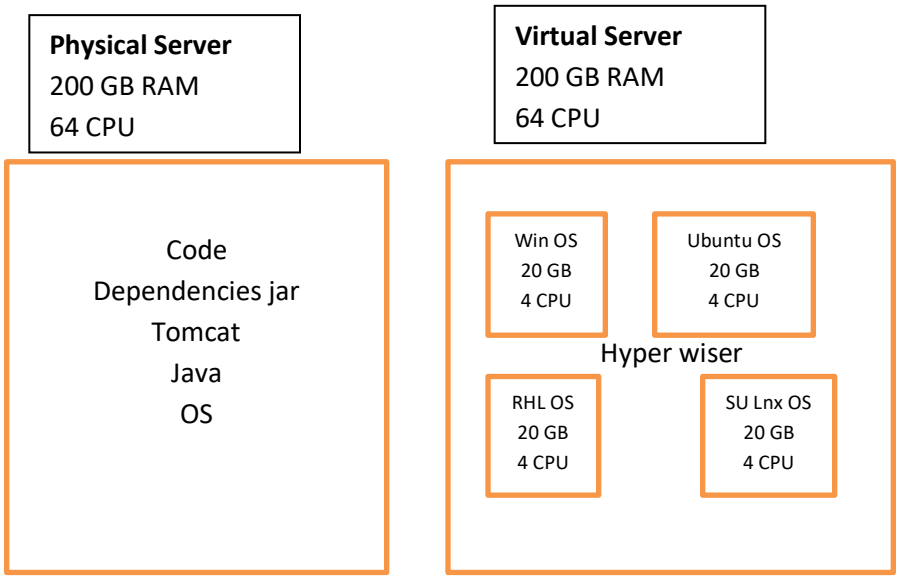
1) What is micro service architecture?

Application which is running independently, having it's own DB & application server.
With this architecture workload management, highly availability can be easily achieved.
One module is not having impact on other module.

2) What was old day's implementation of application?

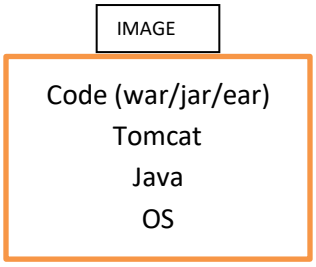


Old days above 2 implantation what there

- **Physical Server :**
  - To run application on these server we need to have OS, Java, Application/web server, dependency jar, code
  - These will unnecessary uses extra resources even if less load.
  - Multiple applications need to be deployed on single system, due to which one application deployment was getting impact on other application.

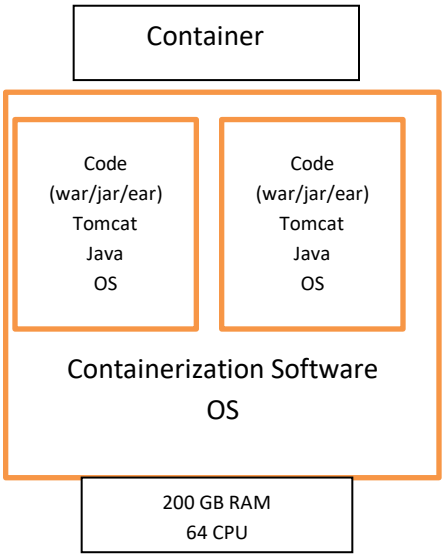
- **Virtual Server :**
  - To overcome above disadvantages virtual server concept came into picture.
  - We can split large machine into multiple machine in which we can install different-different OS
  - This can be achieve by using hyper wiser software
  - System resource can be best utilizes
  - We can run multiple applications on same machine.
  - **Disadvantage of this are** Boot time, dynamic utilization of resource, OS maintenance, scalability, application working on one platform/environment is not running on other platform/environment
  - Application run in DEV environment is not running on UAT/PROD environment.
  - Maintain application constancy in all environment(DEV/UAT/PROD)

3) What is image?



- To oven come disadvantage of virtual server, image concept come into picture
- Image contains all application which is needed to run our application. So we can run application using image in any environment i.e. DEV/UAT/PROD
- Image is our application with all dependencies.
- If we run the image it will become container.

4) What is container?



- It is runtime format of image
- End user can access application from container
- Container is best platform to run micro services code.
- It has lightweight OS & all lightweight tomcat & lightweight code, so it start very quickly which will resolves boot start issue of virtual machine.
- It take dynamic resource from host machine
- It has no OS maintenance, because it have only lightweight OS
- As we have image, we can run any number of container, so scalability can be achieve.

5) How to run containers?

- On physical server we need to have complete OS
- On top of it we need to install containerization software
- We have many containerization software i.e. LXC,LXD, Container D, Podman, Crio, Docker
- Containers are way to host an application in multiple environments such as DEV, QA, IAT, PROD etc.
- Container is simply a software package that has an application & all of its dependencies inside to run it
- Containers are a streamline way to build, test, deploy, & redeploy application on multiple environments from developer's local laptop to an on-premises data center and even to the cloud.

6) Why containers?

- Greater efficacy: container allows application to be more rapidly deployed, patched or scaled.
- Less overhead: containers require less system resources than physical or virtual machine environments because they don't include full OS
- Increase portability: application running in container can be deployed easily to multiple different OS and hardware platform.
- More consistent operation: DevOps team knows application in containers will run the same, regardless of where are they deployed.
- Better application development: containers support agile & devops effort to accelerate development, test & production cycles.

7) What is Docker?

- Docker is software or containerization platform that runs on Linux & windows. It helps to build ship & run container anywhere.
- Docker is OS level Virtualization software that run and deploys containers. In VMWare terms, you can think of it as being similar ESXI.
- Docker provides a way to run application securely isolated in a container, package with all its dependencies & libraries.
- Docket Inc (Mirantis now). It is a company based on San Francisco and is overall maintainer of the open-source project. Docker INC. also has offers commercial version of Docker.

8) What is container orchestration?

- Managing multiple containers & managing them is call as container orchestration.
- If we want to run/manage 1K containers, increase or decrease multiple containers, one container talk to another container, place container in different machine, want high availability, want workload management , scalability, it is called as container orchestration.
- We have Kubernates, Rnncher, Docker Swarm, Nomad, Mesos, openshift , Vmware Tanzu as container orchestration tool for on-premises application.

- We nave EKS (Amazon), AKS (Azure), GKE (Google) as container orchestration tool for cloud application.

8) How to install Docker?

- We can go to https://docs.docker.com/engine/install/ubuntu/ steps given to install Docker in Ubuntu
- We can use https://github.com/lerndevops/docker/tree/master/01-install

9) How to check Docker is running in system?

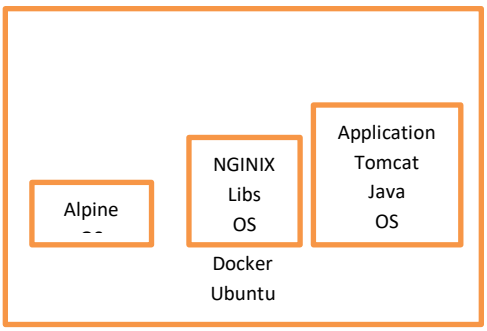
Below are the commands to check Docker running

- Check Docker version : > `docker –v`
- Check Docker location : > `cd /var/lib/docker/`
- Check Docker service status: >`service docker status`

9) What are image commands?

- Get image from docker hub:> `docker pull <image name>`
  Ex :> `docker pull alpine`
- Get specific version image from docker hub:> `docker pull <image name>:<version>`
  Ex :> `docker pull alpine:3.12`
- Check images pulled :> `docker images`
- To remove all dangling images these are images without name/tag
  - Syntax: `docker image prune`
- To remove all images for those we have not run container
  - Syntax: `docker image prune –a`
- To delete images
  - Syntax: `docker image rm <image name>:<version>`
  - Ex: `docker image rm nginx:1.3`
- To inspect images
  - Syntax: `docker inspect <image name>:<version>`
  - Ex: `docker inspect nginx:1.3`
- To get image history/to check layers available in image
  - Syntax: `docker history <image name>:<version>`
  - Ex: `docker history nginx:1.3`

10) What all types of Images are there?



- Image can be simple OS i.e. alpine
- Image can be software i.e. web server, app server, DB.
- Image can be application with its dependencies i.e. any web application.

11) How to run image to become container? What are container commands?

- Run container command
  - syntax:> `docker run –d –P <image name>`
  - Ex :> `docker run -d -P nginx`
- If image is not available in local we can download & run container using below command
  - syntax:> `docker run <image name>`
  - Ex :> `docker run httpd`
- See running container command
  - `Syntax:> docker ps`
- See all container command
  - `Syntax:> docker ps -a`
- Command to get inside container
  - Syntax:>`docker exec -it <containerId> bash`
  - Ex :> `docker exec -it ccff5a6e43ed bash`
  - **–it :** this means interactive terminal
  - `exec` **:** this means to execute using this we can do execute any command inside container
- Command to get inside container to do list
  - Syntax:>`docker exec <containerId> ls /tmp`
- Command to get log on container
  - Syntax:> `docker logs -f <containerId>`
  - Ex: > `docker logs -f db6e71b6b362`
- Command to get outside container
  - Syntax:>`exit`
  - `We can use button CTRL+P+Q`
- Command to check Utilization statistics of container
  - Syntax: > `docker stats --no-stream`
  - Syntax **:** > `free –h`
- Command to stop container
  - Syntax:> `docker stop <container Id1> <container Id2>`
  - Ex: > `docker stop 34a68ce09cca`
- Command to kill container
  - Syntax:> `docker kill <container Id1> <container Id2>`
  - Ex: > `docker kill 34a68ce09cca`
- Difference between is kill & stop is, kill is forceful stop container if container is getting in hang state, where as stop is graceful stopping container if any configuration change happens

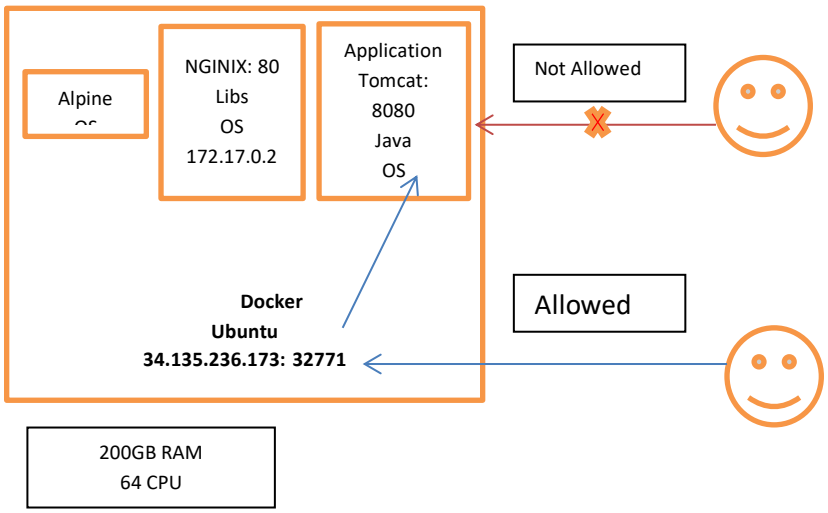
- Command to start container
  - Syntax:> `docker start <container Id1> <container Id2>`
  - Ex: > `docker start 34a68ce09cca`
- Command to re-start container
  - Syntax:> `docker restart <container Id1> <container Id2>`
  - Ex: > `docker restart 34a68ce09cca`
- Command to delete/remove container
  - Syntax: > `docker rm <container Id>`
  - Syntax: > `docker rm <container Id1> <container Id2>`
  - Ex: > docker rm ad957537c159
- Command to delete/remove container without stop
  - Syntax: > `docker rm -f <container Id>`
  - Syntax: > `docker rm -f <container Id1> <container Id2>`
  - Ex: > `docker rm -f ad957537c159`
- Command to delete only all stopped container
  - Syntax: > `docker container prune`
- Command to get list of all container IDs, this is used in case we want to perform some task on all containers
  - Syntax: > `docker ps -aq`
- Command to delete all containers running by get list of all container IDs
  - Syntax: > `docker rm -f` `` `docker ps -aq` ``
- Command to inspect image
  - Syntax:> `docker inspect <image name>`
  - Ex: >`docker inspect leaddevops/petclinic`
- Command to copy file/folder from container
  - Syntax:> `docker cp <container Id>:<File Location> <destination location>`
  - Ex:> `docker cp db6e71b6b362:/usr/share/nginx/html/index.html /tmp/`
- Command to copy file/folder in container
  - Syntax :> `docker cp <source file> <ContainerId>:<destination location>`
  - Ex:> `docker cp 1.txt db6e71b6b362:/usr/share/nginx/html/`
- We can scale application by just running `docker run` command as many time as we needed containers
- Here **docker run** is client command
- If we give this command it run in background & it goes to docker service
- All container data will be available on > `cd /var/lib/docker`

12) How many ways to run container?

There are 3 ways to run containers:

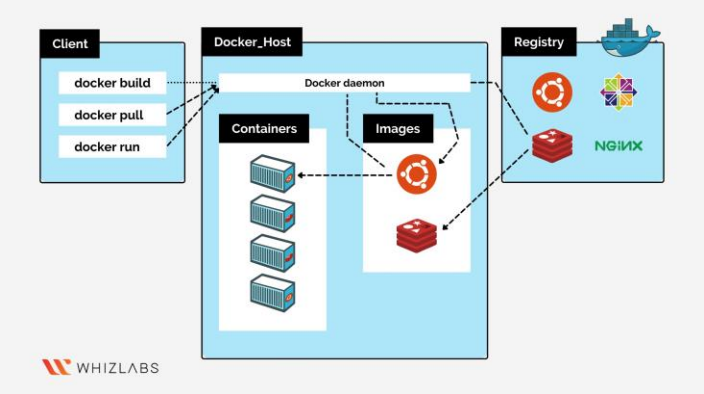
Foreground/Background/Interactive

- **Foreground :**
  - application which doesn't run in background i.e. YouTube
  - We can use below command to run container in foreground mode
  - syntax:> `docker run <image name>`
  - Ex :> `docker run httpd`
  - Above command will download image & run container in fore ground mode
- **Background/Detached/Daemon :**
  - application which run in background i.e. Music Player
  - We can use below command to run container in background mode
  - syntax:> `docker run –d <image name>`
  - where –d denote detached mode
  - Ex :> `docker run –d httpd`
  - **CONTAINER ID:** Whenever we rune any container a unique reference Id will be created which is call container Id, it is use to access container
  - **NAMES** : Whenever we run container a name is assigned to container by docker, we can change by using below command
  - Syntax> `docker run –d -- name <any name> <image>`
  - Ex: > `docker run –d -- name santosh nginix`
  - We can rename container using below command
  - Syntax:> `docker rename <container Id> <new name>`
  - Ex: >`docker rename a7eff53ad550 neetu`
  - **COMMAND:** Images are programed in such way, when we start container it will automatically start application, this can be achieved by using COMMAD field.
  - We can use below command to inspect image
  - Syntax:> `docker inspect <image name>`
  - Ex: >`docker inspect leaddevops/petclinic`
  - **PORTS:** we can access any application using PORT, depending upon technology, different application run on different port.
  - Every container have its own IP address & PORT number
  - Container IP &PORT cannot be access by outside



- If we want to access Container IP &PORT from outside we need to access through Host machine using below command, here after restart every time port get change
- Syntax:> `docker run -d -P <image>`
- Ex :> `docker run -d -P leaddevops/petclinic`
- Here **–P** is the parameter used to publish some port of host machine to access container
- Command to customized port number to give our own port, here after restart port will not change:
- Syntax:> `docker run -d -p 1234:8080 <image>`
- Ex :> `docker run -d -p 1234:8080 leaddevops/petclinic`
- Command to customized port number within range of port if we don't know whether port is free or not:
- Systax:> `docker run -d -p 1234-1236:8080 <image>`
- Ex :> `docker run -d -p 1234-1236:8080 leaddevops/petclinic`

12) Docker Architecture?



13) How to build image? https://github.com/lerndevops/docker/blob/master/04-images/readme.txt

There are 2 ways to create

1) MANAUL
2) AUTOMATED

14) How to build image using Manual Process?

- Install base os first using below command
  - docker pull Ubuntu
- run it as container in interactive mode
  - docker run -it ubuntu bash
- make necessary changes inside the container & create new HTML page
  - apt-get update
  - apt-get install -y nginx
  - apt-get install -y vim

  - vi /var/www/html/index.html (edit & save file)
  - ctrl pq (to comeout of contianer safely)
- Commit new image use below command
  - Syntax:>docker commit -m "<comment>" <containerId> <image name>:<varsion>
  - Ex:> docker commit -m "santosh nginx" 18925d64842d sannginx:v1
- Check all image use below command
  - docker images
- Run new image using below command
  - docker run -d -P sannginx:v1
- Check it is running or not using below command, after executing command, we can see that it is not running because we have not given nginx startup command for image creation. If we will not pass any startup command, it will by default takes whatever is available from OS/ base image
  - docker ps –a

```
root@docker:/tmp# docker ps -a
CONTAINER ID   IMAGE         COMMAND   CREATED          STATUS                      PORTS     NAMES
8818aadbd11f   sannginx:v1   "bash"    19 seconds ago   Exited (0) 17 seconds ago             elastic_kepler
18925d64842d   ubuntu        "bash"    25 minutes ago   Up 25 minutes                         fervent_solomon
root@docker:/tmp#
```

- Commit new image with startup command & expose port use below command
  - Syntax:> docker commit -m "<comment>" -c '<startup command>' -c 'EXPOSE 80' <base container Id> <image name>:<version>
  - Ex:> docker commit -m "santosh nginx2" -c 'CMD /usr/sbin/nginx -g "daemon off;"' -c 'EXPOSE 80' 18925d64842d sannginx:v3
- We can check by inspecting image for startup command
  - docker inspect sannginx:v2

```
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": true,
            "OpenStdin": true,
            "StdinOnce": true,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "usr/sbin/nginx -g \"deamon off;\""
            ],
            "Image": "ubuntu",
```

- run image & access from browser
  - docker run -d -P sannginx:v3

```
root@docker:/tmp# docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED              STATUS                      PORTS                                   NAMES
6fe733af2f20   sannginx:v3   "/bin/sh -c '/usr/sb…"   21 minutes ago       Up 21 minutes               0.0.0.0:32774->80/tcp, :::32774->80/tcp admiring_kare
92241653486a   sannginx:v2   "/bin/sh -c 'usr/sbi…"   26 minutes ago       Exited (1) 26 minutes ago                                           objective_ardinghelli
8818aadbd11f   sannginx:v1   "bash"                   54 minutes ago       Exited (0) 54 minutes ago                                           elastic_kepler
18925d64842d   ubuntu        "bash"                   About an hour ago    Up About an hour                                                    fervent_solomon
root@docker:/tmp#
```

15) How to build image using Docker File/Automated process (real time practice)?

- create a simple text file & write the all the instructions build an image

```
vi Dockerfile
FROM ubuntu
RUN apt-get udpate
RUN apt-get install -y nginx vim
RUN mkdir /home/config/
RUN touch /home/config/db.props
EXPOSE 80
CMD /usr/sbin/nginx -g "daemon off;"
save&quit
```

- build an image using below command
  - Syntax :>docker build –t <image name>:<varsion> .
  - Ex: >docker build –t autonginx:v1 .
  - Where . is current docker context so we have not mansion docker file path, because docker will use default file naming convention as "**Dockefile"** & it look in current folder.
- Check image & run image using below command
  - docker run -d -P autonginx:v1

```
root@docker:/# docker run -d -P autonginx:v1
a4df861c757e47f7b85f787680c72111fcc3f2194d44590ca9bfa705758fe292
root@docker:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED             STATUS             PORTS                                                NAMES
a4df861c757e   autonginx:v1   "/bin/sh -c '/usr/sb…"   5 seconds ago       Up 4 seconds       0.0.0.0:32775->80/tcp, :::32775->80/tcp              cranky_hofstadter
6fe733af2f20   sannginx:v3    "/bin/sh -c '/usr/sb…"   About an hour ago   Up About an hour   0.0.0.0:32774->80/tcp, :::32774->80/tcp              admiring_kare
18925d64842d   ubuntu         "bash"                   2 hours ago         Up 2 hours                                                              fervent_solomon
root@docker:/#
```

16) How to build image using existing image & which take content from host machine?

- Create our input file > `vi /tmp/index.html`
- Create our docker file with any name > `vi file1`

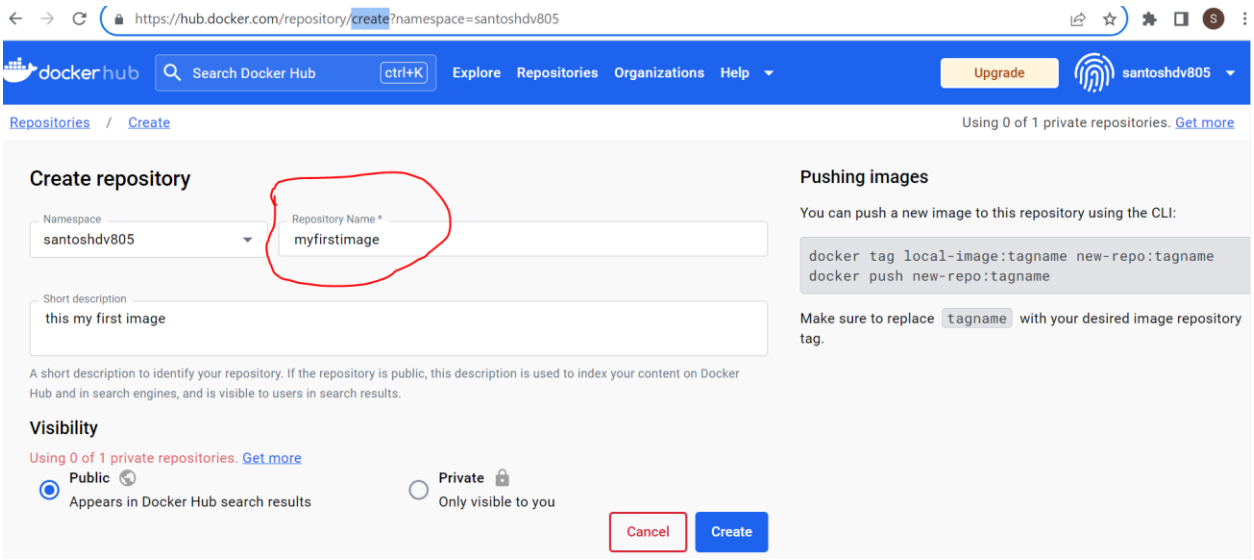
```
FROM autonginx:v1
RUN rm /var/www/html/ind*
COPY index.html /var/www/html/
```

- Build image using below command
  - syntax:> docker build -t <image name>:<version> -f <docker file> <input folder context path>
  - ex:> docker build -t autonginx:v2 -f file1 /tmp
- Check image & run image
  - docker run -d -P autonginx:v2

```
root@docker:/# docker run -d -P autonginx:v2
fe28468953572539f9bc24ed75ebec25a16b913c289830543e54fe525898ee71
root@docker:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS                                                NAMES
fe2846895357   autonginx:v2   "/bin/sh -c '/usr/sb…"   13 seconds ago   Up 12 seconds   0.0.0.0:32777->80/tcp, :::32777->80/tcp              pensive_pasteur
ead1ada8cc4b   autonginx:v2   "/bin/sh -c '/usr/sb…"   4 minutes ago    Up 4 minutes    0.0.0.0:32776->80/tcp, :::32776->80/tcp              objective_keller
a4df861c757e   autonginx:v1   "/bin/sh -c '/usr/sb…"   39 minutes ago   Up 39 minutes   0.0.0.0:32775->80/tcp, :::32775->80/tcp              cranky_hofstadter
6fe733af2f20   sannginx:v3    "/bin/sh -c '/usr/sb…"   2 hours ago      Up 2 hours      0.0.0.0:32774->80/tcp, :::32774->80/tcp              admiring_kare
18925d64842d   ubuntu         "bash"                   3 hours ago      Up 3 hours                                                           fervent_solomon
```

17) How to push/store image in registries?

- 1st registry for pushing/storing image is docker hub https://hub.docker.com/ where we can register & create our own account to store image.
- Login in docker hub, create new repository by going in repositories --> create
- Give repository name “myfirstimage”



- To push image do login in docker hub
  - >docker login
  - Given user name & password

```
root@docker:/# docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't h
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grant
 at https://docs.docker.com/go/access-tokens/

Username: santoshdv805
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

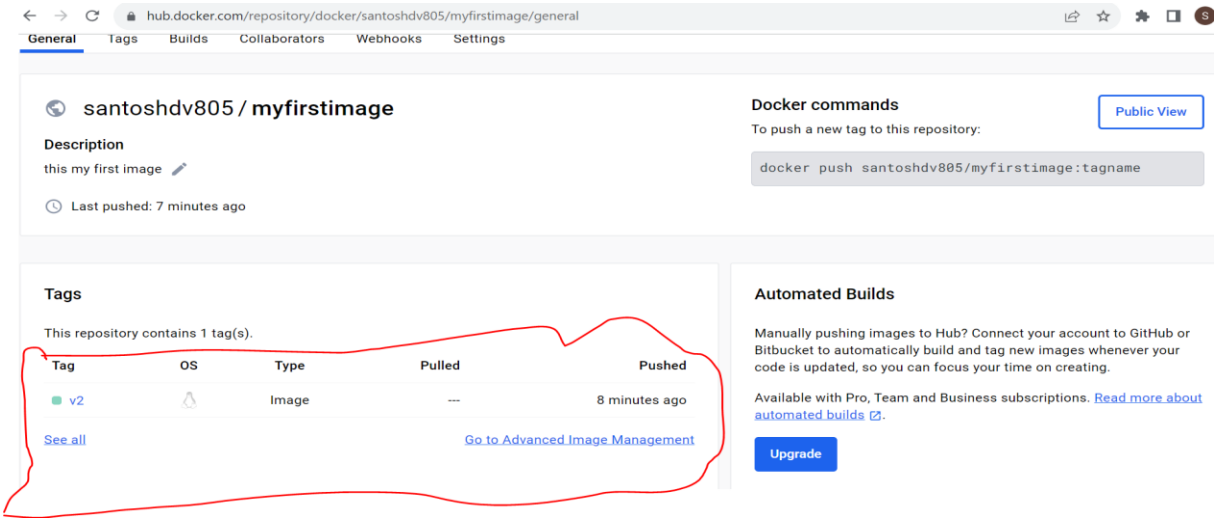
Login Succeeded
root@docker:/#
```

- We need to give name to image in such way, that image name must have account Id as well, like <docker hub id>/<repository name>
  - Here we created copy of existing image with new name as per speciation using below command
  - syntax> `docker tag <old image>:<version> <docker hub id>/<repository name>:<version>`
  - ex> `docker tag autonginx:v2 santoshdv805/myfirstimage:v2`
- push image in docker hub using below command

- Syntax> `docker push <new image name having accounId and repo name>:<version>`
- Ex >`docker push santoshdv805/myfirstimage:v2`

```
root@docker:/# docker tag autonginx:v2 santoshdv805/myfirstimage:v2
root@docker:/# docker images
REPOSITORY                   TAG       IMAGE ID       CREATED         SIZE
santoshdv805/myfirstimage    v2        57b063184dd8   38 hours ago    275MB
autonginx                    v2        57b063184dd8   38 hours ago    275MB
santoshdv805                 latest    57b063184dd8   38 hours ago    275MB
autonginx                    v1        64b0a9915eef   38 hours ago    275MB
sannginx                     v3        6d3c6cea2506   39 hours ago    238MB
sannginx                     v2        06fec3cd1adb   40 hours ago    238MB
sannginx                     v1        a87118cec22a   40 hours ago    238MB
httpd                        latest    a6ca7b52a415   7 days ago      168MB
redis                        latest    961dda256baa   2 weeks ago     138MB
nginx                        latest    593aee2afb64   5 weeks ago     187MB
ubuntu                       latest    e4c58958181a   7 weeks ago     77.8MB
alpine                       latest    8ca4688f4f35   2 months ago    7.34MB
alpine                       3.16      187eae39ad94   3 months ago    5.54MB
leaddevops/petclinic         latest    a23d67c711e5   2 years ago     594MB
root@docker:/# docker push santoshdv805/myfirstimage:v2
The push refers to repository [docker.io/santoshdv805/myfirstimage]
3d58c7038957: Pushed
6e086ebb2bb6: Pushed
182f6f57e0bf: Pushed
eed1dd2e5995: Pushed
cdebc6f2faa3: Pushed
dae5d7afb29d: Pushed
1251204ef8fc: Mounted from library/ubuntu
47ef83afae74: Mounted from library/ubuntu
df54c846128d: Mounted from library/ubuntu
be96a3f634de: Mounted from library/ubuntu
v2: digest: sha256:aaa90610aea7a01a47c17793553d584477686486a979f9a9500c859869612eec size: 2402
root@docker:/#
```

- Go to docker hub account & check



- To store image of private organization we can use DTR(docker trusted registries), artifcatory, nexus

18) How to save image & restore/load image from backup?

- We can use below command to store image

- Syntax> `docker save <image>:<version> -o <destination file>`
- Ex> `docker save autonginx:v2 -o santoshbackup`

```
root@docker:/# docker save autonginx:v2 -o santoshbackup
root@docker:/# ls -l
total 275332
-rw-r--r--   1 root root       209 Nov 27 11:40 Dockerfile
lrwxrwxrwx   1 root root         7 Sep 18 21:40 bin -> usr/bin
drwxr-xr-x   4 root root      4096 Oct 31 06:13 boot
drwxr-xr-x  16 root root      3820 Nov 29 01:48 dev
drwxr-xr-x  97 root root      4096 Nov 29 01:48 etc
-rw-r--r--   1 root root        76 Nov 27 12:15 file1
drwxr-xr-x   6 root root      4096 Oct 31 07:52 home
lrwxrwxrwx   1 root root         7 Sep 18 21:40 lib -> usr/lib
lrwxrwxrwx   1 root root         9 Sep 18 21:40 lib32 -> usr/lib32
lrwxrwxrwx   1 root root         9 Sep 18 21:40 lib64 -> usr/lib64
lrwxrwxrwx   1 root root        10 Sep 18 21:40 libx32 -> usr/libx32
drwx------   2 root root     16384 Sep 18 21:43 lost+found
drwxr-xr-x   2 root root      4096 Sep 18 21:40 media
drwxr-xr-x   2 root root      4096 Sep 18 21:40 mnt
drwxr-xr-x   3 root root      4096 Oct 31 13:06 opt
dr-xr-xr-x 167 root root         0 Nov 29 01:48 proc
drwx------   5 root root      4096 Nov 27 12:16 root
drwxr-xr-x  27 root root       940 Nov 29 01:50 run
-rw-------   1 root root 281862144 Nov 29 02:38 santoshbackup
lrwxrwxrwx   1 root root         8 Sep 18 21:40 sbin -> usr/sbin
drwxr-xr-x   7 root root      4096 Sep 18 21:45 snap
drwxr-xr-x   2 root root      4096 Sep 18 21:40 srv
dr-xr-xr-x  13 root root         0 Nov 29 01:48 sys
drwxrwxrwt  11 root root      4096 Nov 29 01:50 tmp
drwxr-xr-x  14 root root      4096 Sep 18 21:41 usr
drwxr-xr-x  13 root root      4096 Sep 18 21:42 var
```

- We can use below command to store image
  - Syntax: > `docker load < <backup file name>`
  - Ex> `docker load < santoshbackup`

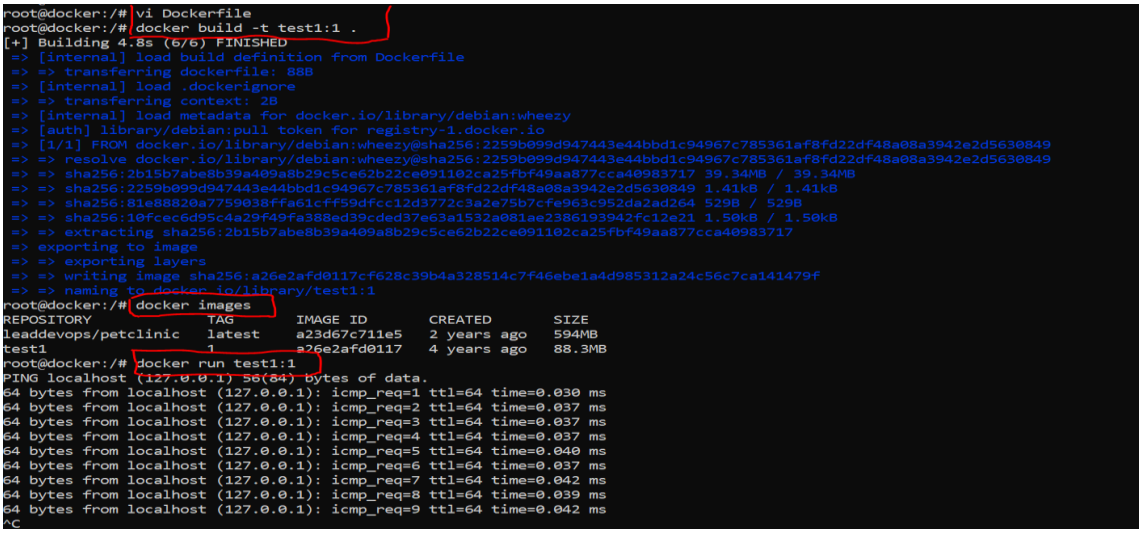
19) What are docker file directives?

- A Dockerfile is set of instructions which are used to construct a Docker Image.
- These instructions are called directives.
- Below are Dockerfiel directive
  - **FROM**: start a new build stage and sets the base image. Usually must be the first directive in the Dockerfile (except ARG can be placed before FROM).
  - **ENV**: set environment variables. These can be referenced in the Dockerfile itself and are visible to the container at runtime.
  - **RUN**: creates a new layer on top of the previous layer by running a command inside that new layer and committing the changes.
  - **CMD**: specify a default command used to run a container at execution time.
  - **EXPOSE**: documents which port(s) are intended to published when running a container
  - **WORKDIR**: sets the current working directory for subsequent directives such as ADD, COPY, CMD, ENTRYPOINT, etc. Can be used multiple times to change the directories through the Dockerfile. You can also use relative path, which sets the new working directory relative to the previous working directory.
  - **COPY**: copy files from the local machine to the image.
  - **ADD**: similar to COPY, but can also pull files using a URL and extract an archive into loose files in the image.
  - **STOPSIGNAL**: specify the signal that will be used to stop the container.

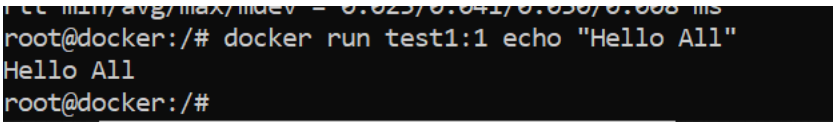
- **HEALTHCHECK**: specify a command to run in order to perform a custom health check to verify that the container is working properly.

20) What difference between CMD & ENTRYPOINT?

- **CMD**: specify a default command used to run a container at execution time. Check below ex.
- Create Docker file
  - >vi Dockerfile
    ```
    FROM debian:wheezy
    CMD ["/bin/ping", "localhost"]
    ```
- Build image from Docker file
  - >docker build -t test1:1 .
- Run image, which will give output as continuous ping to localhost

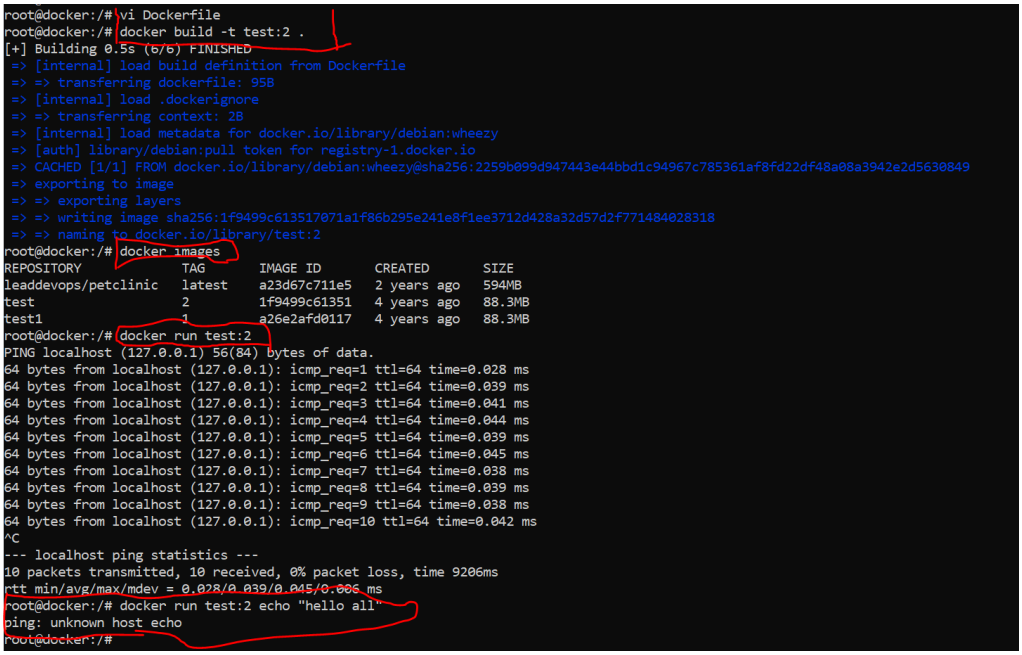


- But if we override run command as below, it will do whatever we have overridden
  - > docker run test1:1 echo "Hello All"



- So **CMD** field can be overridden
- **ENTRYPOINT**: instruction is used to provide executable that will always execute when the container is launched. It also specifies a default command used to run a container at execution time. Check below ex
- Create Docker file
  - >vi Dockerfile
    ```
    FROM debian:wheezy
    ENTRYPOINT ["/bin/ping", "localhost"]
    ```
- Build image from Docker file
  - >docker build -t test:2 .
- Run image, which will give output as continuous ping to localhost
- Not if we try to override this using below command, it will give error

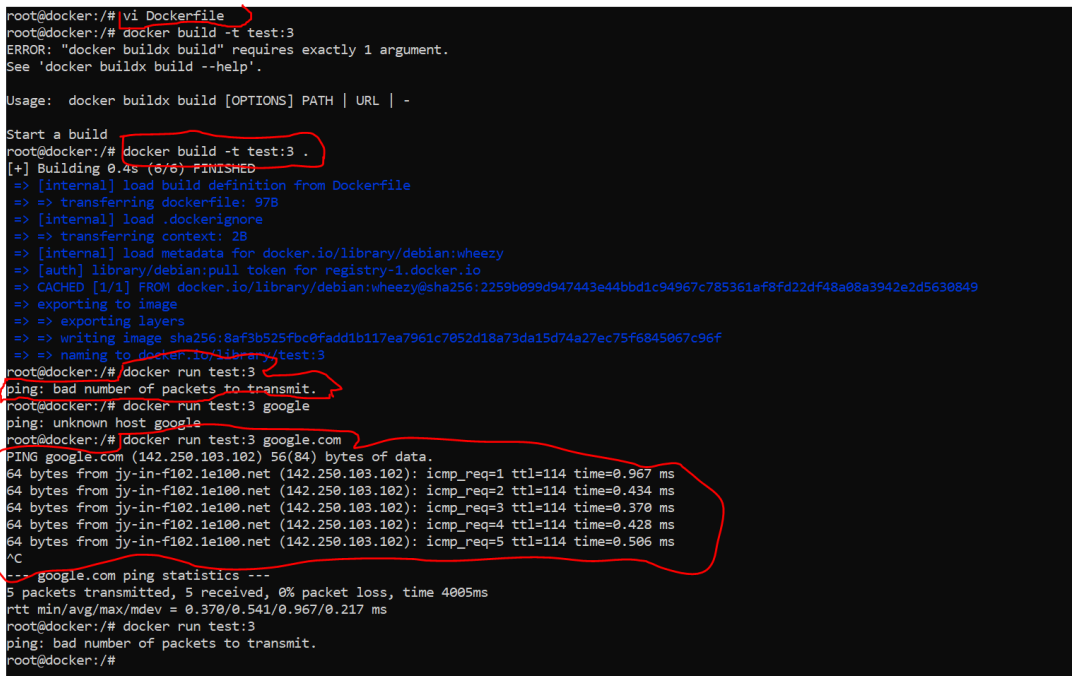
- `> docker run test:2 echo "Hello All"`



- So **ENTRYPOINT** field cannot be overridden
- It is recommended to use both combination, check below example
- Create Docker file
  - `>vi Dockerfile`

```
FROM debian:wheezy
ENTRYPOINT ["/bin/ping"]
 CMD ["localhost"]
```

- Build image from Docker file
  - `>docker build -t test:3 .`
- If we run image, without argument it will give error, so we need to pass argument
  - `>docker run test:3`
  - `> docker run test:3 google.com`



- Both CMD and ENTRYPOINT instructions define what command gets executed when running a container. There are few rules that describe their co-operation.
    - Dockerfile should specify at least one of CMD or ENTRYPOINT commands.
    - ENTRYPOINT should be defined when using the container as an executable.
    - CMD should be used as a way of defining default arguments for an ENTRYPOINT command or for executing an ad-hoc command in a container.
    - CMD will be overridden when running the container with alternative arguments

21) What multistage build?

- Multi-stage builds are a feature introduced in Docker 17.05.
- They allow developers to create a Dockerfile that defines multiple stages for building an image.
- Each stage can have its own set of instructions and build context, which means that the resulting image can be optimized for size and performance.
- For example to create & deploy any application we need to do below steps
    - Get source code
    - Build Jar/War
    - Get Tomcat server
    - Place Jar/War in webapps folder
- For above we can build image in 2 stages
    - To build war
    - Deploy war in tomcat
- Create Docker file
    - >vi Dockerfile

```
FROM maven:3.6.2-jdk-8 AS stage1
#stage1 is an alias name for build stage
```

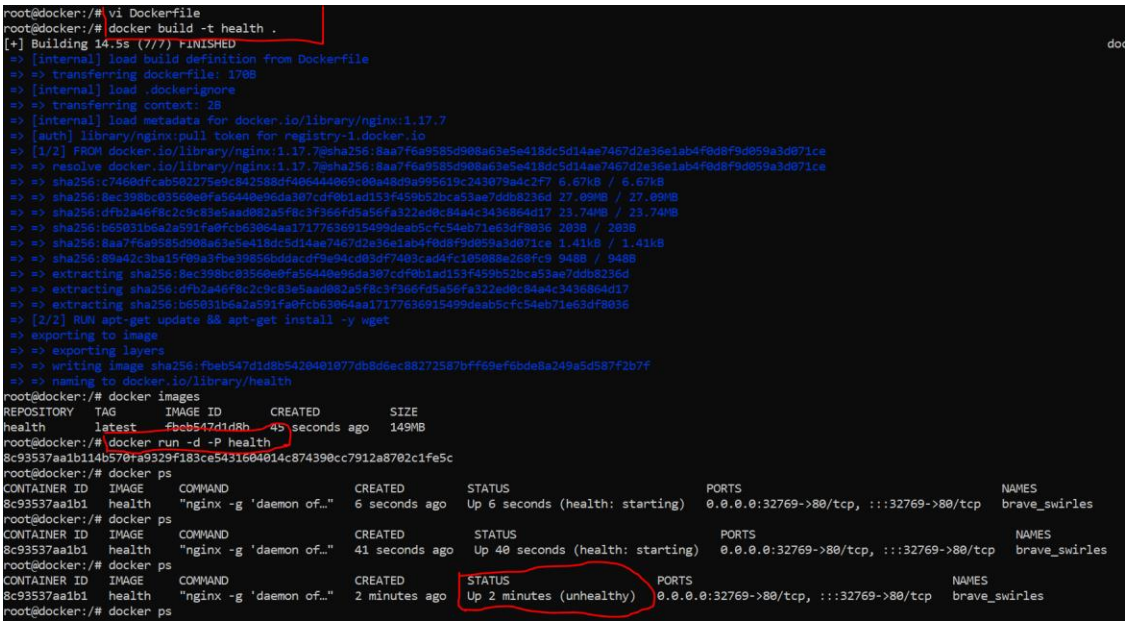
```
RUN git clone http://github.com/lerndevops/petclinic
WORKDIR  /petclinic
RUN mvn package

FROM tomcat:8.5
WORKDIR /usr/local/tomcat/webapps/
COPY --from=stage1 /petclinic/target/petclinic.war .
# importing the .war from build stage
```

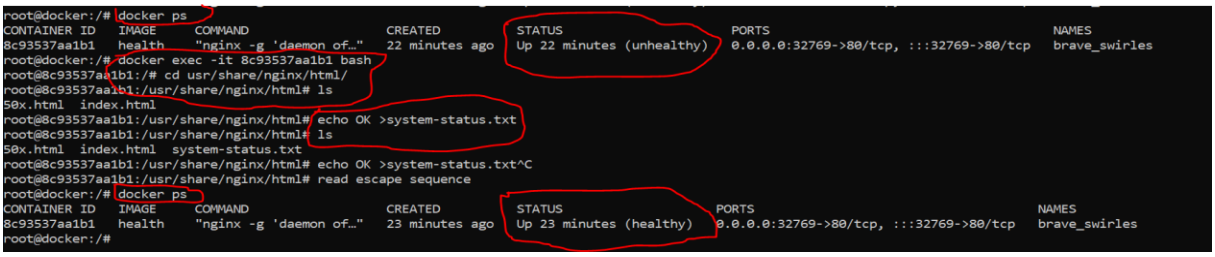
- Build image from Docker file
  - `>docker build -t multi:1 .`
- Build image
  - `>docker run –d -P multi:1`

22) What is health check?

- Health checks in Docker provide a way to monitor the status of the services running inside a Docker container.
- These checks can report whether a container is running as expected.
- The HEALTHCHECK instruction in Docker has two forms:
- **HEALTHCHECK [OPTIONS] CMD command**: This form allows you to set a command that Docker will use to check the health of your container.
- **HEALTHCHECK NONE**: This form disables any health check set in the image or any parent image.
- Check below example
  - Create Docker file
  - `>vi Dockerfile`
  - `FROM nginx:1.17.7`
  - `RUN apt-get update && apt-get install -y wget`
  - `HEALTHCHECK CMD wget -q --method=HEAD localhost/system-status.txt`
  - Build the image and run the container, once it run health check it will show unhealthy, because we don't have file `localhost/system-status.txt` in container
  - `>docker build -t health .`
  - `docker run -d -P health`



- The HEALTHCHECK command can also be used with four different options:
  --interval=DURATION (default: 30s)
  --timeout=DURATION (default: 30s)
  --start-period=DURATION (default: 0s)
  --retries=N (default: 3)
- The **interval** option specifies the number of seconds to initially wait before executing the health check and then the frequency at which subsequent health checks will be performed.
- The **timeout** option specifies the number of seconds Docker awaits for your health check command to return an exit code before declaring it as failed (and your container as unhealthy).
- The **start-period** option specifies the number of seconds your container needs to bootstrap. During this period, health checks with an exit code greater than zero won't mark the container as unhealthy; however, a status code of 0 will mark the container as healthy.
- The **retries** option specifies the number of consecutive health check failures required to declare the container as unhealthy.
- To make above container healthy we need to create file `/usr/share/nginx/html/system-status.txt` in container using below command
  - `docker exec -it 8c93537aa1b1 bash`
  - `cd usr/share/nginx/html/`
  - `echo OK >system-status.txt`
  - `docker ps`



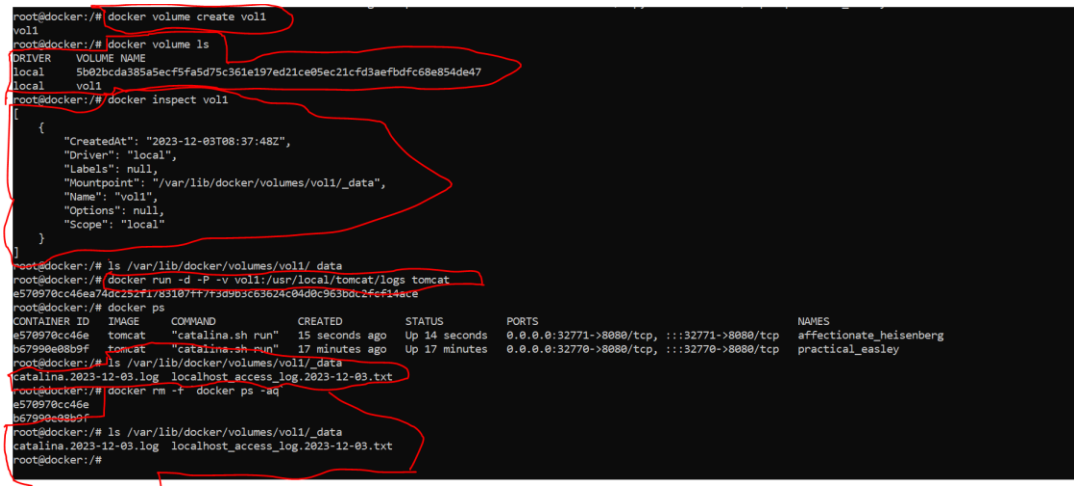
23) What is Docker Volumes & Bind mounts? What is difference between?

- Docker containers are designed so that their internal storage can be easily destroyed. However, sometimes you might need more permanent data.
- **Docker volumes** and bind mounts allow you to attach external storage to containers.
- **Bind Mounts vs. Volumes:**
  - When mounting external storage to a container, you can use either a bind mount or a volume.
  - We can use --mount OR --volume syntax to attach external storage to a container.
- **Bind mounts:**
  1) Mount a specific path on the host machine to the container.
  2) Not portable, dependent on the host machine's file system and directory structure.
- **Volumes:**
  1) Stores data on the host file system, but the storage location is managed by Docker.
  2) More portable.
  3) Can mount the same volume to multiple containers.
  4) Work in more scenarios.
- We can create both volumes/bind mounts using either of the syntax.
- I.e. we can create a "bind mount" using --mount syntax as well with --volume syntax
- Similarly; we can create a "volume" using --mount syntax as well with --volumes syntax.
- --mount syntax: > `docker run --mount type=<value>,source=<value>,destination=<value>,readonly..`
  - **type:** bind (bind mount), volume, or tmpfs (temporary in-memory storage)
  - **Source, src:** Volume name or bind mount path.
  - **Destination, dst, target**: Path to mount inside the container.
  - **Readonly**: Make the volume or bind mount read-only.
- Example:
  - `cd ~/`
  - `mkdir message`
  - `echo Hey, You! > message/message.txt`
  - Mount the directory to a container with a bind mount.
  - `docker run --mount type=bind,source=/home/cloud_user/message,destination=/root,readonly busybox cat /root/message.txt`
  - Run a container with a mounted volume.
  - `docker run --mount type=volume,source=my-volume,destination=/root busybox sh -c 'echo hello > /root/message.txt && cat /root/message.txt'`

24) How to create Volumes & use them?

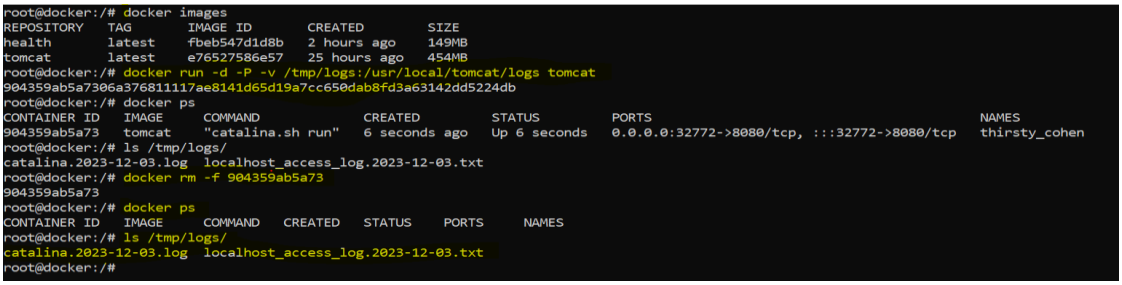
- We can create volume using below command
  - `Syntax:> docker volume create <volume name>`
  - `Ex:> docker volume create vol1`

- Check list of volumes using below command
  - `>docker volume ls`
- Inspect volumes using below command
  - `Syntax> docker inspect <volume name>`
  - `Ex> docker inspect vol1`
- We can use volumes using below command
  - `Syntax> docker run -d -P -v <volume name>:<folder to map> <image>`
  - `Ex> docker run -d -P -v vol1:/usr/local/tomcat/logs tomcat`
- Check data is getting stored in host machine using command
  - `ls /var/lib/docker/volumes/vol1/_data`
- Now even though if we remove/stop container we can see log/data is available on mounted volume path
  - `docker rm -f `docker ps -aq``
  - `ls /var/lib/docker/volumes/vol1/_data`



25) How to give user defined location/bind mount to docker container?

- We can bind our own directory to container to store data by using below command
  - Syntax:> `docker run -d -P -v <destination folder>:<source folder>`
  - Ex:> `docker run -d -P -v /tmp/logs:/usr/local/tomcat/logs tomcat`
- Check on location you can see logs are available there:> `ls /tmp/logs/`



26) How to give user defined data as input to docker container using volume?

- We can give dynamic data to container by using below
- Crete folder & place user data in index.html
  - >cd /root/dev
  - >vi index.html
  - Syntax:>docker run -d -P -v <source folder >:<destination folder> <image>
  - Ex:> docker run -d -P -v /root/dev:/usr/share/nginx/html nginx
- Now you can check in browser by hitting URL:port, our customize html will display message

```
root@docker:~/dev# cd /root/dev/
root@docker:~/dev# ls
index.html
root@docker:~/dev# docker run -d -P -v /root/dev:usr/share/nginx/html nginx
docker: Error response from daemon: invalid volume specification: '/root/dev:usr/share/nginx/html': invalid mount config for type "bind": invalid mount path: 'usr
nx/html' mount path must be absolute.
See 'docker run --help'.
root@docker:~/dev# docker run -d -P -v /root/dev:usr/cd ^Care/nginx/html nginx
root@docker:~/dev# ^C
root@docker:~/dev# ^C
root@docker:~/dev# ^C
root@docker:~/dev# ^C
root@docker:~/dev# cd /
root@docker:/# docker run -d -P -v /root/dev:usr/share/nginx/html nginx
docker: Error response from daemon: invalid volume specification: '/root/dev:usr/share/nginx/html': invalid mount config for type "bind": invalid mount path: 'usr
nx/html' mount path must be absolute.
See 'docker run --help'.
root@docker:/# docker run -d -P -v /root/dev:/usr/share/nginx/html nginx
dd41bb7b8aca8194c2303db0b0ba6e15e9451f3936a991698e903cc126582939
root@docker:/# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                     NAMES
dd41bb7b8aca   nginx     "/docker-entrypoint.…"   13 seconds ago   Up 12 seconds   0.0.0.0:32774->80/tcp, :::32774->80/tcp   xenodochial_kalam
a04d5eb4ac24   nginx     "/docker-entrypoint.…"   9 minutes ago    Up 9 minutes    0.0.0.0:32773->80/tcp, :::32773->80/tcp   ecstatic_moore
root@docker:/#
```

- So we can give dynamic data to container without updating its content by using volume concept

27) What is docker daemon? How we can customize docker?

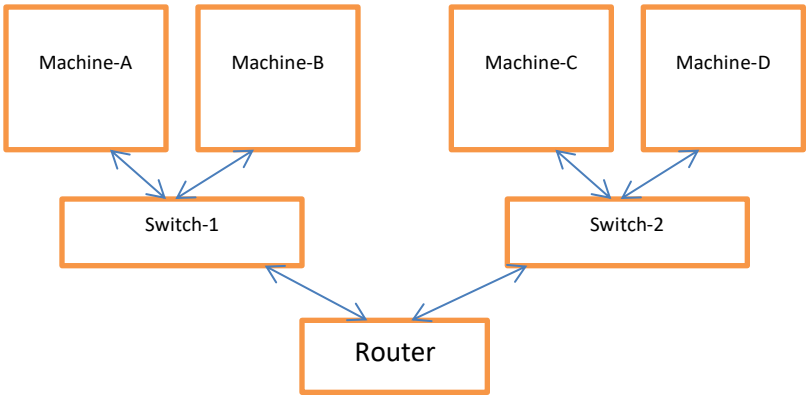
- docker daemon is software running in host machine.
- We can customize docker by changing daemon configuration file in JSON format.
- The default location of the configuration file on Linux is `/etc/docker/daemon.json`
- We can update logging driver as below:
  - `sudo vi /etc/docker/daemon.json`
  - `{"log-driver": "syslog"}`
  - OR
  - ```
    {
          log-driver": "json-file",
          "log-opts": {
                "max-size": "15m"
        }
    }
    ```
  - `>sudo systemctl restart docker`
  - `>docker info | grep Logging`
- We can enable debugging :
  - `vi /etc/docker/daemon.json`
  - `{"debug": true}`
    ```

- We can configure storage driver:
  - `sudo vi /etc/docker/daemon.json`
  - `{"storage-driver": "devicemapper"}`
- We can configure a default registry:
  - `{"registry-mirrors": ["https://<my-docker-mirror-host>"] }`
- We can secure the docker daemon with TLS / SSL:

```
{
  "tlsverify": true,
  "tlscacert": "/home/certs/ca.pem",
  "tlscert": "/home/certs/server-cert.pem",
  "tlskey": "/home/certs/server-key.pem"
}
```

28) How Networking work in Docker?

- To access any service running on any container/machine we use **PORT** Number.
- Machine to machine communication happened by using **SWITCH**.
- If one switch wants to communicate with another switch we can use **ROUTER**.

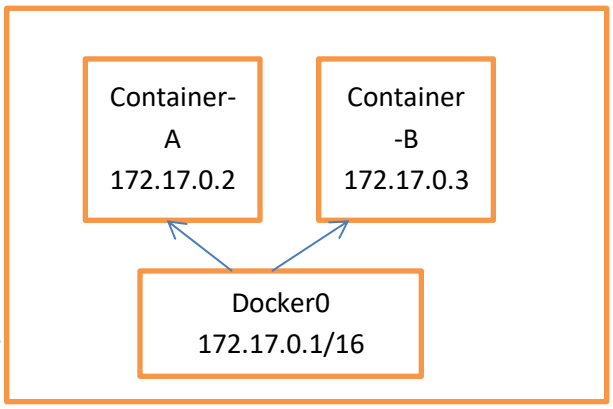


- Docker implements container networking using a framework called the Container Networking Model (CNM) and manages the networking for containers.
- The CNM utilizes the following concepts:
- **Sandbox:** An isolated unit containing all networking components associated with a single container. Usually a Linux network namespace.
- **Endpoint:** Connects a sandbox to a network. Each sandbox/container can have any number of endpoints, but has exactly one endpoint for each network it is connected to.
- **Network**: A collection of endpoints connected to one another.
- **Network Driver**: Handles the actual implementation of the CNM concepts.
- **IPAM Driver**: IPAM means IP Address management. Automatically allocates subnets and IP Addresses for networks and endpoints.

Network Drivers:

- Docker includes several built-in network drivers, known as Native Network Drivers.
- These network drivers implement the concepts described in the CNM.
- The Native Network Drivers are:
    1) host
    2) bridge
    3) overlay
    4) macvlan
    5) none
- All the containers running in host get IP address from docker0 network.
- We can check by using command `>ip a`

```
root@docker:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group default qlen 1000
    link/ether 42:01:0a:80:00:02 brd ff:ff:ff:ff:ff:ff
    inet 10.128.0.2/32 scope global dynamic ens4
       valid_lft 3424sec preferred_lft 3424sec
    inet6 fe80::4001:aff:fe80:2/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:3d:90:87:de brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
root@docker:/#
```

- 

    - `>docker run -d -P nginx`
    - `>docker run -d -P redis`
    - Inspect redis to take IP ("IPAddress": "172.17.0.3")
    - `>docker inspect bd2e67f6a`

```
root@docker:/# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
bd2e67f6a9fd   redis     "docker-entrypoint.s…"   7 seconds ago    Up 6 seconds    0.0.0.0:32769->6379/tcp, :::32769->6379/tcp            elegant_mcnulty
55a4aa9c5407   nginx     "/docker-entrypoint.…"   30 seconds ago   Up 29 seconds   0.0.0.0:32768->80/tcp, :::32768->80/tcp                naughty_meninsky
root@docker:/# docker inspect bd2e67f6a
```

- Login in nginix
- > `docker exec -it 55a4aa9c5407 bash`
- Try to ping redish IP ("IPAddress": "172.17.0.3") it will get connected
- > `ping 172.17.0.3`

```
root@55a4aa9c5407:/# ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3) 56(84) bytes of data.
64 bytes from 172.17.0.3: icmp_seq=1 ttl=64 time=0.190 ms
64 bytes from 172.17.0.3: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 172.17.0.3: icmp_seq=3 ttl=64 time=0.109 ms
64 bytes from 172.17.0.3: icmp_seq=4 ttl=64 time=0.083 ms
^C
```

- One container can communicate with another by IP only, it cannot communicate with container Name or container Id

29) What is The Bridge Network Driver?

- The Bridge Network Driver uses Linux bridge networks to provide connectivity between containers on the same host.
- By default docker0 network uses bridge drivers to run container.
- To make name based communication within container we need to create our own bridge and our own network.
- Command to create our own network
  - Syntax:> `docker network create <name of network>`
  - `EX:> docker network create app1`
- Command to check all network
  - Syntax:> docker network ls
- Command to inspect any network
  - Syntax :> `docker network inspect <network name>`
  - Ex:>`docker network inspect app1`
- Command to remove any network
  - Syntax :> `docker network rm <network name>`
  - Ex:>`docker network rm app1`
- Command to connect any network
  - Syntax :> `docker network connect <network name> <container name>`
  - Ex:>`docker network connect app1 myApp`
- Command to disconnect any network
  - Syntax :> `docker network disconnect <network name> <container name>`
  - Ex:>`docker network disconnect app1 myApp`
  -

```
root@docker:/#
root@docker:/# docker network create app1
d0ad4b0ebff10daf18c9187c711399e00e82727037f36de130143ebcf83823c5
root@docker:/# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
d0ad4b0ebff1   app1      bridge    local
c77eb8e7a3a6   bridge    bridge    local
b25736214de5   host      host      local
d4aa7599854a   none      null      local
root@docker:/# docker  inspect app1
[
    {
        "Name": "app1",
        "Id": "d0ad4b0ebff10daf18c9187c711399e00e82727037f36de130143ebcf83823c5",
        "Created": "2023-12-09T05:06:41.9494334052",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

- Command to run container using our own network
    - Syntax:> `docker run -d -P --name <app name> --net <network name> <image name>`
    - Ex:> `docker run -d -P --name myapp --net app1 nginx`
- Now we can inspect our own application & see our own network is assign to application
    - Ex:> `docker inspect myapp`

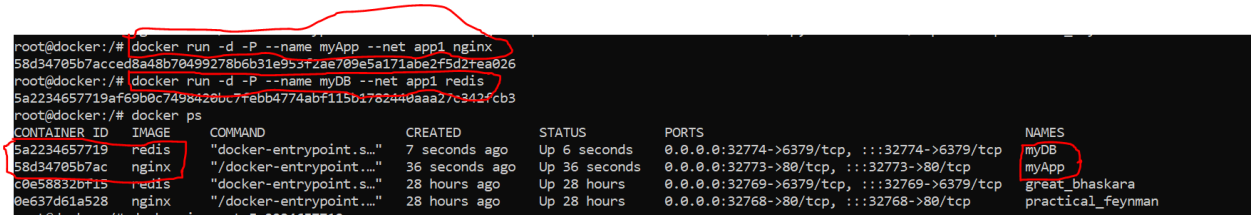
```
root@docker:/# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                                          NAMES
0f4c751c7f52   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:32770->80/tcp, :::32770->80/tcp        myapp
c0e58832bf15   redis     "docker-entrypoint.s…"   27 hours ago    Up 27 hours    0.0.0.0:32769->6379/tcp, :::32769->6379/tcp    great_bhaskara
0e637d61a528   nginx     "/docker-entrypoint.…"   27 hours ago    Up 27 hours    0.0.0.0:32768->80/tcp, :::32768->80/tcp        practical_feynman
root@docker:/# docker inspect myapp
```

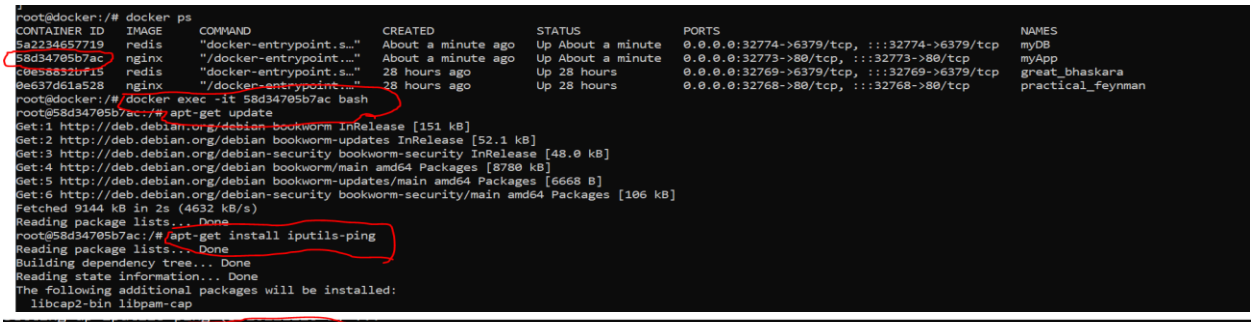
```
        "Networks": {
            "app1": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": [
                    "0f4c751c7f52"
                ],
                "NetworkID": "d0ad4b0ebff10daf18c9187c711399e00e82727037f36de130143ebcf83823c5",
                "EndpointID": "1d4a7566a32d20207aabe5055e2b231966964b747b1e581aa4205872f0294173",
                "Gateway": "172.18.0.1",
                "IPAddress": "172.18.0.2",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:12:00:02",
                "DriverOpts": null
            }
        }
    }
]
```

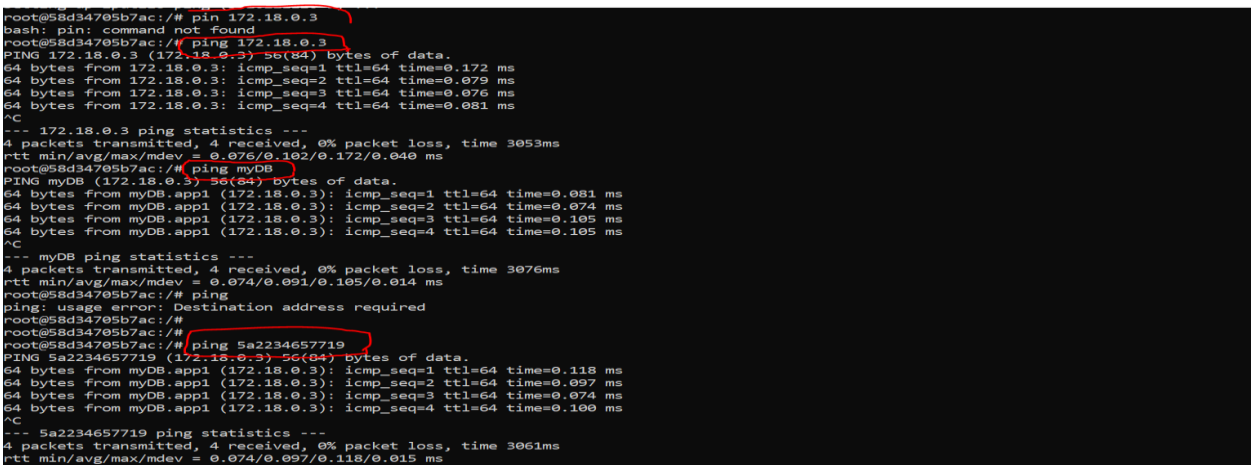
- Now we create one DB container using same network & check communicate using IP/name/containerId
    - Ex:> `docker run -d -P --name mydb --net app1 redis`

```
root@docker:/# docker run -d -P --name myDB --net app1 redis
4b61933324c396491de223a2d99a5f32753a1a781e53853d1c556d4917a848b5
root@docker:/# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                          NAMES
4b61933324c3   redis     "docker-entrypoint.s…"   5 seconds ago    Up 4 seconds    0.0.0.0:32772->6379/tcp, :::32772->6379/tcp    myDB
0f4c751c7f52   nginx     "/docker-entrypoint.…"   25 minutes ago   Up 25 minutes   0.0.0.0:32770->80/tcp, :::32770->80/tcp        myapp
c0e58832bf15   redis     "docker-entrypoint.s…"   28 hours ago     Up 28 hours     0.0.0.0:32769->6379/tcp, :::32769->6379/tcp    great_bhaskara
0e637d61a528   nginx     "/docker-entrypoint.…"   28 hours ago     Up 28 hours     0.0.0.0:32768->80/tcp, :::32768->80/tcp        practical_feynman
root@docker:/# docker inspect 4b61933324c3
[
```

- Note its DB container's IP/name/container Id to check connectivity from other container
  - IP> `172.18.0.3`
  - Name>`myDB`
- Container Id> `5a2234657719`
- Login on **myApp** container & check ping to **myDB** container
- We can ping now myDB container from myApp container using IP/Name/Container Id







30) What is The Host Network Driver?

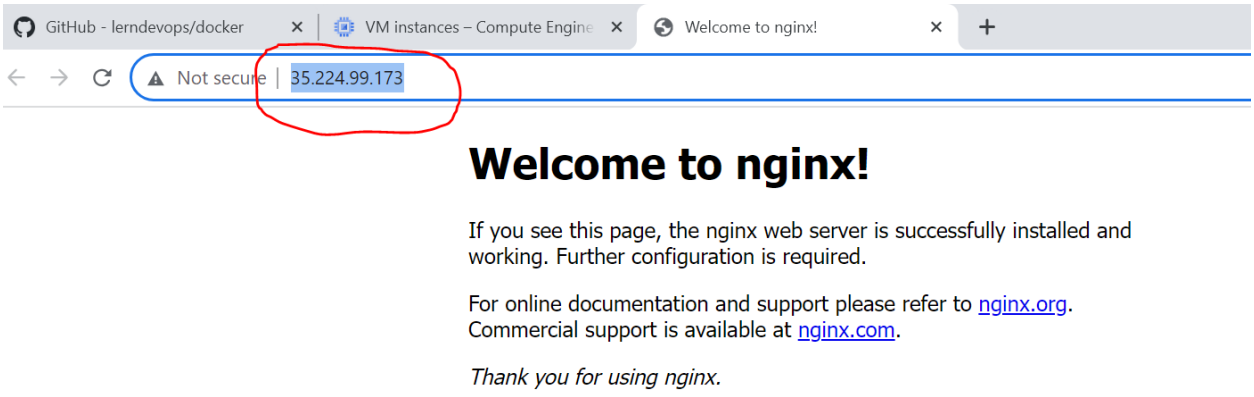
- The Host Network Driver allows containers to use the host's network stack directly.
- Containers use the host's networking resources directly.
- No sandboxes, all containers on the host using the host driver share the same network namespace
- No two containers can use the same port(s)
- Use Cases: Simple and easy setup, one or only few containers on a single host.
- We can run a container which can be accessible using Host IP using below command
    - Syntax:>`docker run -d -P --net host <image>`
    - Ex:> `docker run -d -P --net host  nginx`

```
root@docker:/# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
d0ad4b0ebff1   app1      bridge    local
c77eb8e7a3a6   bridge    bridge    local
b25736214de5   host      host      local
d4aa7599854a   none      null      local
root@docker:/# docker run -d -P --net host nginx
e6dbfd8e6ec576bc9897a1d5e5222dda4f1d3e610947d8dc65a02f90d5d93e56
root@docker:/# docker inspect e6dbfd8e6ec
[
    {
        "Id": "e6dbfd8e6ec576bc9897a1d5e5222dda4f1d3e610947d8dc65a02f90d5d93e56",
        "Created": "2023-12-09T06:50:27.257582417Z",
        "Path": "/docker-entrypoint.sh",
```

- We can check by inspecting this container, it is using host network only

```
            "Networks": {
                "host": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "b25736214de55b47145fd6c6e99959e8cccf5f6f54f34136a059e335c1044ff7",
                    "EndpointID": "0a0b594aa0171786ade3daa3fd562a79b7e011881d596f5d3566de352865fd09",
                    "Gateway": "",
                    "IPAddress": "",
                    "IPPrefixLen": 0,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

- We can access this container, it is using host IP & application port http://35.224.99.173/



31) What is The None Network Driver?

- The None Network Driver does not provide any networking implementation.
- Container is completely isolated from other containers and the host.
- If you want networking with none driver, you must set everything up manually.
- None does create a separate networking namespace for each container, but no interfaces or endpoints.
- If we don't want to give accesses anywhere to our container we can use none network.
- We can run container with none network using below command
    - Syntax:>`docker run -d -P --net none <image name>`
    - Ex:>`docker run -d -P --net none nginx`

```
root@docker:/# docker run -d -P --net none nginx
47b103ae80e0996341d8b871948037963b9a7e829d16d51ebb8f7c2121c65329
root@docker:/# docler ps

Command 'docler' not found, did you mean:

  command 'docker' from snap docker (20.10.24)
  command 'docker' from deb docker.io (24.0.5-0ubuntu1~20.04.1)

See 'snap info <snapname>' for additional versions.

root@docker:/# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED             STATUS             PORTS     NAMES
47b103ae80e0   nginx     "/docker-entrypoint.…"   11 seconds ago      Up 10 seconds                upbeat_meninsky
e6dbfd8e6ec5   nginx     "/docker-entrypoint.…"   About an hour ago   Up About an hour             laughing_hoover
root@docker:/# docker inspect 47b103ae80e0
[
    {
        "Id": "47b103ae80e0996341d8b871948037963b9a7e829d16d51ebb8f7c2121c65329",
        "Created": "2023-12-09T07:52:53.573899825Z",
        "Path": "/docker-entrypoint.sh",
        "Args": [
```

- We can inspect this container & see no IP address is assign to this
    - Ex:>docker inspect 47b103ae80e0

```
            "Networks": {
                "none": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "d4aa7599854a3293f88295aa0a65cece2f770fc78523b2b16411c4c7385da7ea",
                    "EndpointID": "0cd166bf9da272ddf277ad27b3e1d7f72cda91941c94c33fda15aa16c5be7ae7",
                    "Gateway": "",
                    "IPAddress": "",
                    "IPPrefixLen": 0,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "",
                    "DriverOpts": null
                }
            }
        }
    }
]
```

32) What is The Overlay Network Driver?

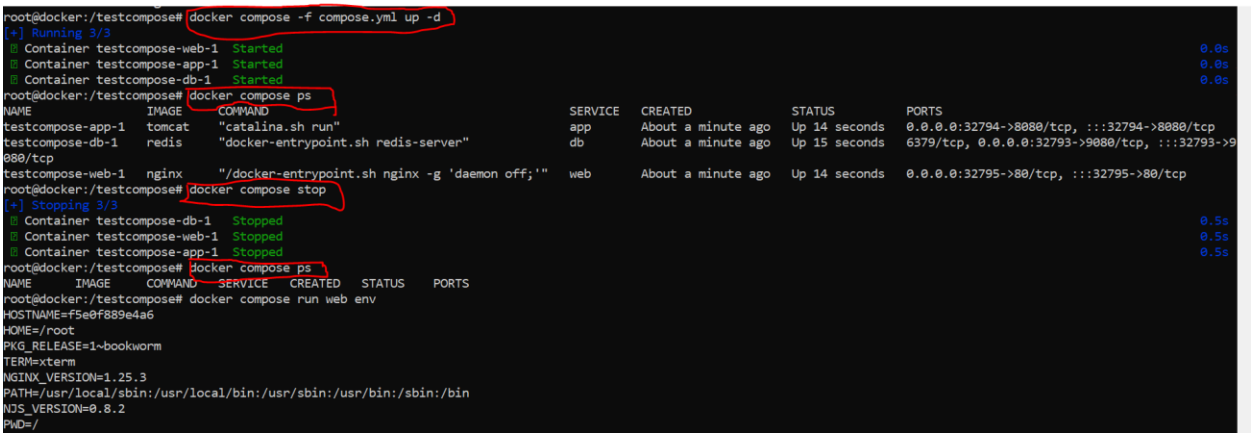
- The Overlay Network Driver provides connectivity between containers across multiple Docker hosts, i.e. with Docker swarm.
- Uses a VXLAN data plane, which allows the underlying network infrastructure (underlay) to route data between hosts in a way that is transparent to the containers themselves.
- This is used for networking between containers in a swarm.
- It helps for communication between containers across machines also.
- When we use multiple host machines we can use overlay network.

33) What is Docker Compose?

- Docker Compose is a tool that helps you define and share multi-container applications.
- With Compose, you can create a YAML file to define the services and with a single command, you can spin everything up or tear it all down.
- For example if we want to create application having UI, Application & DB layer we can create YML file & run
- Create folder “testcompose”
  - >mkdir testcompose
  - >cd testcompose/
- Create compose.yml
  - >vi compose.yml

```
services:
  web:
    image: nginx
    ports:
      - "80"
    volumes:
      - /home/dockerlabs/dockerfiles/:/var/www/html
  app:
    image: tomcat
    ports:
      - "8080"
  db:
    image: redis
    ports:
      - "9080"
```

- Use below command to run application in detached mode using compose
  - `>docker compose -f compose.yml up -d`
- Use below command to check application using compose
  - `>docker compose ps`
- Use below command to stop application using compose
  - `>docker compose stop`
- Use below command to run one-off commands for your services, For example, to see what environment variables are available to the web service
  - `>docker compose run web env`

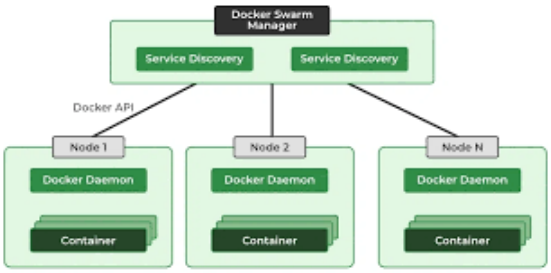


- We can scale up & down containers for each service, suppose we want 1 web, 2 app & 1 db container so we can use below command
    - `> docker-compose -f compose.yml up --scale web=1 --scale app=2 --scale db=1 -d`
- You can bring everything down, removing the containers entirely, with the down command. Pass --volumes to also remove the data volume used by the nginx container:
    - `>docker compose down`

```
root@docker:/testcompose# docker compose -f compose.yml up --scale web=1 --scale app=2 --scale db=1 -d
[+] Running 4/4
 ⠿ Container testcompose-db-1   Started                                                                    0.1s
 ⠿ Container testcompose-web-1  Started                                                                    0.1s
 ⠿ Container testcompose-app-2  Started                                                                    0.1s
 ⠿ Container testcompose-app-1  Started                                                                    0.1s
root@docker:/testcompose# docker compose ps
NAME                IMAGE     COMMAND                                  SERVICE   CREATED          STATUS         PORTS
testcompose-app-1   tomcat    "catalina.sh run"                        app       14 seconds ago   Up 13 seconds  0.0.0.0:32801->8080/tcp, :::32801->8080/tcp
testcompose-app-2   tomcat    "catalina.sh run"                        app       14 seconds ago   Up 12 seconds  0.0.0.0:32803->8080/tcp, :::32803->8080/tcp
testcompose-db-1    redis     "docker-entrypoint.sh redis-server"      db        14 seconds ago   Up 13 seconds  6379/tcp, 0.0.0.0:32802->9080/tcp, :::32802->9080/
tcp
testcompose-web-1   nginx     "/docker-entrypoint.sh nginx -g 'daemon off;'"   web       14 seconds ago   Up 13 seconds  0.0.0.0:32800->80/tcp, :::32800->80/tcp
root@docker:/testcompose# docker compose ps
NAME                IMAGE     COMMAND                                  SERVICE   CREATED          STATUS         PORTS
testcompose-app-1   tomcat    "catalina.sh run"                        app       18 minutes ago   Up 18 minutes  0.0.0.0:32801->8080/tcp, :::32801->8080/tcp
testcompose-app-2   tomcat    "catalina.sh run"                        app       18 minutes ago   Up 18 minutes  0.0.0.0:32803->8080/tcp, :::32803->8080/tcp
testcompose-db-1    redis     "docker-entrypoint.sh redis-server"      db        18 minutes ago   Up 18 minutes  6379/tcp, 0.0.0.0:32802->9080/tcp, :::32802->9080/
tcp
testcompose-web-1   nginx     "/docker-entrypoint.sh nginx -g 'daemon off;'"   web       18 minutes ago   Up 18 minutes  0.0.0.0:32800->80/tcp, :::32800->80/tcp
root@docker:/testcompose# docker compose down --volumes
[+] Running 5/5
 ⠿ Container testcompose-app-2  Removed                                                                    0.6s
 ⠿ Container testcompose-db-1   Removed                                                                    0.4s
 ⠿ Container testcompose-web-1  Removed                                                                    0.5s
 ⠿ Container testcompose-app-1  Removed                                                                    0.6s
 ⠿ Network testcompose_default  Removed                                                                    0.1s
root@docker:/testcompose#
```

34) What is docker swarm?

- Docker Swarm is a clustering and scheduling tool for Docker containers.
- With Swarm, IT administrators and developers can establish and manage a cluster of Docker nodes as a single virtual system.
- A Docker Swarm cluster also provides administrators and developers with the ability to add or subtract container iterations as computing demands change.



- Docker swarm consist of
    - Node
        - Worker
        - Manager
    - Service
    - Taks
    - Load balancing

- A **node** is an instance of the Docker engine participating in the swarm.
- **Manager nodes** perform the orchestration and cluster management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks.
- By default manager nodes also run services as worker nodes, but you can configure them to run manager tasks exclusively and be manager-only nodes.
- **Worker nodes** receive and execute tasks dispatched from manager nodes.
- An agent runs on each worker node and reports on the tasks assigned to it.
- The worker node notifies the manager node of the current state of its assigned tasks so that the manager can maintain the desired state of each worker.
- A **service** is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm.
- A **service** is used to run an application on a Docker Swarm. A service specifies a set of one or more replica tasks. These tasks will be distributed automatically acorss the nodes in the cluster and executed as containers.
- A **stack** is nothing but a collection of one or more services deployed as a single unit.
- The **stack** is deployed by using compose file in which you can mention the service of the stack and all the required configurations to deploy the stack
- The swarm manager uses **ingress load balancing** to expose the services you want to make available externally to the swarm.
- The swarm manager uses **internal load balancing** to distribute requests among services within the cluster based upon the DNS name of the service.

35) How to setup docker swarm?

- To setup docker swarm, it requires three Linux hosts which have Docker installed and can communicate over a network. These can be physical machines, virtual machines, Amazon EC2 instances, or hosted in some other way.
- One of these machines is a manager (called **manager**) and two of them are workers (**worker1** and **worker2**)
- Create 3 users on these 3 machines, so that we can connect to cloud machine using putty or any other terminal.
- By default swarm feature is there but not enable
- To enable docker swarm we need to use blow command, this command should run on manager machine. If we run this command on any host it will become manager.
    - `> docker swarm init`
- This command will give output as command to join/add worker nodes to this manager

```
root@manager:/# docker swarm init
Swarm initialized: current node (niv7hyaqszvs9799hyeag3var) is now a manager.

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

    docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-dt66mqtftwjk01r42dn766q7w 10.128.0.2:2377

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

root@manager:/#
```

- Use below command in worker machine to join manager
- `>docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-dt66mqtftwjk01r42dn766q7w 10.128.0.2:2377`

```
root@worker-1:/# docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-dt66mqtftwjk01r42dn766q7w 10.128.0.2:2377
This node joined a swarm as a worker.
root@worker-1:/#
```

```
root@worker-2:/# docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-dt66mqtftwjk01r42dn766q7w 10.128.0.2:2377
This node joined a swarm as a worker.
root@worker-2:/#
```

- Use below command in manager machine to check number of nodes joins. We cannot run this command from worker nodes.
    - `>docker node ls`

```
root@manager:/home/devops# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Ready     Active                          24.0.7
root@manager:/home/devops#
```

- We can join any host as manager or worker by using below command on manager node. This token can be generated at any point of time. We cannot run this command from worker node
    - `> docker swarm join-token manager`
    - `> docker swarm join-token worker`

```
root@manager:/home/devops# docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-f3hgz3tfxahovtgrkavn0nyr9 10.128.0.2:2377

root@manager:/home/devops# docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-0kamccwnn8e1poto0lwef978y6ko9xhqco10w7tins64vp75pq-dt66mqtftwjk01r42dn766q7w 10.128.0.2:2377

root@manager:/home/devops#
```

36) What is docker service in docker swarm?

- A service is used to run an application on a Docker Swarm.
- A service creates a set of one or more replica tasks (containers) to run on multiple machines.
- These tasks will be distributed automatically across the nodes in the cluster and executed as containers.
- We can use below command to create single service in docker swarm
    - `Syntax:> docker service create --name <service name> -p <new port>:<default port> <image>`
    - `Ex:> docker service create --name myapp -p 1234:80 nginx`

- But in real world we need to multiple services using below command
    - Syntax:> docker service create --name <service name> --replicas <number of replica> -p <new port>:<default port> <image>
    - Ex:> docker service create --name myapp --replicas 5 -p 1234:80 nginx
- We can use below command to list services available in docker swarm
    - Syntax:> docker service ls
- We can use below command to delete services available in docker swarm
    - Syntax:> docker service rm <service names space separated>
- We can use below command to check in which host containers are running
    - Syntax:> docker service ps <service name >

root@manager: /

```
root@manager:/# docker service create --name myapp -p 1234:80 nginx
kp4qehxgzdm1quj2q1niiynp4
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ls
ID             NAME                MODE         REPLICAS   IMAGE          PORTS
kp4qehxgzdm1   myapp               replicated   1/1        nginx:latest   *:1234->80/tcp
gvzzlnv95lbw   recursing_torvalds   replicated   0/1        myapp:latest
root@manager:/# docker service rm myapp recursing_torvalds
myapp
recursing_torvalds
root@manager:/# docker service create --name myapp --replicas 5 -p 1234:80 nginx
jc5smdhwq39o3r1e82xmglvgm
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
jc5smdhwq39o   myapp     replicated   5/5        nginx:latest   *:1234->80/tcp
root@manager:/# docker service ps myapp
ID             NAME      IMAGE          NODE       DESIRED STATE   CURRENT STATE           ERROR     PORTS
jteeupsgls6q   myapp.1   nginx:latest   worker-1   Running         Running 2 minutes ago
zjo7p0t01ti8   myapp.2   nginx:latest   worker-2   Running         Running 2 minutes ago
ylht0xys98hl   myapp.3   nginx:latest   worker-1   Running         Running 2 minutes ago
p46tbh5aoosd   myapp.4   nginx:latest   worker-2   Running         Running 2 minutes ago
vy3iosc9bceu   myapp.5   nginx:latest   manager    Running         Running 2 minutes ago
root@manager:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
23e5df41dc0c   nginx:latest   "/docker-entrypoint.…"   4 minutes ago   Up 4 minutes   80/tcp    myapp.5.vy3iosc9bceu6xs9uc4djihic
root@manager:/#
```

root@worker-1: /

```
root@worker-1:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
9c89ac5d5049   nginx:latest   "/docker-entrypoint.…"   9 minutes ago   Up 9 minutes   80/tcp    myapp.3.ylht0xys98hl3r38gglwrzebs
2ffeb0bb1764   nginx:latest   "/docker-entrypoint.…"   9 minutes ago   Up 9 minutes   80/tcp    myapp.1.jteeupsgls6qnzig3ma2iabtz
root@worker-1:/#
```

root@worker-2: /

```
root@worker-2:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
cf1ee97a1f67   nginx:latest   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   80/tcp    myapp.4.p46tbh5aoosdpzc7gx7k3kewh
c99e14b82906   nginx:latest   "/docker-entrypoint.…"   10 minutes ago   Up 10 minutes   80/tcp    myapp.2.zjo7p0t01ti8nujvcztbrob7n
root@worker-2:/#
```

37) How can we achieve high availability & desire state in docker service by using docker swarm?

- If we remove any number of container from worker, than docker swarm will create that automatically, this is call **desired state & high availability**.
- We can check this by deleting all containers from worker 1.
- Even after deleting all container we can see that container is getting created again

root@worker-1: /

```
root@worker-1:/# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
a5b2ad82c5ae   nginx:latest   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   80/tcp    myapp.3.9qgams3oi6v8pyy04g3yvw0o7
root@worker-1:/# docker rm -f `docker ps -aq`
a5b2ad82c5ae
root@worker-1:/# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS    PORTS     NAMES
f20796912a44   nginx:latest   "/docker-entrypoint.…"   3 seconds ago   Created             myapp.3.kzmhv1w79qj5z3c7npqtypict
root@worker-1:/#
```

root@manager: /

```
root@manager:/# docker service ps myapp
ID             NAME          IMAGE          NODE       DESIRED STATE   CURRENT STATE            ERROR                         PORTS
t36klvx3c1lp   myapp.1       nginx:latest   manager    Running         Running 11 minutes ago
27ntgyhrfc8s    \_ myapp.1   nginx:latest   worker-1   Shutdown        Failed 11 minutes ago    "task: non-zero exit (137)"
jteeupsgls6q    \_ myapp.1   nginx:latest   worker-1   Shutdown        Failed 33 minutes ago    "task: non-zero exit (255)"
d2t3lnt5i34m   myapp.2       nginx:latest   worker-2   Running         Running 33 minutes ago
zjo7p0t01ti8    \_ myapp.2   nginx:latest   worker-2   Shutdown        Failed 33 minutes ago    "task: non-zero exit (255)"
kzmhv1w79qj5   myapp.3       nginx:latest   worker-1   Running         Running 2 minutes ago
9qgams3oi6v8    \_ myapp.3   nginx:latest   worker-1   Shutdown        Failed 2 minutes ago     "task: non-zero exit (137)"
b2ey8qffp2oz    \_ myapp.3   nginx:latest   worker-1   Shutdown        Failed 11 minutes ago    "task: non-zero exit (137)"
ylht0xys98hl    \_ myapp.3   nginx:latest   worker-1   Shutdown        Failed 33 minutes ago    "task: non-zero exit (255)"
lnwljc74nfsx   myapp.4       nginx:latest   worker-2   Running         Running 33 minutes ago
p46tbh5aoosd    \_ myapp.4   nginx:latest   worker-2   Shutdown        Failed 33 minutes ago    "task: non-zero exit (255)"
q70kpwn0el06   myapp.5       nginx:latest   manager    Running         Running 33 minutes ago
vy3iosc9bceu    \_ myapp.5   nginx:latest   manager    Shutdown        Failed 33 minutes ago    "task: non-zero exit (255)"
root@manager:/# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
jc5smdhwq39o   myapp     replicated   5/5        nginx:latest   *:1234->80/tcp
root@manager:/#
```

- If any of worker host machine is getting down, than docker swarm will **create desired number** of containers on other host machine i.e. on manager or other worker.
- We can check this by stopping docker on worker1
    - `Syntax> service docker stop`
- After stopping docker on worker1, we can check in manager for how many container are running by using below command
    - `Syntax> docker service ps myapp | grep -i running`

root@worker-1: /

```
root@worker-1:/# service docker stop
Warning: Stopping docker.service, but it can still be activated by:
  docker.socket
root@worker-1:/#
```

root@manager: /

```
root@manager:/# docker service ps myapp | grep -i running
t36klvx3c1lp   myapp.1   nginx:latest   manager    Running   Running 46 minutes ago
d2t3lnt5i34m   myapp.2   nginx:latest   worker-2   Running   Running about an hour ago
2vpdyn08rr7x   myapp.3   nginx:latest   worker-2   Running   Running 21 minutes ago
lnwljc74nfsx   myapp.4   nginx:latest   worker-2   Running   Running about an hour ago
q70kpwn0el06   myapp.5   nginx:latest   manager    Running   Running about an hour ago
```

- Now even if stopped worker1 host machine is bring up again, docker swarm will not create containers on that host machine, it will keep running container on that machine where ever it was running. We can check this by starting docker on worker1 again by using below command.
- `Syntax> service docker start`
- Check below command on manager machine
- `Syntax>docker service ps myapp | grep -i running`

root@worker-1: /

```
root@worker-1:/# service docker start
root@worker-1:/# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@worker-1:/#
```

root@manager: /

```
root@manager:/# docker service ps myapp | grep -i running
t36klvx3c1lp   myapp.1       nginx:latest   manager    Running         Running 46 minutes ago
d2t3lnt5i34m   myapp.2       nginx:latest   worker-2   Running         Running about an hour ago
2vpdyn08rr7x   myapp.3       nginx:latest   worker-2   Running         Running 21 minutes ago
lnwljc74nfsx   myapp.4       nginx:latest   worker-2   Running         Running about an hour ago
q70kpwn0el06   myapp.5       nginx:latest   manager    Running         Running about an hour ago
```

38) How can we achieve scalability in docker service by using docker swarm?

- If we want to add containers on restarted worker1 host machine, we need to increase or **scale** number of replicas, i.e. if we add more replicas (**scale**) to our application; automatically load will be distributed on empty host machine, in this way we can achieve **scalability.**
- We can increase replicas or **scale our** application by using below command on manager
- `Syntax> docker service scale <application name>=<number of replica>`
- `Ex> docker service scale myapp=7`
- We can check replicas using below command on manager
- `Ex> docker service ps myapp | grep -i running`

root@manager: /

```
root@manager:/# docker service scale myapp=7
myapp scaled to 7
overall progress: 7 out of 7 tasks
1/7: running   [==================================================>]
2/7: running   [==================================================>]
3/7: running   [==================================================>]
4/7: running   [==================================================>]
5/7: running   [==================================================>]
6/7: running   [==================================================>]
7/7: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps myapp | grep -i running
t36klvx3c1lp   myapp.1       nginx:latest   manager    Running         Running about an hour ago
d2t3lnt5i34m   myapp.2       nginx:latest   worker-2   Running         Running 2 hours ago
2vpdyn08rr7x   myapp.3       nginx:latest   worker-2   Running         Running 49 minutes ago
lnwljc74nfsx   myapp.4       nginx:latest   worker-2   Running         Running 2 hours ago
q70kpwn0el06   myapp.5       nginx:latest   manager    Running         Running 2 hours ago
k6dod0l6w1qn   myapp.6       nginx:latest   worker-1   Running         Running 22 seconds ago
wk7neoquqe4g   myapp.7       nginx:latest   worker-1   Running         Running 22 seconds ago
root@manager:/#
```

root@worker-1: /

```
root@worker-1:/# service docker start
root@worker-1:/# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@worker-1:/# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
12d95017ff94   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes   80/tcp    myapp.7.wk7neoquqe4gqkrz01g99ulzf
5f356ea44c46   nginx:latest   "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes   80/tcp    myapp.6.k6dod0l6w1qnhl47svzbgu96e
root@worker-1:/#
```

- We can also reduce replicas also using below command on manager
- `Ex> docker service scale myapp=3`

- Ex> docker service ps myapp | grep -i running

```
Select root@manager: /
root@manager:/# docker service scale myapp=3
myapp scaled to 3
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3:
3/3: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps myapp | grep -i running
t36klvx3c1lp   myapp.1       nginx:latest   manager    Running         Running about an hour ago
d2t3lnt5i34m   myapp.2       nginx:latest   worker-2   Running         Running 2 hours ago
k6dod0l6w1qn   myapp.6       nginx:latest   worker-1   Running         Running 6 minutes ago
root@manager:/#
```

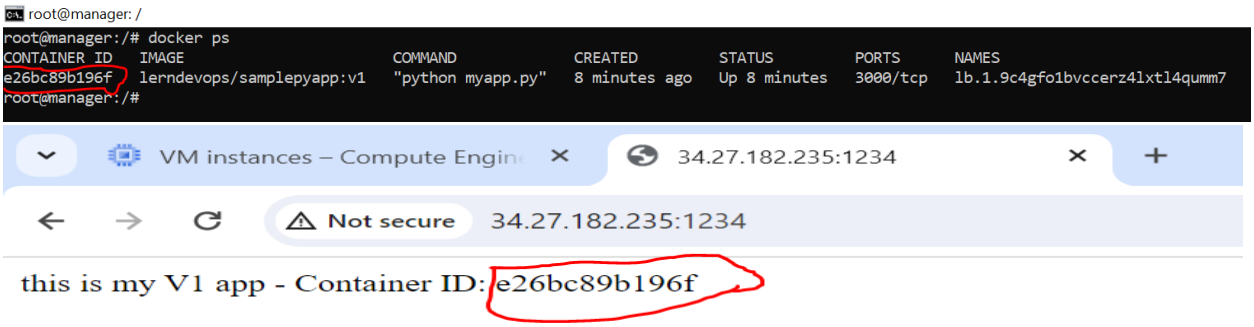
39) How can we achieve load balancing in docker service by using docker swarm?

- To check load balancing we need to use one sample application which will give you container id as output.
- Now let's create docker service using this application image.
- Syntax> docker service create --name <name of service> --replicas <number of replica> -p <new port>:<default application port> <image name>:<version of image>
- Ex> docker service create --name lb --replicas 3 -p 1234:3000 lerndevops/samplepyapp:v1
- Ex> docker service ps lb
- Ex> docker service ls

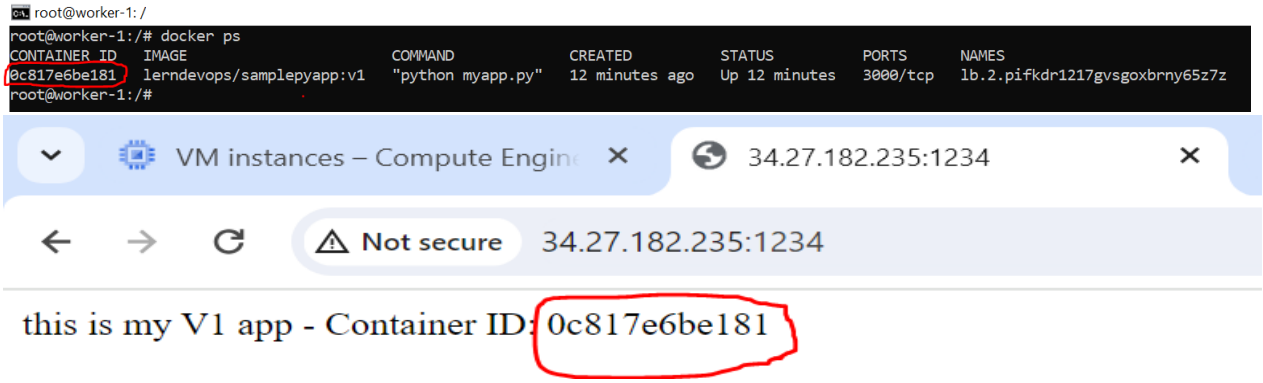
```
root@manager: /
root@manager:/# docker service create --name lb --replicas 3 -p 1234:3000 lerndevops/samplepyapp:v1
p3fsimdek5bram0l75i3oa52r
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps lb
ID             NAME      IMAGE                        NODE       DESIRED STATE   CURRENT STATE            ERROR     PORTS
9c4gfo1bvcce   lb.1      lerndevops/samplepyapp:v1    manager    Running         Running 13 seconds ago
pifkdr1217gv   lb.2      lerndevops/samplepyapp:v1    worker-1   Running         Running 14 seconds ago
vpeidgp8vf5u   lb.3      lerndevops/samplepyapp:v1    worker-2   Running         Running 14 seconds ago
root@manager:/# docker service  ls
ID             NAME      MODE         REPLICAS   IMAGE                        PORTS
p3fsimdek5br   lb        replicated   3/3        lerndevops/samplepyapp:v1    *:1234->3000/tcp
root@manager:/#
```

- We can access application by using any host (manager/worker1/worker2) IP i.e. (34.27.182.235/ 34.67.134.163/ 34.132.31.197).
- We also need service port to access application, docker service will create same port on all nodes manager/worker1/worker2 given during create service command.
- When we tried to hit application on browser multiple time we will get different container Id each time.
- Here docker service is acting as load balancer which will redirect request to each container.

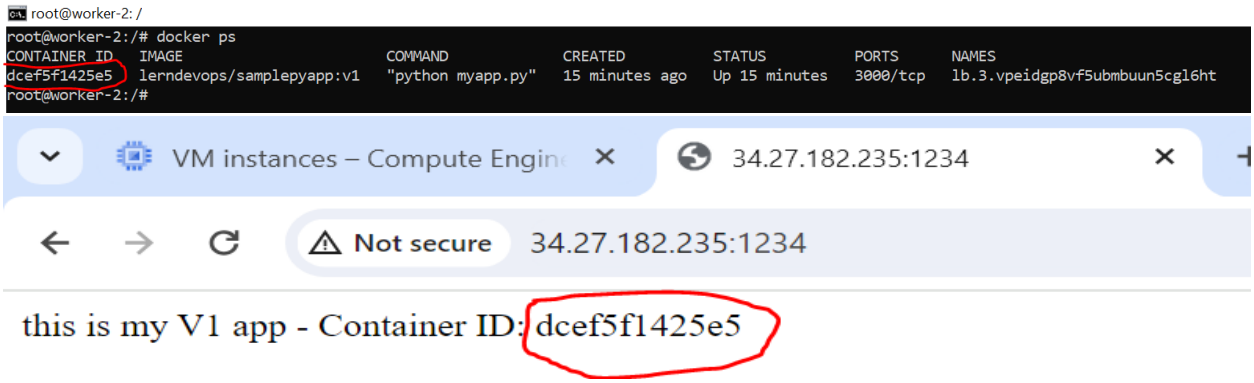
Manager Container ID



Worker1 Container ID



Worker2 Container ID



- We can test application by using below script on any host machine
- `Syntax>while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done`

```
root@worker-2: /
root@worker-2:/# docker ps
CONTAINER ID   IMAGE                      COMMAND             CREATED          STATUS          PORTS      NAMES
dcef5f1425e5   lerndevops/samplepyapp:v1  "python myapp.py"   15 minutes ago   Up 15 minutes   3000/tcp   lb.3.vpeidgp8vf5ubmbuun5cgl6ht
root@worker-2:/# while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: e26bc89b196f^C
```

- If we add any number of containers during run time, docker service will automatically add them in load balancing & redirecting request on newly added container.
- Suppose we want to add 2 more replica in existing app
- `Ex> docker service scale lb=5`

```
root@manager: /
root@manager:/# docker ps
CONTAINER ID   IMAGE                      COMMAND             CREATED         STATUS         PORTS      NAMES
e26bc89b196f   lerndevops/samplepyapp:v1  "python myapp.py"   8 minutes ago   Up 8 minutes   3000/tcp   lb.1.9c4gfo1bvccerz4lxtl4qumm7
root@manager:/# docker service scale lb=5
lb scaled to 5
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service converged
root@manager:/#
```

```
root@worker-2: /
root@worker-2:/# while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: 54f62205abd2
this is my V1 app - Container ID: 7e3694f04fe1
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: 54f62205abd2
this is my V1 app - Container ID: 7e3694f04fe1
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: 54f62205abd2
this is my V1 app - Container ID: 7e3694f04fe1
this is my V1 app - Container ID: e26bc89b196f
```

- Similarly if we reduce any number of containers during run time, docker service will automatically remove them in load balancing & redirecting request on active containers only.
- Suppose we want to keep only 2 replica running existing app
- `Ex> docker service scale lb=2`

```
root@manager:/# docker service scale lb=2
lb scaled to 2
overall progress: 2 out of 2 tasks
1/2: running   [==================================================>]
2/2: running   [==================================================>]
verify: Service converged
root@manager:/#
```

root@worker-2: /

```
root@worker-2:/# while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done
this is my V1 app - Container ID: 7e3694f04fe1
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: 54f62205abd2
this is my V1 app - Container ID: 7e3694f04fe1
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: dcef5f1425e5
this is my V1 app - Container ID: 54f62205abd2
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 0c817e6be181
this is my V1 app - Container ID: e26bc89b196f^C
root@worker-2:/#
```

40) How can we achieve application version update in docker service by using docker swarm?

- We can achieve application version update without down time, by slowly adding updated container in load balancer & removing old container, this concept is known as **rolling update**.
- We just need to update the image is application; remaining things will be handling by docker service only.
- Suppose we want to update new image of existing application **lb** by using below command
- `Syntax> docker service update <application name> --image <image name>:<new version>`
- `Ex:> docker service update lb --image lerndevops/samplepyapp:v2`

root@manager: /

```
root@manager:/# docker service update lb --image lerndevops/samplepyapp:v2
lb
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/#
```

root@worker-2: /

```
root@worker-2:/# while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done
this is my V1 app - Container ID: 5c0ceee109df
this is my V1 app - Container ID: 10ea1e021420
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 5c0ceee109df
this is my V1 app - Container ID: 10ea1e021420
this is my V1 app - Container ID: e26bc89b196f
this is my V1 app - Container ID: 5c0ceee109df
this is my V1 app - Container ID: 10ea1e021420
this is my V1 app - Container ID: 5c0ceee109df
this is my V1 app - Container ID: 10ea1e021420
this is my V2 app - Container ID: 69638381fd9e
this is my V1 app - Container ID: 5c0ceee109df
this is my V2 app - Container ID: 69638381fd9e
this is my V1 app - Container ID: 5c0ceee109df
this is my V2 app - Container ID: 72d28d45f70f
this is my V2 app - Container ID: 69638381fd9e
this is my V2 app - Container ID: 72d28d45f70f
this is my V2 app - Container ID: 69638381fd9e
this is my V2 app - Container ID: 6f21f04e53c1
this is my V2 app - Container ID: 72d28d45f70f
this is my V2 app - Container ID: 69638381fd9e^C
root@worker-2:/#
```

- Suppose updated new image of application **lb** is not working properly, we can rollback version by using below command on manager
- `Syntax> docker service rollback <application name>`
- `Ex> docker service rollback lb`

root@manager: /

```
root@manager:/# docker service rollback lb
lb
rollback: manually requested rollback
overall progress: rolling back update: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/#
```

root@worker-2: /

```
root@worker-2:/# while true; do curl http://34.132.31.197:1234/;sleep 1;echo "";done
this is my V2 app - Container ID: 6f21f04e53c1
this is my V2 app - Container ID: 72d28d45f70f
this is my V2 app - Container ID: 69638381fd9e
this is my V2 app - Container ID: 6f21f04e53c1
this is my V2 app - Container ID: 72d28d45f70f
this is my V2 app - Container ID: 69638381fd9e
this is my V2 app - Container ID: 6f21f04e53c1
this is my V2 app - Container ID: 69638381fd9e
this is my V1 app - Container ID: 731d4ab88cdb
this is my V2 app - Container ID: 6f21f04e53c1
this is my V1 app - Container ID: 731d4ab88cdb
this is my V2 app - Container ID: 6f21f04e53c1
this is my V1 app - Container ID: fb9d00103583
this is my V1 app - Container ID: 731d4ab88cdb
this is my V1 app - Container ID: fb9d00103583
this is my V1 app - Container ID: 731d4ab88cdb
```

41) What is global service mode in docker swarm? What is use of it?

- If we did not specified number of replicas in service definition, by default it will create only one replica of container, so if there is requirement to create default number of replica as many number of host we have, in that case we need to specify mode as global.
- So if we specify --mode as global it will create as many number on containers, as many we have host machines.
- We can use below command to create global service
- `Synatx> docker service crate --name <name of service> --mode global -p <new port>:<default port> <image>`
- `Ex> docker service crate --name test --mode global -p 3456:80 httpd`
- Here if we don't give replicas number, by default it will create as many number of replica as many host machine we have.
- Goble service is type of service in docker swarm, when you create as global service it dentifies number of host & create that many replicas of containers automatically.
- It will add remove replica based of number of host machines.
- This type of service is used to setup container which are used to monitor or do logging worker or manager nodes.

```
root@manager: /
root@manager:/# docker service create --name test --mode global -p 3456:80 httpd
g7de8d9qp08a4f2yh0byl1267
overall progress: 3 out of 3 tasks
n98axbfbcnuu: running   [==================================================>]
q9gybo1datpf: running   [==================================================>]
niv7hyaqszvs: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps test
ID             NAME                             IMAGE          NODE       DESIRED STATE   CURRENT STATE            ERROR     PORTS
w4kw1d892aj0   test.n98axbfbcnuup46giv32dodl7   httpd:latest   worker-2   Running         Running 17 seconds ago
ynvg5r8yzu0j   test.niv7hyaqszvs9799hyeag3var   httpd:latest   manager    Running         Running 17 seconds ago
xrfleupqtzzm   test.q9gybo1datpf5iafofdzarfhu   httpd:latest   worker-1   Running         Running 17 seconds ago
root@manager:/# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                          PORTS
p3fsimdek5br   lb        replicated   3/3        lerndevops/samplepyapp:v1      *:1234->3000/tcp
g7de8d9qp08a   test      global       3/3        httpd:latest                   *:3456->80/tcp
root@manager:/#
```

42) How to apply constraints to application?

- We can apply constraints to application so that it can run on either worker or manager node only.
- To apply constraints we can use below command on manager.
- `Syntax>docker service create --name <service name> --replicas <number of replica> --constraint <condition> -p <new port>:<default port> <image>:<version>`
- Command to apply constraints to run container on worker node only.
- `Ex> docker service create --name svc1 --replicas 3 --constraint node.Role==worker -p 1234:80 nginx`
- Command to apply constraints to run container on manager node only.
- `Ex> docker service create --name svc1 --replicas 3 --constraint node.Role==manager -p 1234:80 nginx`

```
root@manager: /
root@manager:/# docker service create --name svc1 --replicas 3 --constraint node.Role==worker -p 1234:80 nginx
qqg4pbkcvpjx24mdgs9f2ocsx
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps svc1
ID             NAME      IMAGE          NODE       DESIRED STATE   CURRENT STATE           ERROR     PORTS
ykm5pxydbka7   svc1.1    nginx:latest   worker-1   Running         Running 6 minutes ago
2ewf7cdq7dvd   svc1.2    nginx:latest   worker-2   Running         Running 6 minutes ago
2ymirfwmil1c   svc1.3    nginx:latest   worker-1   Running         Running 6 minutes ago
root@manager:/# docker service rm svc1
svc1
root@manager:/# docker service create --name svc1 --replicas 3 --constraint node.Role==manager -p 1234:80 nginx
1hi3muu5hc1xtzjcoj2qofr3z
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps svc1
ID             NAME      IMAGE          NODE      DESIRED STATE   CURRENT STATE           ERROR     PORTS
hu6t1lae1e1k   svc1.1    nginx:latest   manager   Running         Running 9 seconds ago
9pjxomz46h1p   svc1.2    nginx:latest   manager   Running         Running 9 seconds ago
d56q6rghoqpc   svc1.3    nginx:latest   manager   Running         Running 9 seconds ago
root@manager:/#
```

43) What is use of label? How can we apply label to node? How to apply constraints to application on labels?

- You can add pieces of metadata to your swarm nodes using node labels.
- You can then use these labels to determine which nodes tasks will run on.
- **Use Case:** With nodes in multiple datacenters or availability zones, you can use labels to specify which zone each node is in. Then, execute tasks in specific zones or distribute them evenly across zones.
- We can add label to node by using below command
- `Syntax:> docker node update --label-add LABEL=VALUE NODE`
- `Ex:> docker node update --label-add own=santosh worker-1`
- `Ex:> docker node update --label-add own=santosh q9gybo1datpf5iafofdzarfhu`
- We can inspect & check whether label is added or not by using below command

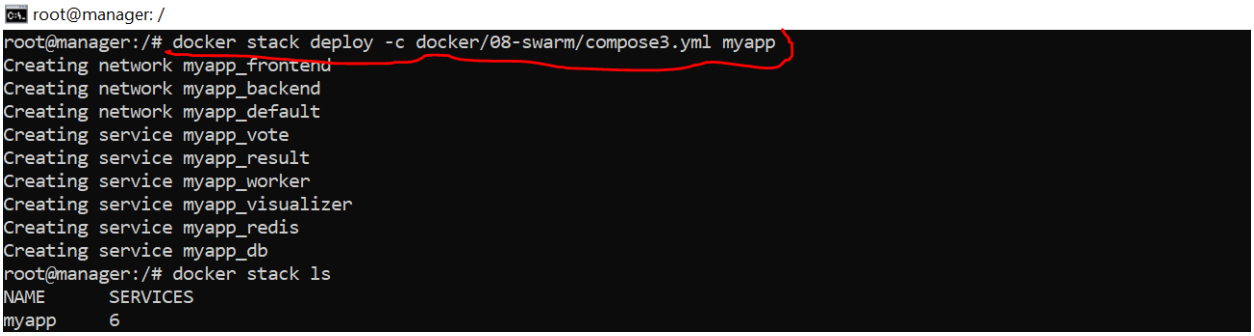
```
root@manager: /
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dod17     worker-2   Ready     Active                          24.0.7
root@manager:/# docker node update --label-add own=santosh worker-1
Error response from daemon: node worker-1 is ambiguous (2 matches found)
root@manager:/# docker node update --label-add own=santosh q9gybo1datpf5iafofdzarfhu
q9gybo1datpf5iafofdzarfhu
root@manager:/# docker inspect q9gybo1datpf5iafofdzarfhu
[
    {
        "ID": "q9gybo1datpf5iafofdzarfhu",
        "Version": {
            "Index": 577
        },
        "CreatedAt": "2023-12-19T02:24:51.938785463Z",
        "UpdatedAt": "2023-12-29T16:41:42.220313434Z",
        "Spec": {
            "Labels": {
                "own": "santosh"
            },
            "Role": "worker",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "worker-1",
            "Platform": {
                "Architecture": "x86_64",
                "OS": "linux"
            },
```

- We can use below command to create service to apply constraint base on label
- `Ex:> docker service create --name svc2 --replicas 3 --constraint node.labels.own==santosh -p 12345:80 nginx`
- `Ex:> docker service ps svc2`

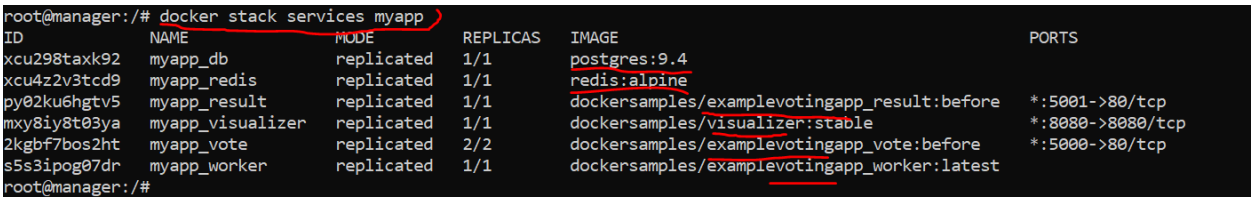
```
root@manager: /
root@manager:/# docker service create --name svc2 --replicas 3 --constraint node.labels.own==santosh -p 12345:80 nginx
qp1w2bxhi1hsyn2b5gxiimkm0
overall progress: 3 out of 3 tasks
1/3: running   [==================================================>]
2/3: running   [==================================================>]
3/3: running   [==================================================>]
verify: Service converged
root@manager:/# docker service ps svc2
ID             NAME      IMAGE          NODE       DESIRED STATE   CURRENT STATE            ERROR     PORTS
idfrnjfhxi3t   svc2.1    nginx:latest   worker-1   Running         Running 19 seconds ago
ouos7pmkxyk5   svc2.2    nginx:latest   worker-1   Running         Running 19 seconds ago
8wioa1iokcw8   svc2.3    nginx:latest   worker-1   Running         Running 19 seconds ago
root@manager:/#
```

44) What is docker stack?

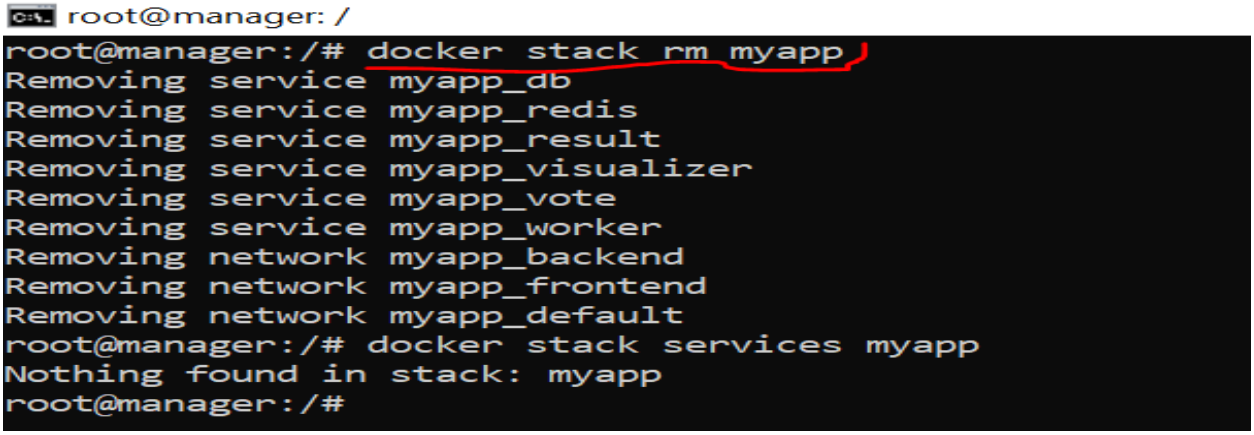
- Docker stack is nothing but set of services.
- A Stack is a collection of interrelated services that can be deployed and scaled as a unit
- We can create stack of different-different application by using YML file & below command
- Syntax:> `docker stack deploy -c <YML file> <app name>`
- Ex:> `docker stack deploy -c docker/08-swarm/compose3.yml myapp`



- We can check service running within stack using below command
- Syntax:> `docker stack services <app name>`
- Ex:> `docker stack services myapp`

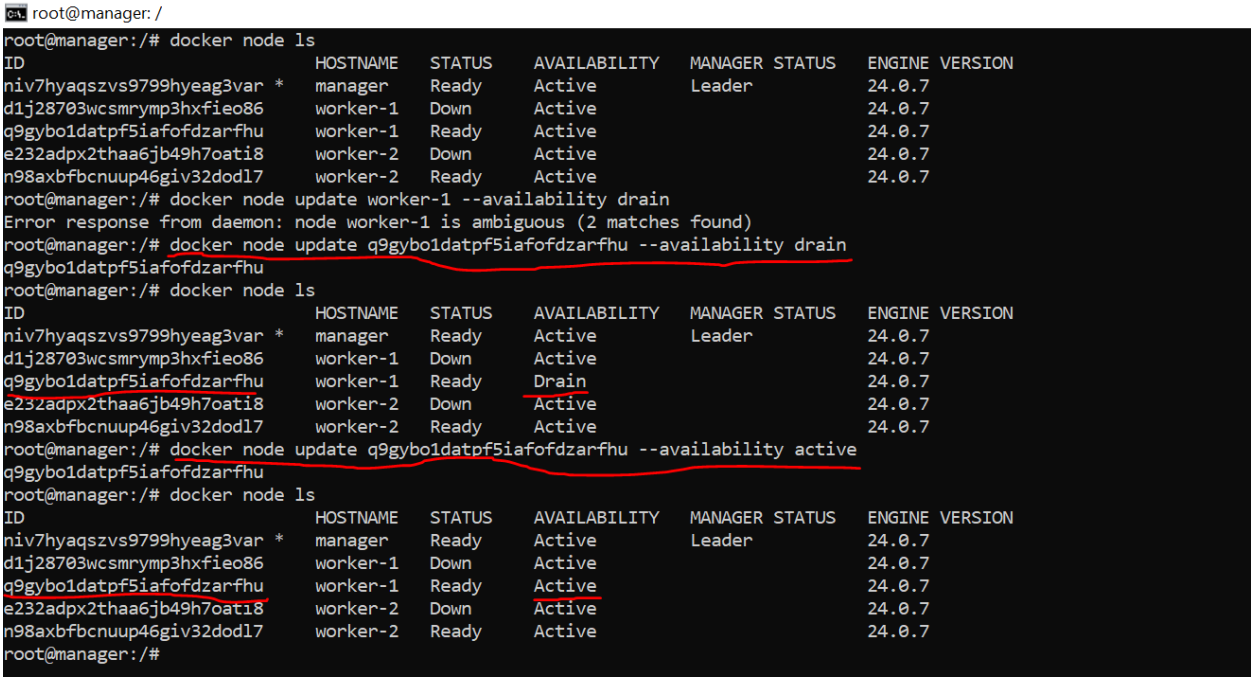


- We can remove entire application using below command
- Syntax:> `docker stack rm <app name>`
- Ex:> `docker stack rm myapp`



45) How we can manage worker node during maintenance activity?

- We can manage worker node during maintenance, by updating worker node's availability, as drain so that not container will run on this node, using below command
- `Syntax:> docker node update <worker node> --availability drain`
- `Ex:> docker node update q9gybo1datpf5iafofdzarfhu --availability drain`
- We can check availability using below command
- `Syntax:> docker node ls`
- We can change availability using below command
- `Syntax:> docker node update <worker node> --availability active`
- `Ex:> docker node update q9gybo1datpf5iafofdzarfhu --availability active`



46) How we can we promote & demote a node?

- We can promote a node as manager using below command
- `Syntax:> docker node promote <worker node>`
- `Ex:> docker node promote q9gybo1datpf5iafofdzarfhu`
- We can demote a node as worker using below command
- `Syntax:> docker node demote <worker node>`
- `Ex:> docker node demote q9gybo1datpf5iafofdzarfhu`

root@manager: /

```
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Ready     Active                          24.0.7
root@manager:/# docker node promote worker-1
Error response from daemon: node worker-1 is ambiguous (2 matches found)
root@manager:/# docker node promote q9gybo1datpf5iafofdzarfhu
Node q9gybo1datpf5iafofdzarfhu promoted to a manager in the swarm.
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active         Reachable        24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Ready     Active                          24.0.7
root@manager:/# docker node demote q9gybo1datpf5iafofdzarfhu
Manager q9gybo1datpf5iafofdzarfhu demoted in the swarm.
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Ready     Active                          24.0.7
root@manager:/#
```

46) How we worker can leave Swarm cluster?

- Worker can leave swarm cluster using below command
- `Syntax:>docker swarm leave`

root@worker-2: /home/devops

```
root@worker-2:/home/devops# docker swarm leave
Node left the swarm.
```

```
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Down      Active                          24.0.7
root@manager:/#
```

- Now we can remove worker-2 from cluster by using below command on mamager
- `Syntax:>docker node rm <worker node>`
- `Ex:> docker node rm <worker node>`

```
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
n98axbfbcnuup46giv32dodl7     worker-2   Down      Active                          24.0.7
root@manager:/# docker node rm worker-2
Error response from daemon: node worker-2 is ambiguous (2 matches found)
root@manager:/# docker node rm n98axbfbcnuup46giv32dodl7
n98axbfbcnuup46giv32dodl7
root@manager:/# docker node ls
ID                            HOSTNAME   STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
niv7hyaqszvs9799hyeag3var *   manager    Ready     Active         Leader           24.0.7
d1j28703wcsmrymp3hxfieo86     worker-1   Down      Active                          24.0.7
q9gybo1datpf5iafofdzarfhu     worker-1   Ready     Active                          24.0.7
e232adpx2thaa6jb49h7oati8     worker-2   Down      Active                          24.0.7
```

- We can remove all worker from swarm cluster in manager by using below command
- `Syntax> docker swarm leave -f`