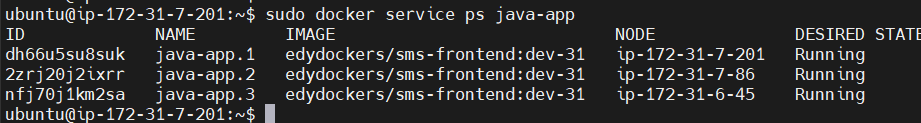
<http://35.183.45.18:8080/sportapp>



ubuntu@ip-172-31-7-201:~$ sudo docker service scale java-app=3



ubuntu@ip-172-31-7-201:~$ sudo docker service ps java-app



Now I copy publicIP of worker node 1, same go to worker node 2 and check

<http://35.182.212.31:8080/sportapp>

