INTERVIEW QUESTIONS

1) What is docker mainly useful for and why?

A: Docker is a containerization tool which reduces the setup of all dependencies to deploy an application instead here we pack all the dependencies as an image and we run the image to deploy the application. With the help of docker we can implement the microservices architecture deployment of an application where in case any particular service fails or any bug is found instead of stopping the entire production server that has all the microservices we can just stop that one particular container that has issue and can the remaining can run uninterruptedly.

2) What are the components of a docker?

A: The components of a docker are docker client, docker host and docker registry. Docker client is the CLI used to interact with the Docker Daemon or in simple words, where we write the docker commands, docker host is the machine where docker is installed and docker registry is the hub or place where we can store and pull the images from.

3) What is Docker Daemon and what is exactly does?

A: Docker Daemon is a service that runs in the background of the Docker which is responsible for creating containers, running and managing containers. It's the engine of the docker that carries all the operations and requests that are related to Docker.

4) What is dockerfile and why is it useful or needed?

A: A dockerfile is a text file that has set of instructions which helps to automate the image creation. In Dockerfile "D" should always be in uppercase and the start components in the file should be in uppercase.

5) How to create a container and go inside the container?

A: To create a container we first need an image and in case if there's no image it pulls from the docker registry and then creates a container as shown below.

To create a container: docker run -it --name container-name image-name

To go inside the container: docker attach container-name.

If you create a new container you directly go inside the container but if a container is already created and stopped/exited then if you want to go then you need to use the command "**docker attach container-name**".

6) How do you move the code or files from one container to another container or to run the same application of one container in another container?

A: First we need to create a container (cont-1), then create an image from the container and at last now if you run the image or in other words create a new container (cont-2) with the created image of first

container (cont-1) then you get all files.

i) docker run -it --name cont-name (cont-1) image-name (mahi.jpg) - - > Container is created.

ii) docker commit cont-name (cont-1) image-name - - > An image is created from the container.

iii) docker run -it --name mahi.jpg - - > A container is created with the image created from the first container.

7) How to reduce the size of a docker image?

A: The size of the docker image can be reduced with the help of a multi stage build and also with the help of Distroless images.

A multi stage build is a process which includes multiple stages in a single Dockerfile where we have multiple FROM components in the Dockerfile. Each FROM begins a new stage for the build, you can selectively copy the artifacts from one stage to another, leaving behind everything that you don't want in the final image, that eventually reduces the size of the image.

Ex: https://docs.docker.com/build/building/multi-stage/

```
# syntax=docker/dockerfile:1
FROM golang:1.21
WORKDIR /src
COPY <<EOF ./main.go
package main

import "fmt"

func main() {
  fmt.Println("hello, world")
}
EOF
RUN go build -o /bin/hello ./main.go

FROM scratch
COPY --from=0 /bin/hello /bin/hello
CMD ["/bin/hello"]
```

A distroless image contain only your application and it's runtime dependencies, they don't contain any package managers, shells or any other programs you'd expect to find it in a standard Linux distribution. Like if you install ubuntu, you get wget, curl, apt, ls and grep and this occupy the size in the image and distroless images doesn't contain all these things. The distroless images are the ones which is very small in size and provides high security which is less prone to attack by the hackers.

https://github.com/GoogleContainerTools/distroless?tab=readme-ov-file

8) What are the challenges that you faced in real time while using docker?

A: The size of the image used to be very huge in size which used to eat a lot of memory so I've learnt about the distroless images and multi-stage build and that really helped a lot in terms of security, size and overall performance as well.

9) How do you check the container name or ID associated to an image or how to find out a container name or ID if you only have image name?

A: If you want to limit your search to only running containers "**docker ps --filter ancestor=image-name**" and if you want to include your search in the list of running and stopped containers then you can use **"docker ps -a --filter ancestor=image-name ".**

The list of docker images.

```
[root@ip-172-31-84-187 ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
image1              latest    91e7ac150752   4 hours ago      146MB
nginx               latest    fffffc90d343   13 days ago      188MB
ubuntu              latest    35a88802559d   3 weeks ago      78.1MB
shaikmustafa/dm     latest    adc857dee30b   13 months ago    146MB
[root@ip-172-31-84-187 ~]#
```

Using grep to find the container name using an image.

```
[root@ip-172-31-84-187 ~]# docker ps -a | grep image1
e7459d976a92   image1          "/docker-entrypoint.…"   4 hours ago      Exited (0) 2 minutes ago                                  cont-3
[root@ip-172-31-84-187 ~]# docker ps | grep image1
[root@ip-172-31-84-187 ~]# docker ps -a | grep nginx
cdfb6e8781f6   nginx           "/docker-entrypoint.…"   12 minutes ago   Up 12 minutes            0.0.0.0:2222->80/tcp, :::2222->80/tcp   cont-5
2395ed6eeda4   nginx           "/docker-entrypoint.…"   4 hours ago      Exited (0) 3 minutes ago                                  cont-1
```

Using --filter and ancestor.

```
[root@ip-172-31-84-187 ~]# docker ps --filter ancestor=shaikmustafa/dm
CONTAINER ID   IMAGE     COMMAND                  CREATED       STATUS       PORTS     NAMES
e7459d976a92   image1    "/docker-entrypoint.…"   3 hours ago   Up 3 hours   80/tcp    cont-3
[root@ip-172-31-84-187 ~]#
```

10) How do you expose an application inside the docker container to make it available to the public?

A: In order to expose the application inside the docker container to access we need to publish a port number while creating a container itself and you can't do it for running containers.

docker run -itd --name cont-name -p hostport:containerport image-name

11) What are the components of a Dockerfile?

A: The below are the components of a Dockerfile.

FROM: It is used to define the image name.

LABEL: It is used to define the author name.

COPY: It is used to copy the files from local to container.

ADD: It also does the same as COPY but in addition to that it also downloads the internet objects.

Ex: **ADD URL destination**

RUN: It is used to perform any command during the build.

CMD: It is used to perform a command during the run time which means while running the image build by dockerfile or while creating a container.

ENTRYPOINT: It is used to pass the values in run time and has high priority than CMD.

WORKDIR: It is used to create a folder inside the default path of the container (/) and as soon as the container is built it goes into that path directly without /

ARG: It is used to pass the variables and it overwrites the values that were previously mentioned in the ARG and can be passed during the run time.

ENV: It can't be passed during the run time and it doesn't overwrite.

EXPOSE: It is a component which is used to listen the incoming requests on the port 80, let's say we've a web server httpd it works on port 80 and if we mention expose 80 that means the requests goes to web servers in this way useful and it doesn't publish any port it is useful only for documentation purpose.

12) What are the different ways to create a volume, mount a volume?

A: You can create a volume individually with a command, a dockerfile and also you while creating a new container as well.

i) **docker volume create volume-name**

ii) **docker run -itd --name cont-name -v /vol-name image-name**

MOUNTING A VOLUME:

Let's say you've a volume that has some code or files in local and if you want to attach it to a container then you can do it by using the following command.

docker run -itd --name cont-name **--mount source=vol-name (in local),destination=/new-vol-name inside the container** image-name

Another way is you can mount a

13) How data replication is possible or implemented in docker?

A: Data replication is possible in docker with the help of docker volumes.

If we've two containers cont-1 and cont-2, in cont-1 we have like 10 files and if you create an image from cont-1 and then create a new container from created image then we get all the data from cont-1 but if any data is deleted or added we don't get to see those changes and these changes can be replicated only when a volume is attached.

If we first create a **container-1 with volume and if an image is created from container-1 and if we run the created image a container-2 gets created, we get all files from cont-1 to cont-2, but we don't get the data in the volume.**

i) Created a cont-1 along with volume.

```
root@ip-172-31-17-122:~# docker run -it --name cont-1 -v /my-vol ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
9c704ecd0c69: Pull complete
Digest: sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc436217b30
Status: Downloaded newer image for ubuntu:latest
```

ii) Go inside the container and you can see the volume (my-vol) has been created.

```
root@12f982b00cab:/# ll
total 60
drwxr-xr-x   1 root root 4096 Jul  6 13:58 ./
drwxr-xr-x   1 root root 4096 Jul  6 13:58 ../
-rwxr-xr-x   1 root root    0 Jul  6 13:58 .dockerenv*
lrwxrwxrwx   1 root root    7 Apr 22 13:08 bin -> usr/bin/
drwxr-xr-x   2 root root 4096 Apr 22 13:08 boot/
drwxr-xr-x   5 root root  360 Jul  6 13:58 dev/
drwxr-xr-x   1 root root 4096 Jul  6 13:58 etc/
drwxr-xr-x   3 root root 4096 Jun  5 02:06 home/
lrwxrwxrwx   1 root root    7 Apr 22 13:08 lib -> usr/lib/
lrwxrwxrwx   1 root root    9 Apr 22 13:08 lib64 -> usr/lib64/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 media/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 mnt/
drwxr-xr-x   2 root root 4096 Jul  6 13:58 my-vol/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 opt/
```

iii) Three files have been created inside the container i.e. container files.

```
root@12f982b00cab:/# touch files{1..3}
root@12f982b00cab:/# ll
total 60
drwxr-xr-x   1 root root 4096 Jul  6 13:59 ./
drwxr-xr-x   1 root root 4096 Jul  6 13:59 ../
-rwxr-xr-x   1 root root    0 Jul  6 13:58 .dockerenv*
lrwxrwxrwx   1 root root    7 Apr 22 13:08 bin -> usr/bin/
drwxr-xr-x   2 root root 4096 Apr 22 13:08 boot/
drwxr-xr-x   5 root root  360 Jul  6 13:58 dev/
drwxr-xr-x   1 root root 4096 Jul  6 13:58 etc/
-rw-r--r--   1 root root    0 Jul  6 13:59 files1
-rw-r--r--   1 root root    0 Jul  6 13:59 files2
-rw-r--r--   1 root root    0 Jul  6 13:59 files3
drwxr-xr-x   3 root root 4096 Jun  5 02:06 home/
lrwxrwxrwx   1 root root    7 Apr 22 13:08 lib -> usr/lib/
lrwxrwxrwx   1 root root    9 Apr 22 13:08 lib64 -> usr/lib64/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 media/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 mnt/
drwxr-xr-x   2 root root 4096 Jul  6 13:58 my-vol/
```

iv) Also a 5 files have been created inside the volume as well.

```
root@12f982b00cab:/# cd my-vol/
root@12f982b00cab:/my-vol# ll
total 8
drwxr-xr-x 2 root root 4096 Jul  6 13:58 ./
drwxr-xr-x 1 root root 4096 Jul  6 13:59 ../
root@12f982b00cab:/my-vol# touch vol-files{1..5}
root@12f982b00cab:/my-vol# ll
total 8
drwxr-xr-x 2 root root 4096 Jul  6 14:00 ./
drwxr-xr-x 1 root root 4096 Jul  6 13:59 ../
-rw-r--r-- 1 root root    0 Jul  6 14:00 vol-files1
-rw-r--r-- 1 root root    0 Jul  6 14:00 vol-files2
-rw-r--r-- 1 root root    0 Jul  6 14:00 vol-files3
-rw-r--r-- 1 root root    0 Jul  6 14:00 vol-files4
-rw-r--r-- 1 root root    0 Jul  6 14:00 vol-files5
```

v) An image is created with image:1 from container-1.

```
root@12f982b00cab:/# root@ip-172-31-17-122:~# docker commit cont1 image:1
Error response from daemon: No such container: cont1
root@ip-172-31-17-122:~# docker commit cont-1 image:1
sha256:ee1ce8a59daa7d1123667e5ca233ea97195a756244616a0bb945ac81a9d7466d
root@ip-172-31-17-122:~# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
12f982b00cab   ubuntu    "/bin/bash"   2 minutes ago   Up 2 minutes             cont-1
root@ip-172-31-17-122:~# docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
image        1         ee1ce8a59daa   11 seconds ago   78.1MB
ubuntu       latest    35a88802559d   4 weeks ago      78.1MB
```

vi) A container is created from image:1 and you got the container files into the cont-2.

```
root@ip-172-31-17-122:~# docker run -it --name cont-2 image:1
root@79075d5d1b49:/# ll
total 60
drwxr-xr-x   1 root root 4096 Jul  6 14:01 ./
drwxr-xr-x   1 root root 4096 Jul  6 14:01 ../
-rwxr-xr-x   1 root root    0 Jul  6 14:01 .dockerenv*
lrwxrwxrwx   1 root root    7 Apr 22 13:08 bin -> usr/bin/
drwxr-xr-x   2 root root 4096 Apr 22 13:08 boot/
drwxr-xr-x   5 root root  360 Jul  6 14:02 dev/
drwxr-xr-x   1 root root 4096 Jul  6 14:01 etc/
-rw-r--r--   1 root root    0 Jul  6 13:59 files1
-rw-r--r--   1 root root    0 Jul  6 13:59 files2
-rw-r--r--   1 root root    0 Jul  6 13:59 files3
drwxr-xr-x   3 root root 4096 Jun  5 02:06 home/
lrwxrwxrwx   1 root root    7 Apr 22 13:08 lib -> usr/lib/
lrwxrwxrwx   1 root root    9 Apr 22 13:08 lib64 -> usr/lib64/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 media/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 mnt/
drwxr-xr-x   2 root root 4096 Jul  6 13:58 my-vol/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 opt/
dr-xr-xr-x 187 root root    0 Jul  6 14:01 proc/
drwx------   2 root root 4096 Jun  5 02:05 root/
drwxr-xr-x   4 root root 4096 Jun  5 02:06 run/
lrwxrwxrwx   1 root root    8 Apr 22 13:08 sbin -> usr/sbin/
drwxr-xr-x   2 root root 4096 Jun  5 02:02 srv/
dr-xr-xr-x  13 root root    0 Jul  6 13:59 sys/
drwxrwxrwt   2 root root 4096 Jun  5 02:05 tmp/
drwxr-xr-x  12 root root 4096 Jun  5 02:02 usr/
drwxr-xr-x  11 root root 4096 Jun  5 02:05 var/
```

vii) However, you haven't got any files inside the volume. This is conclusion of the demonstration that if a container is created from the image of a container that has files inside the container and volume, so the newly container gets only the files inside the container and not the files inside the volume.

"VOLUME WILL NOT BE INCLUDED WHEN IMAGE IS UPDATED".

Here we've "updated the ubuntu image of cont-1 to image:1 of our choice from cont-1 to use the image to create another container".

```
root@79075d5d1b49:/# cd my-vol/
root@79075d5d1b49:/my-vol# ll
total 8
drwxr-xr-x 2 root root 4096 Jul  6 13:58 ./
drwxr-xr-x 1 root root 4096 Jul  6 14:01 ../
```

SHARING A VOLUME:

Let's say we've a volume attached to cont-1 and if we create an image from cont-1 and run to create cont-2, then we get the files inside the container and we don't get the files inside the volume.

To get those, that volume needs to be shared to another new creating container so we can get all the files directly, to do that we need to use a command.

docker run -itd --name cont-2 --privileged=true --volumes-from cont-1 ubuntu

i) Created a container with volume.

```
[root@ip-172-31-84-187 ~]# docker volume ls
DRIVER    VOLUME NAME
[root@ip-172-31-84-187 ~]# docker run -itd --name cont1 -v /MY-VOLUME ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
9c704ecd0c69: Already exists
Digest: sha256:2e863c44b718727c860746568e1d54afd13b2fa71b160f5cd9058fc436217b30
Status: Downloaded newer image for ubuntu:latest
a082b15884bdaed267630506c57843d71ce1178da7f8e7c8af5034ab325e6e3d
[root@ip-172-31-84-187 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
a082b15884bd   ubuntu    "/bin/bash"   5 seconds ago   Up 4 seconds             cont1
[root@ip-172-31-84-187 ~]# docker volume ls
DRIVER    VOLUME NAME
local     3fbbe8a18080a82ec8d5c37dbf93b91677a52dd25fe07189e273ec52a843ba3f
```

ii) Go inside the container and you can find the created volume(MY-VOLUME) and there are no files.

```
[root@ip-172-31-84-187 ~]# docker attach cont1
root@a082b15884bd:/# ll
total 0
drwxr-xr-x   1 root root  23 Jul  6 10:50 ./
drwxr-xr-x   1 root root  23 Jul  6 10:50 ../
-rwxr-xr-x   1 root root   0 Jul  6 10:50 .dockerenv*
drwxr-xr-x   2 root root   6 Jul  6 10:50 MY-VOLUME/
lrwxrwxrwx   1 root root   7 Apr 22 13:08 bin -> usr/bin/
drwxr-xr-x   2 root root   6 Apr 22 13:08 boot/
drwxr-xr-x   5 root root 360 Jul  6 10:50 dev/
drwxr-xr-x   1 root root  66 Jul  6 10:50 etc/
drwxr-xr-x   3 root root  20 Jun  5 02:06 home/
lrwxrwxrwx   1 root root   7 Apr 22 13:08 lib -> usr/lib/
lrwxrwxrwx   1 root root   9 Apr 22 13:08 lib64 -> usr/lib64/
drwxr-xr-x   2 root root   6 Jun  5 02:02 media/
drwxr-xr-x   2 root root   6 Jun  5 02:02 mnt/
drwxr-xr-x   2 root root   6 Jun  5 02:02 opt/
dr-xr-xr-x 164 root root   0 Jul  6 10:50 proc/
drwx------   2 root root  37 Jun  5 02:05 root/
drwxr-xr-x   4 root root  33 Jun  5 02:06 run/
lrwxrwxrwx   1 root root   8 Apr 22 13:08 sbin -> usr/sbin/
drwxr-xr-x   2 root root   6 Jun  5 02:02 srv/
dr-xr-xr-x  13 root root   0 Jul  6 10:50 sys/
drwxrwxrwt   2 root root   6 Jun  5 02:05 tmp/
drwxr-xr-x  12 root root 133 Jun  5 02:02 usr/
drwxr-xr-x  11 root root 139 Jun  5 02:05 var/
root@a082b15884bd:/# cd MY-VOLUME/
root@a082b15884bd:/MY-VOLUME# ll
total 0
```

iii) Now a few files are created and came out of the container with ctrl PQ.

```
root@a082b15884bd:/MY-VOLUME# touch files{1..3}
root@a082b15884bd:/MY-VOLUME# ll
total 0
drwxr-xr-x 2 root root 48 Jul  6 10:51 ./
drwxr-xr-x 1 root root 23 Jul  6 10:50 ../
-rw-r--r-- 1 root root  0 Jul  6 10:51 files1
-rw-r--r-- 1 root root  0 Jul  6 10:51 files2
-rw-r--r-- 1 root root  0 Jul  6 10:51 files3
```

iv) Created another container-2 with sharing the volume of cont-1.

```
[root@ip-172-31-84-187 ~]# docker run -itd --name cont2 --privileged=true --volumes-from cont1 ubuntu
6682d83306e301dfd7b7d968620bbf94a4c78d1ac0676ed79ed7fc921707048a
[root@ip-172-31-84-187 ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
6682d83306e3   ubuntu    "/bin/bash"   4 seconds ago    Up 3 seconds              cont2
a082b15884bd   ubuntu    "/bin/bash"   2 minutes ago    Up 2 minutes              cont1
```

v) Here you can find the files that were in cont-1 are reflecting in cont-2 as well. And also created 5 more files with name new.

```
[root@ip-172-31-84-187 ~]# docker attach cont2
root@6682d83306e3:/# ll
total 0
drwxr-xr-x   1 root root   23 Jul  6 10:52 ./
drwxr-xr-x   1 root root   23 Jul  6 10:52 ../
-rwxr-xr-x   1 root root    0 Jul  6 10:52 .dockerenv*
drwxr-xr-x   2 root root   48 Jul  6 10:51 MY-VOLUME/
lrwxrwxrwx   1 root root    7 Apr 22 13:08 bin -> usr/bin/
drwxr-xr-x   2 root root    6 Apr 22 13:08 boot/
drwxr-xr-x  11 root root 2720 Jul  6 10:52 dev/
drwxr-xr-x   1 root root   66 Jul  6 10:52 etc/
drwxr-xr-x   3 root root   20 Jun  5 02:06 home/
lrwxrwxrwx   1 root root    7 Apr 22 13:08 lib -> usr/lib/
lrwxrwxrwx   1 root root    9 Apr 22 13:08 lib64 -> usr/lib64/
drwxr-xr-x   2 root root    6 Jun  5 02:02 media/
drwxr-xr-x   2 root root    6 Jun  5 02:02 mnt/
drwxr-xr-x   2 root root    6 Jun  5 02:02 opt/
dr-xr-xr-x 162 root root    0 Jul  6 10:52 proc/
drwx------   2 root root   37 Jun  5 02:05 root/
drwxr-xr-x   4 root root   33 Jun  5 02:06 run/
lrwxrwxrwx   1 root root    8 Apr 22 13:08 sbin -> usr/sbin/
drwxr-xr-x   2 root root    6 Jun  5 02:02 srv/
dr-xr-xr-x  13 root root    0 Jul  6 10:50 sys/
drwxrwxrwt   2 root root    6 Jun  5 02:05 tmp/
drwxr-xr-x  12 root root  133 Jun  5 02:02 usr/
drwxr-xr-x  11 root root  139 Jun  5 02:05 var/
root@6682d83306e3:/# cd MY-VOLUME/
root@6682d83306e3:/MY-VOLUME# ll
total 0
drwxr-xr-x 2 root root 48 Jul  6 10:51 ./
drwxr-xr-x 1 root root 23 Jul  6 10:52 ../
-rw-r--r-- 1 root root  0 Jul  6 10:51 files1
-rw-r--r-- 1 root root  0 Jul  6 10:51 files2
-rw-r--r-- 1 root root  0 Jul  6 10:51 files3
root@6682d83306e3:/MY-VOLUME# touch new{1..5}
```

vi) Now go inside the container-1 and you can find the 5 new files that are created in cont-2 will be there in cont-1 as well. That's how "**the data replication is possible with the docker volume concept**".

```
[root@ip-172-31-84-187 ~]# docker attach cont1
root@a082b15884bd:/MY-VOLUME# ll
total 0
drwxr-xr-x 2 root root 108 Jul  6 10:52 ./
drwxr-xr-x 1 root root  23 Jul  6 10:50 ../
-rw-r--r-- 1 root root   0 Jul  6 10:51 files1
-rw-r--r-- 1 root root   0 Jul  6 10:51 files2
-rw-r--r-- 1 root root   0 Jul  6 10:51 files3
-rw-r--r-- 1 root root   0 Jul  6 10:52 new1
-rw-r--r-- 1 root root   0 Jul  6 10:52 new2
-rw-r--r-- 1 root root   0 Jul  6 10:52 new3
-rw-r--r-- 1 root root   0 Jul  6 10:52 new4
-rw-r--r-- 1 root root   0 Jul  6 10:52 new5
```

DATA REPLICATION FROM HOST TO CONTAINER:

If you've your source code in your local machine in a folder and if you want to use the data with a container to deploy the code. Then you need to mention the path where you've the source code in the source section and in the destination section mention the path with a volume-name that gets created inside the container. So that the the data in the local is shared to a container with the help of volume for deployment.

docker run -itd --name cont-name -v path of the folder:/vol-name --privileged=true image-name

14) What is Docker Swarm and how it's useful?

A: Docker swarm is an orchestration service which is an alternate for Kubernetes though it doesn't have the advantages like K8s does but it is also used in companies. It is used to manage and handle multiple containers on multiple servers at the same time. All it does is, it implements a cluster where it has a Swarm manager, workers, in it and the manager assigns the work to worker nodes to do. Let's say we take the source code from the developers through github we clone it if needed, we write a dockerfile, push the image to the dockerhub and we create a service in docker swarm it creates a container and the application runs in it. In case if the container gets deleted, the "replication mode" feature in docker swarm creates container in either worker nodes or manager nodes based on the current count of running containers and also you can scale up or scale down and this mode works as a Auto Scaling group as well. This is the main advantage of Docker swarm.

You can implement with installing docker on manager and nodes and then initialize it using the below command:

docker swarm init --advertise-addr private IP of manager

15) What is Docker compose and how it is useful?

A: Docker compose is used to build, run and handle multiple containers in a single server at the same time with the help of a YAML file. We use compose yaml file to pass the parameters where it automates the creation of containers, volumes, networks etc. We can just execute the compose file to create them and you'll use the command docker-compose up -d and if you want to down you can use docker-compose down -d, it will down the service and it will also delete the running containers and which eventually stops the availability of application.