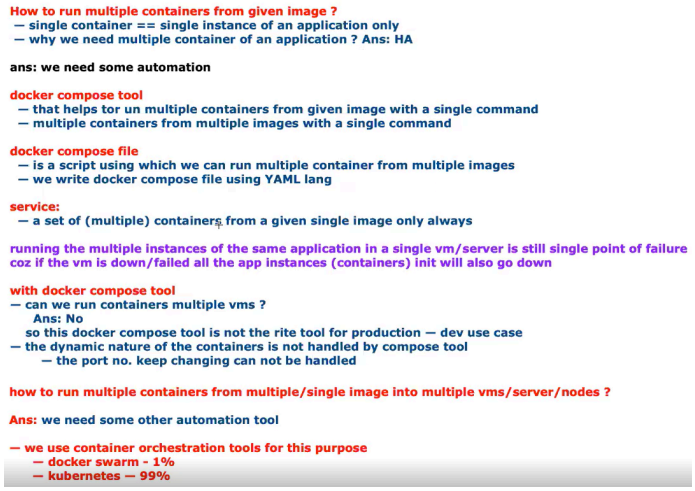
Docker Hands-on:5

In this we are going to learn how to make our container available 24/7 without any delay highly available.

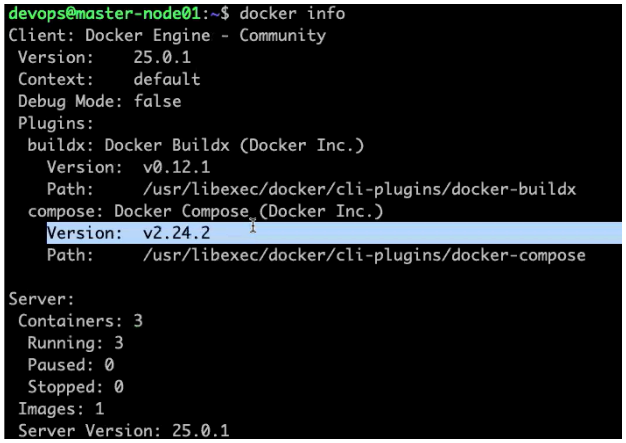
First thing which we need to keep in mind that only one instance of the container is not capable for make container highly available to our application

As if now we are creating the 3instance of the same instance if in case one failed other 2 will take care of it. With the below command:
Docker run -d -P nginx:latest

A single container is always a single instance of an application running on the server to make our application highly available.Here we need to run a multiple container of the same image multiple instance of the same application.



Docker compose is actual a plugin which is the part of the docker installation already we need not to install anything specially.



docker compose version
To work on the “docker compose” we need write the YAML code

YAML file

```
devops@master-node01:~$ cat compose.yaml
volumes:
  nginxlogs:
networks:
  mynet:
services:
   app1:  # this is the service name / can be any name
    image: nginx:latest
    deploy:
      replicas: 3
    ports:
      - 80
    volumes:
      - nginxlogs:/var/log/nginx
    networks:
      - mynet
   app2:
     image: tomcat
     deploy:
       replicas: 2
     ports:
       - 8080
     networks:
       - mynet
   #app3:
```

Docker compose --file compose.yaml up -d

```
root@ip-172-31-47-65:~# docker container ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                          NAMES
7886af770432   nginx:latest   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   0.0.0.0:32770->80/tcp, :::32770->80/tcp        root-
app1-1
b19b5c8295c7   nginx:latest   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   0.0.0.0:32772->80/tcp, :::32772->80/tcp        root-
app1-3
660c2f9970f7   nginx:latest   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   0.0.0.0:32773->80/tcp, :::32773->80/tcp        root-
app1-2
a25f07382993   tomcat         "catalina.sh run"        10 minutes ago   Up 10 minutes   0.0.0.0:32769->8080/tcp, :::32769->8080/tcp    root-
app2-2
fde2add5e4fc   tomcat         "catalina.sh run"        10 minutes ago   Up 10 minutes   0.0.0.0:32771->8080/tcp, :::32771->8080/tcp    root-
app2-1
810dbec83700   nginx:latest   "/docker-entrypoint.…"   23 minutes ago   Up 23 minutes   0.0.0.0:32768->80/tcp, :::32768->80/tcp        unruf
fled_lovelace
```

If i want to delete whatever i created with the above compose file :
Docker compose --file compose.yaml down

If i want to scaleup any of the app container use the below command:
Docker compose --file compose.yaml up --scale app1=5 --scale app2=3 -d

==========

volumes:
  nginxlogs: null
networks:
  mynet: null
services:

```
  app1:
    image: nginx:latest
    deploy:
      replicas: 3
    ports:
      - 80
    volumes:
      - nginxlogs:/var/log/nginx
    networks:
      - mynet
  app2:
    image: tomcat
    deploy:
      replicas: 2
    ports:
      - 8080
    networks:
      - mynet
```

==========

##################Not that much use in the market####################

Container orchestration with docker / docker swarm

how to run multiple containers from single / multiple images ?
how to deploy the containers into multiple vms/servers/nodes ?
how to manage the dynamic nature of the containers ?

Ans: we need automation

Container Orchestration:
— helps you to deploy the cont(s) across the vms/servers/nodes
— manage the dynamic nature of the containers
— provides various other benefits
ex: scale up / down into multiple vms
request routing into multiple cotnainer with dynamic nature (HA)
desired state / auto healing

container orchestration tools
— docker swarm — free — 1%
— kubernetes (k8s) — free — 99%

to implement container orchestration we need minimumm 2 vms/servers/nodes

— 1 server/vm/node must be acting as manager / control plane (can be multiple)
— 1 server/vm/node must be acting as worker (like team member) (can be n number)

So,for this concept we have created the 2Vm 1 is master server and 2 is workers
Install devops in all 3Vm's

Docker Swarm
- is cont orch sol with atleast one master node / one worker node
- is a feature given as part of the docker container runtime software
- by default swarm feat is in inactive state

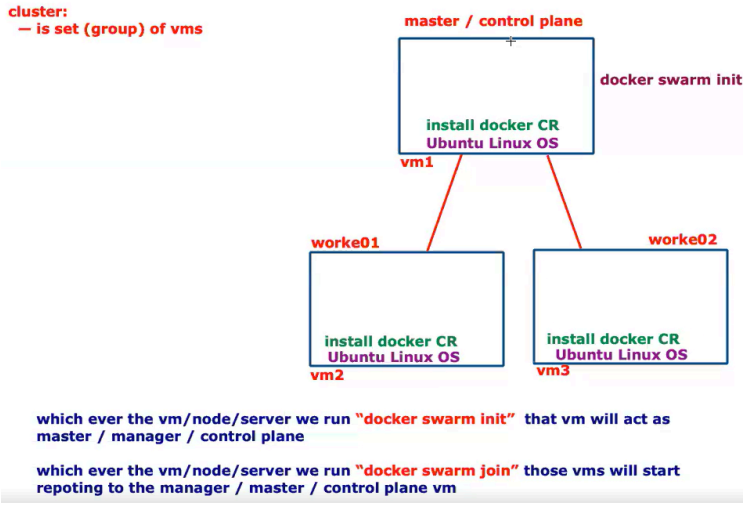
“docker info” to see the below swarm active of inactive.

```
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
Swarm: inactive
```

```
devops@master-node01:~$ docker swarm init
Swarm initialized: current node (lc36ycvv2x5wdqtpcwz2u21bc) is now a manager.

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

    docker swarm join --token SWMTKN-1-23hpjvla0f0rx7xcoxqnm98512o9a43hv8cq2i39z1vpgsw4oj-czpk2bw601r752qyu70tmvr1t 172.31.95.11:2377

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

devops@master-node01:~$ docker info | grep -i swarm
 Swarm: active
```

After init we got the output as Ex:docker swarm join --token SWMTKN-1-4h8whjrjk5ft6mwvuhkeb89s2xkfimfqc7bz4pvyfvn10lkly7-7tdsc5tpkwr4fax4eznh8st3a 172.31.38.179:2377

We need to copy this and paste it to the workers1 and workers2
By the above step the workers1 and 2 are connected with the master server and now they will listen the master



IN the master server if master server want to check who is working in my downline use the below command:
“Docker node ls “

```
root@ip-172-31-38-179:~# docker node ls
ID                            HOSTNAME            STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
woc30gmpl3cgkwf0qecs4qmt9 *   ip-172-31-38-179    Ready     Active         Leader           26.0.0
tmg9fbk92fdf4848h9szbjwlz     ip-172-31-44-117    Ready     Active                          26.0.0
v8hhzgor3t5wqlojgythf5h9z     ip-172-31-47-71     Ready     Active                          26.0.0
root@ip-172-31-38-179:~#
```

If a manager wants to rejoin as a worker he is not able to join as he is joined as a leader already.

Stop the docker services
Systemctl stop docker.service
Systemctl stop docker.socket (to stop the demon services)

Docker is the client server architecture.

If we want to inspect the node :
Docker node ls
Docker node inspect node_name

To perform any task we need not to go on the workers master server can perform from their end only to the workers.

Docker service create --name app1 --replicas 5 -p 9080:3000 harshit407/cizaar:v1
Docker service create --name app2 --replicas 2 -p 9088:80 nginx:latest

```
root@ip-172-31-38-179:~# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                  PORTS
pzn6xjcjt99e   app1      replicated   5/5        harshit407/cizaar:v1   *:9080->3000/tcp
c9a7vda17m0a   app2      replicated   3/3        nginx:latest           *:9990->80/tcp
```

```
root@ip-172-31-38-179:~# docker service ps app1
ID             NAME      IMAGE                  NODE               DESIRED STATE   CURRENT STATE           ERROR     PORTS
tcjd7y5yrx5z   app1.1    harshit407/cizaar:v1   ip-172-31-47-71    Running         Running 2 minutes ago
830bpvrq47oy   app1.2    harshit407/cizaar:v1   ip-172-31-38-179   Running         Running 2 minutes ago
vxwv43jz3c6p   app1.3    harshit407/cizaar:v1   ip-172-31-47-71    Running         Running 2 minutes ago
owvh0wkigng2   app1.4    harshit407/cizaar:v1   ip-172-31-38-179   Running         Running 2 minutes ago
2s7vt6739epx   app1.5    harshit407/cizaar:v1   ip-172-31-44-117   Running         Running 2 minutes ago
root@ip-172-31-38-179:~# docker service ps app2
ID             NAME      IMAGE          NODE               DESIRED STATE   CURRENT STATE               ERROR     PORTS
ketyq9cg4psq   app2.1    nginx:latest   ip-172-31-44-117   Running         Running about a minute ago
22hb3xqomvig   app2.2    nginx:latest   ip-172-31-38-179   Running         Running about a minute ago
5jdcj9kh5819   app2.3    nginx:latest   ip-172-31-47-71    Running         Running about a minute ago
```

Output from the worker 1

```
root@ip-172-31-44-117:~# docker container ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS      NAMES
cddbf16d57db   nginx:latest           "/docker-entrypoint.…"   2 minutes ago    Up 2 minutes    80/tcp     app2.1.ketyq9cg4psqncvxbt0tf8ygu
1a67e995d393   harshit407/cizaar:v1   "/opt/tomcat/bin/cat…"   3 minutes ago    Up 3 minutes    8080/tcp   app1.5.2s7vt6739epxal3d3dp58i5yu
```

Output from the worker 2

```
root@ip-172-31-47-71:~# docker container ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS      NAMES
0dfe0e5588ca   nginx:latest           "/docker-entrypoint.…"   3 minutes ago    Up 3 minutes    80/tcp     app2.3.5jdcj9kh5819xn3ho2ybzisd6
6c6b3fd1ac42   harshit407/cizaar:v1   "/opt/tomcat/bin/cat…"   4 minutes ago    Up 4 minutes    8080/tcp   app1.3.vxwv43jz3c6p5g0t0dbwz351x
9cf5ca0161d3   harshit407/cizaar:v1   "/opt/tomcat/bin/cat…"   4 minutes ago    Up 4 minutes    8080/tcp   app1.1.tcjd7y5yrx5zp3c4l4d4xtsyz
```

From master server we are deleting the app services:

```
root@ip-172-31-38-179:~# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                  PORTS
pzn6xjcjt99e   app1      replicated   5/5        harshit407/cizaar:v1   *:9080->3000/tcp
c9a7vda17m0a   app2      replicated   3/3        nginx:latest           *:9990->80/tcp
root@ip-172-31-38-179:~# docker service rm pzn6xjcjt99e
pzn6xjcjt99e
root@ip-172-31-38-179:~# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
c9a7vda17m0a   app2      replicated   3/3        nginx:latest   *:9990->80/tcp
```

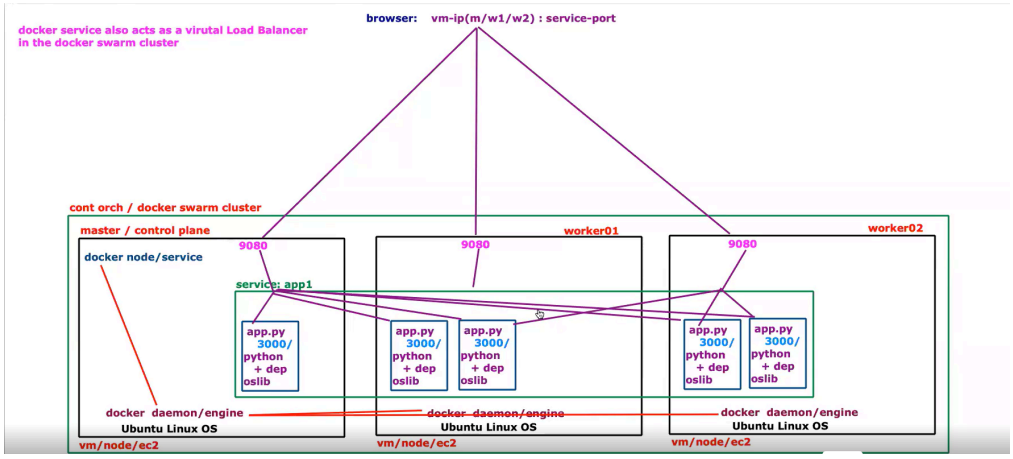
If we want to scale any application we can use the below command:

Docker service scale app1=10

For scale down:
Docker service scale app1=4
Everytime whenever we will create or scale up the application instance it will distribute it to the multiple workers to achieve the high availability.
From here all the above things we can see the container application is creating on the other nodes also and if in case any nodes crashes other notes/workers or on master server it is running already which helping us to achieve the high available application for the end user.

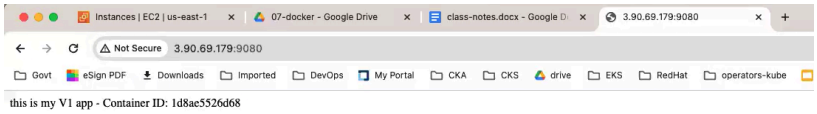
If in case someone deleted the anyone application process from the worker so in that situation orchestration is very intelligent it will automatically created the process again if in case anychnge done on the worker.
In the below output we can see that it will automatically created by the orchestration and this is called auto healing and maintain the desired state.

```
root@ip-172-31-44-117:~# docker container ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
cddbf16d57db   nginx:latest   "/docker-entrypoint.…"   19 minutes ago   Up 19 minutes   80/tcp    app2.1.ketyq9cg4psqncvxbt0tf8ygu
root@ip-172-31-44-117:~# docker container rm -f cddbf16d57db
cddbf16d57db
root@ip-172-31-44-117:~# docker container ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@ip-172-31-44-117:~# docker container ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@ip-172-31-44-117:~# docker container ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS         PORTS     NAMES
4a5fd27cc66f   nginx:latest   "/docker-entrypoint.…"   7 seconds ago    Up 1 second    80/tcp    app2.1.nqymxor4bg4b83eif9is9nnd4
root@ip-172-31-44-117:~#
```

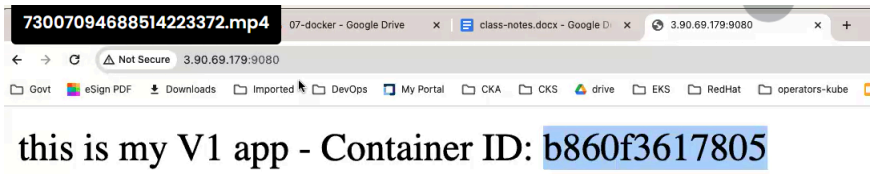


Main agenda of the above diagram is to whenever we hit the request on the application on that time we are not sure the request will go anywhere they want on any worker1/worker2/master on anyone request can go randomly…

Ex:Here we are running from the master node:



As if now the request gone on the worker1 application
We refresh the page and again it went on the master application.



Whatever we do in the application this command output will update automatically
On the master run the below command:
Docker service logs -f app1

——--

```
root@ip-172-31-38-179:~# docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
6b44766f2578   bridge            bridge    local
5fd97a41f4e4   docker_gwbridge   bridge    local
fb1512dca35e   host              host      local
ccex1m9vhkt6   ingress           overlay   swarm
9a09e57cc9db   none              null      local
```

Here we are seeing the “ingress”

```
root@ip-172-31-38-179:~# docker network inspect ingress
[
    {
        "Name": "ingress",
        "Id": "ccex1m9vhkt68hedv9lptlv0r",
        "Created": "2024-04-06T10:51:36.779553747Z",
        "Scope": "swarm",
        "Driver": "overlay"
```

```
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/24",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
```

```
"Labels": {},
"Peers": [
    {
        "Name": "506b1ade2476",
        "IP": "172.31.38.179"
    },
    {
        "Name": "ecec0bc84c47",
        "IP": "172.31.47.71"
    },
    {
        "Name": "0379cbbdbcc6",
        "IP": "172.31.44.117"
    }
]
```


By the above output we can see this is having the dynamic IPaddress and containing all the IP or worker and master

In docker default network is ingress network

We can also create the customer overlay network with the below command:
Docker network create myoverlay --driver overlay

If we want to create the subnet also inside the overlay network else by default it will choose different subnets:
Docker network create myoverlay --driver overlay --subnet

```
devops@master-node01:~$ docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
5e9155a4602e   bridge            bridge    local
13af97a09b92   docker_gwbridge   bridge    local
ae2d9f95a544   host              host      local
x9anoncogfjr   ingress           overlay   swarm
e6zs8i51endo   myoverlay         overlay   swarm
1feb4c289ec2   none              null      local
```

```
devops@master-node01:~$ docker network inspect myoverlay
[
    {
        "Name": "myoverlay",
        "Id": "e6zs8i51endo8l6a5vfu9oohx",
        "Created": "2024-01-30T03:30:31.122661Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.1.0/24",
                    "Gateway": "10.0.1.1"
```

Docker service create --name app2 --network myoverlay --replicas 6 -p 9000:80 nginx:latest

```
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS             PORTS     NAMES
111ce63314eb   nginx:latest   "/docker-entrypoint.…"   36 seconds ago      Up 35 seconds      80/tcp    app3.6.cc8mc8qhr8z6h4599qisu9r26
d2209143c80d   nginx:latest   "/docker-entrypoint.…"   36 seconds ago      Up 35 seconds      80/tcp    app3.3.4yhueavjt1l48bnu5n61vw6y7
0b66bbf4158c   nginx:latest   "/docker-entrypoint.…"   About an hour ago   Up About an hour   80/tcp    app2.2.22hb3xqomvig75gw2ivddcv5w
```

Docker container inspect

```
root@ip-172-31-38-179:~# docker container inspect 111ce63314eb
[
    {
        "Id": "111ce63314ebda0336a23469596bf4ca2f07ed7654cde6d5f5e9adfbdb2d049f",
        "Created": "2024-04-06T12:33:41.070122405Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
            "nginx",
```

```
            },
            "myoverlay": {
                "IPAMConfig": {
                    "IPv4Address": "10.0.1.7"
                },
                "Links": null,
                "Aliases": null,
                "MacAddress": "02:42:0a:00:01:07",
                "NetworkID": "nhcrc8ucmvt2sa427338b89n3",
                "EndpointID": "0d5ffda4a62f2e1c223937245d26b75c0c7f5def3efe152d043f67bb17
                "Gateway": "",
                "IPAddress": "10.0.1.7",
```

IN the above output we can see now it is using the overlay network instead of ingress.

==========END DOCKER##########