DOCKER

Monolithic:

It is an Architecture. For all Services, if we use one server and one database we can called as monolithic

Eg: Ecommerce SAAS application (or) take Paytm App - movie tickets, bookings, etc.., these are called services

- If all these services are included in one server then it will be called monolithic architecture
- It is tight coupling i.e. the services highly dependent on each other

Drawback:

If one service is down means we have to shutdown entire application to solve that service. So, user is facing problems because it is tightly coupled

Microservice:

- If every service has its own individual servers then it is called microservices
- Every microservice architecture has its own database for each service
- Take same above example. For every service if we keep 1-database, 1-server it is microservice
- It is loose coupling

Drawback:

It is too cost, because we have to maintain so many servers and database. So, maintenance is high

- ☐ When compared microservice to monolithic. Microservice is good to use. Because, if one service is not working. So, we can work on it without shutdown the application. That's the reason microservices is good and preferable

Why Docker:

Let us assume that we are developing an application, and every application has Frontend, Backend and database

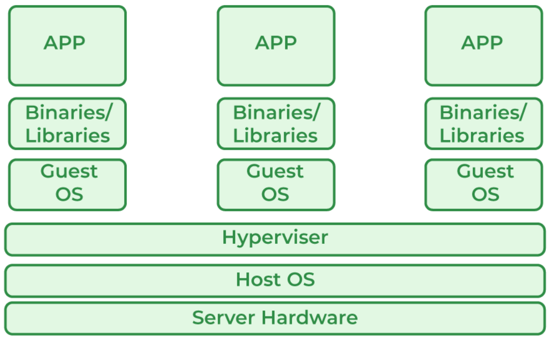
To overcome the above monolithic architecture we're using "Docker"

- So while creating the application we need to install the dependencies to run the code
- So, I installed Java11, reactJS and MongoDB to run the code. After sometime, I need another versions of java, react and MongoDB for my application to run the code
- So, it's really a hectic situation to maintain multiple versions of same tool in our system

To overcome this problem we will use "Virtualization"

Virtualization:

- It is used to create a virtual machines inside on our machine. In that virtual machines we can hosts guest OS in our machine
- By using this guest OS we can run multiple application on same machine



Virtualization

Virtualization Architecture

- Here, Host OS means our windows machine. Guest OS means virtual machine
- Hypervisor is also known as Virtual machine monitor (VMM). It is a component/software and it is used to create the virtual machines

Drawback:

- It is old method
- If we use multiple guest OS (or) Virtual machines then the system performance is low

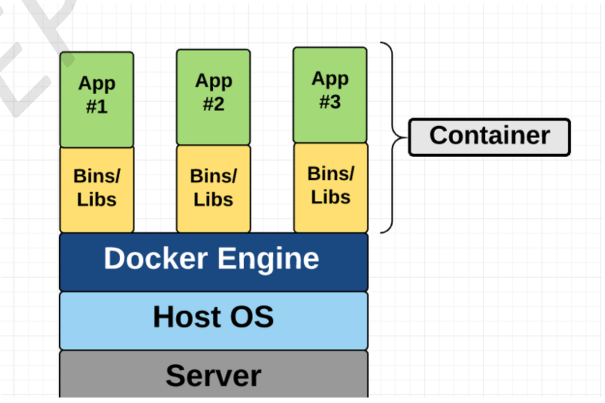
To overcome this virtualization, we are using "Containerization" ie., called Docker

Containerization:

It is used to pack the application along with it's dependencies to run the application. This process is called containerization

Container:

- It's the runtime of the application which is created through docker image
- Container is nothing but it is a virtual machine, which doesn't have any OS
- With the help of images container will run
- Docker is a tool. It is used to create the containers



Container Architecture

- It is similar to virtualization architecture, instead od hypervisor we are having Docker Engine
- Through Docker Engine we're creating the containers
- Inside the container we're having the application
- Docker Engine - The software that hosts the container
- In one container we can keep only image

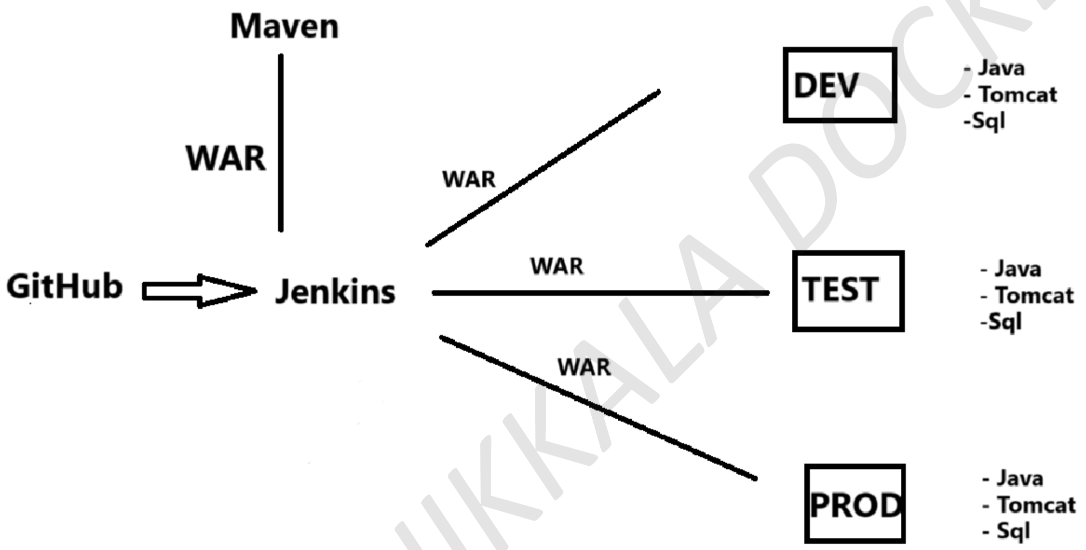
Docker Image:

- A Docker image is a file used to execute code in a Docker container.
- Docker images act as a set of instructions to build a Docker container, like a template.
- Docker images also act as the starting point when using Docker.

(or)

- It is a template that contains applications, bin/libs, configs, etc., packaged together

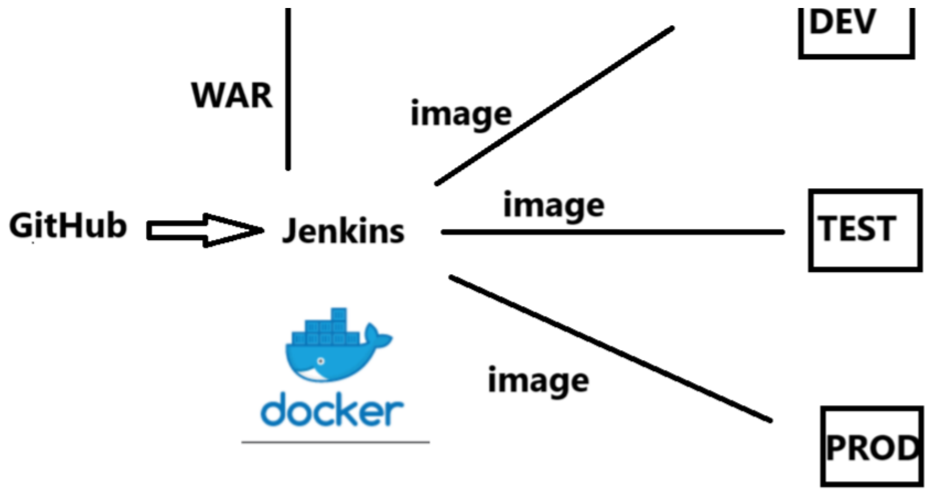
Before Docker:



- First, get the code from the GitHub and integrate with Jenkins
- Integrate maven with Jenkins. So, we get War file
- So, that war file we have to deploy in different environments
- So, if you want to deploy war file/application we have to install the dependencies

This is the process when docker is not there

After Docker:



- First, get the code from the GitHub and integrate with Jenkins
- Integrate maven with Jenkins. So, we get War file
- Here, we're not going to install dependencies on any server. Because, we're following containerization rule
- So, here we're creating image i.e. image is the combination of application and dependencies
    - image = war + Java+Tomcat+MySql
- Now, in this image application and dependencies present. So, overall this process is called containerization
- So, whenever if you want to run your application.
    - Run that image in the particular environment. No need to install again dependencies. Because these are already present in image
- So, after run the image. In that server that applications and dependencies installed
- When we run an image, container will gets created. Inside the container we're having application
- This images are already prebuilted by docker
- Container is independent in AMI i.e. If we launch AMI in ubuntu (or) CentOS, any other OS. this container will work

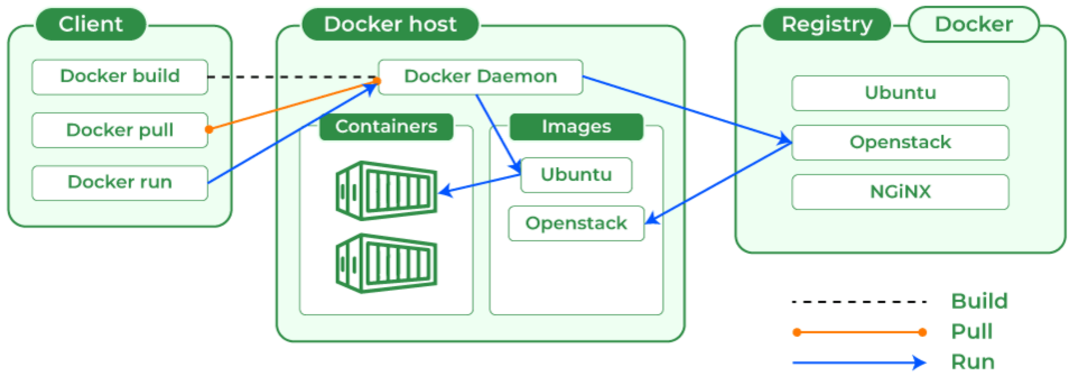
So, overall after docker, In any environment no need to install dependencies. We can just run the images in that particular environment. If container is created means application created

Docker

- It is an open source centralized platform designed to create, deploy and run applications
- Docker is written in the GO language
- Docker uses containers on host OS to run applications. It allows applications to use the same linux kernel as system on the host computer, rather than creating a whole virtual OS
- Docker is platform independent. i.e. we can install docker on any OS but the "docker engine" runs natively on "Linux" distribution
- Docker performs OS level virtualization also known as containerization
- Before Docker, many users face problems that a particular code is running in the developers system but not in the user system
- Docker is a set of PAAS that use OS level virtualization, where as VMware uses hardware level virtualization

- Container have OS files but it's negligible in size compared to original files of that OS

Docker Architecture:



We are having 4 components

1. Docker Client
   a. It is a primary way that many docker users interact with docker. When you use commands such as docker run, the client sends these commands to docker daemon, which carries them out
   b. The docker commands uses the docker API
   c. Overall, here we perform the commands
2. Docker Host
   a. It contains containers, images, volumes, networks
   b. It is also a server, where we install the docker in a system
3. Docker Daemon
   a. Docker daemon runs on the host OS.
   b. It is responsible for running containers to manage docker services
   c. Docker daemon communicates with other daemons
   d. It offers various docker objects such as images, containers, networking and storage
4. Docker Registry
   a. A Docker registry is a scalable open-source storage and distribution system for docker images
   b. It is used for storing and sharing the images
   c. Eg: For git we had GitHub. Same like for docker we had Docker registry

Advantages of Docker:

- Caching a cluster of containers
- Flexible resources sharing
- Scalability - Many containers can be placed in a single host
- Running your service on network that is much cheaper than standard servers

Note: [points to be followed when you're using Docker practically]

1. You can't use directly, you need to start/restart first (observe the docker version before and after restart) (Just like Jenkins)
2. You need a base image for creating a container
3. You can't enter directly into container, you need to start first
4. If you run an image, By default one container will create

5. Docker client and Host - Both are in same server

Launch normal server in AWS EC2 - t2.micro, normal-sg, 8GB volume

- Install Docker in server
  - yum install docker -y
- Check the version
  - docker --version (or) docker -v
- If we want Client details
  - docker version (or) docker info
- Check the docker is running/not
  - systemctl status docker (or) service docker status
- Start the docker
  - systemctl start docker (or) service docker start

  Now, again perform → docker version (or) docker info

  - Now, we got server details. When the docker is in running state we can see server details. ie., when the daemon is in running state we can see the server details. Daemon is not running means we can see the client details

Create Docker Image

- Checking how many images are present
  - docker images
- Create an Image
  - docker run ubuntu (or) docker pull image
    - Here, ubuntu is an image. Through this image if we create a container, that container will be run in ubuntu OS
- See the list of images
  - docker images
- To get the image count
  - docker image/wc -l

Create Docker Container

- Checking how many containers are present
  - docker ps -a (or) docker container ls -a
    - Here, default containers will present, whenever we create a image automatically default containers will created
    - Here, ps means process status
- Create own/custom container
  - docker run -it --name cont-1 ubuntu
    - Here, '-it' is interactive terminal, used to execute the commands
    - Here, cont-1 is container name and ubuntu is imagename
    - When you perform the command, you will get 'root@containerID' terminal. That means we are inside the container
    - if we give 'll', we will get container default files
    - If you type exit, you are coming out from the container
  - docker ps -a

        - you will see the list of containers, but all containers are in exited state
        - So, here default and normal containers are present.
- The main difference between is default and normal containers is, if you run the default containers also it will be in exited state. Because it doesn't have '-it'
- Going inside the container
    - docker attach cont-1
        - we got the container terminal, now if you type 'exit' we're came out from the terminal and our container is in exited state.
        - So, Without direct exiting the status how to do normal exit. So, if we do normal exit
        - Just ctrl+p,q in the terminal
- Start the container
    - docker start cont-1
    - docker ps -a
    - docker attach cont-1
        - Now, I want to do exit but not exit state
        - So, perform ctrl+p,q. If you perform means exit from that user not from the state
    - docker ps -a
- Stop the container
    - docker stop cont-1

Note: If you want to go from exit state to running state, you have to start the container

- See the running containers
    - docker ps
- Delete the container
    - docker rm containerName/containerID

Note: We can't delete the running containers. First, we have to stop the containers, then we can delete

- Delete multiple containers at a time
    - docker rm $(docker ps -a -q)
        - Here, a → all and q → id of the container
- Stop multiple containers
    - docker stop $(docker ps -aq)
        - If we do restart the docker, all containers will be in exited state
            - systemctl restart docker
- Delete the images
    - docker rmi imageName
- Delete multiple images
    - docker rmi $(docker images)
        - Sometimes, some images won't delete. Because they're running. So, stop the containers and delete the images
- Delete unused images
    - docker image prune
        - the image contains 'none' we can simply called as unused images
- Delete unused containers
    - docker container prune
        - It is used to remove the unused containers (or) unwanted thing means exited state
        - If the containers are in exited state it all deleted

- Rename the container name
  - docker rename oldName newname
- See the Latest created containers
  - docker container ls -n 2
    - that means we can see the latest 2 containers
  - docker container ls -n --latest
    - that means we can see the latest single containers
- See the Container ID's
  - docker container ls -a -q (or) docker ps -a -q
- See the running container ID's
  - docker container ls -q (or) docker ps -q
- See the container size
  - docker ps -a -s (or) docker container ls -a -s
- Delete unwanted/unused images, containers, networks, volumes at a time
  - docker system prune

Deploying Web Server in Docker

HTTPD

- If you want to deploy a web application we have to take either HTTPD (or) NGINX image
- If we run (or) pull the image. So, that image will be downloaded in local
- So, whenever you run the image, container created. Inside the container we are having web application

1. Create the HTTPD image
   - docker pull httpd
   - docker images
2. Now, Create the container by using images
   - docker run -itd --name cont1 -p 8081:80 httpd
     - Here, 8081 - host port, we can give any number
     - 80 → it is container port
     - So, here container port is change depends upon the image. So, if it's Apache it's 8080
     - So, here we are accessing the application through host port
     - Here, -d is used for detach mode. means it will run in foreground+background
       - If we don't give '-d' we can't enter into container directly
       - usually, we give '-it' directly we go into container. but here '-itd' we can't enter into container

Now, After running the container. Copy the public Ip:8081, you can access the httpd application

Now, if I want to access into the container, we can't perform the 'docker attach' command. Because we're using '-d'.

exec : It is a command used to perform inside the container without going inside the container in detach mode

So, let's start the container → docker start cont1

So, now I want to see the whole files inside the container

Syntax: docker exec containerName "commands"

- See the list of files in a container → docker exec cont1 "ls"
- Create the file → docker exec cont1 touch file

Above, we're performing the commands outside the container, but we are not going inside.

So, how we can go inside a container through detach mode

  - docker exec -it containerName /bin/bash
    - Here, /bin/bash is docker default path
    - So, here by default it work in ubuntu image

After performing the above command, we're inside a container

  - apt update -y
  - If I want to use any thing we have to install
  - apt install vim -y
  - vim index.html → it works

So, here it won't work, for that one we have to use the docker file

Inspect

Through inspect we can see the container full information like source code, etc..,

  - docker inspect containerName/ContainerID

we can check the particular information in inspect. For that, we can use "grep"

  - docker inspect containerName | grep -i "wordName"
    - docker inspect cont1 | grep -i id

Curl

Here, curl means Check URL. Using curl, we are checking the network connections

  - curl publicIp:8080
    - That means, if app is running means we can't check in browser. We can check directly in the server
    - So, curl tells whether the application is running/not

Limitations to the container

Here, CPU allocate (or) Memory allocate to the container, we can called container limits.

Generally, we took t2.micro i.e. 1 CPU, 1GB ram. So, in overall server we have 1 CPU & 1 GB ram

So, right now for server inside created containers, I want to provide 0.25% CPU, 250 MB of ram. So, these we called as limitations

Note: we have to mention that limits when you're performing a command

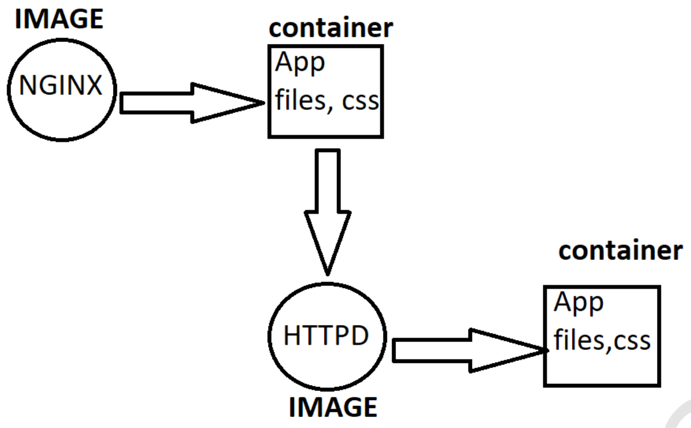
- docker run -itd --name cont1 -p 8084:80 --memory=250m --cpus="0.25" httpd

Now, check the limits whether it's applied/not. So, for that we have to do inspect the container

- docker inspect cont1 | grep -i memory

Container to Image Creation

So, upto now we're creating the container through the image. But, right now we're creating the image through the container



So, in above diagram, we're creating the container from nginx image. So, we need the same container again. Generally, we're creating another image and we created the container

But, Instead of above process. Here, In container already files are present. So, if we create the image from that container. We will get the same files in that image

Now, Instead of creating the separate container. Already, through Httpd image if we run means that all files directly came to another container.

i.e. automatically same application deployed

  - docker commit containerName NewImageName
- Create the image from the container
  - docker commit cont1 httpd
- Check the images
  - docker images
- Through that image create the container
  - docker run -itd --name cont2 -p 8085:80 httpd

So, Overall we're creating the images in 2 ways

1. Command

2. Docker file

Docker file

- It is basically a text file which contains set of instructions (or) commands
- To create the docker images, we're using Docker file
- Here, we're not having multiple Docker files
- i.e. For single directory we're having single Docker file
- In Docker file the first letter should be 'D'
- In Docker file we're having components
- And start components also be capital letter
  - Here, this is not mandatory. But official/formal way looks means we have to maintain capital letters

How it works :

- First, create a Docker file
- In this file, we're writing some commands
- If we build the Docker file, we got one image
- If we run the image, container will create
- i.e. Application started/running

Docker file Components

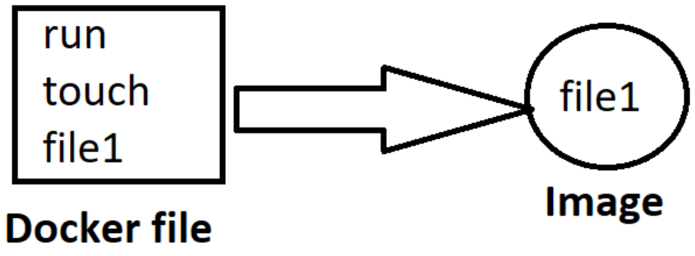
1. FROM
   - This is the 1st component in the Docker file, which is used to give/defined that images (HTTPD,NGINX,UBUNTU)
2. LABEL (or) MAINTAINER
   - We can give Author details i.e. we're mentioning the author name who wrote the Docker file
3. RUN
   - It is used to execute the commands, while we build the image
4. COPY
   - It is used to copy the files from server to container
5. ADD
   - It is also used to copy the files from server to container. But, it will also download the files from the internet. Eg: (targz, zip) and send to the container
6. EXPOSE
   - It is used to publish the port numbers. It is only used for documentation purpose
7. WORKDIR
   - It is used to create a directory and we will directly go into the particular directory/folder
   - i.e. inside the containers we're having so many folders. Particularly, create a folder means use workdir. Eg: /folder
8. CMD
   - It is also used to execute the commands
9. ENTRYPOINT
   - It is also used to execute the commands
10. ENV
    - It is used to assign/declare variables. Here, we can't override the values in runtime
11. ARG

- It is also used to assign/declare variables. Here, we can override the values in runtime

Difference between RUN & CMD

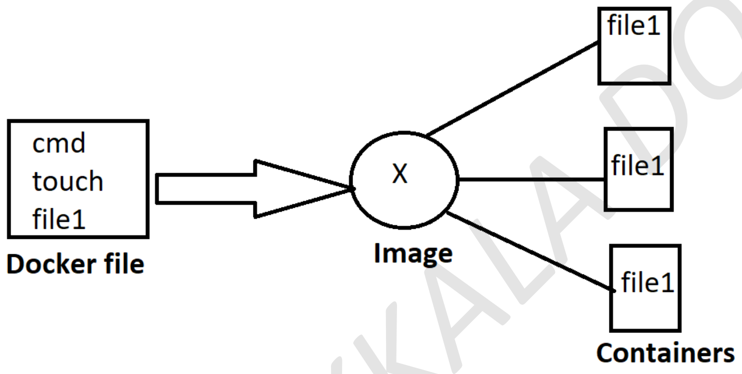
For RUN

- When we build the Docker file through 'RUN' . We're getting one image. This image contains the data



For CMD

- When we build the Docker file, We're getting one image. But this image doesn't contain any data
- But, when we run that image. We will get containers. In that containers we're having the data



Difference between CMD and ENTRYPOINT

- In Docker file if we give CMD, ENTRYPOINT at a time. The 1st preference goes to entrypoint. i.e. Entrypoint is having high priority
- Entrypoint values will overrides the commands/values in CMD

Eg: If we give 'git' in CMD. And in ENTRYPOINT you give 'maven'. So, Docker file chose maven

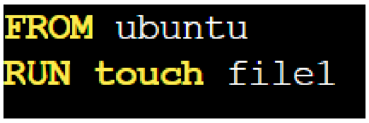
Docker file Practice

- [ ] Creating a file inside from Docker file

Step-1 : Creating the Docker file

- vim Dockerfile

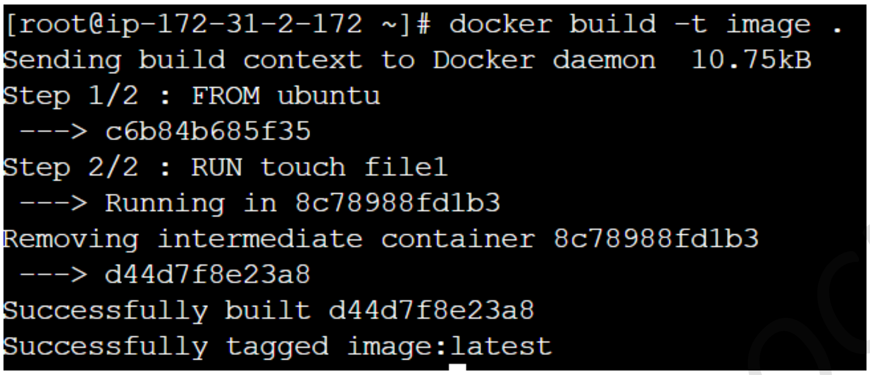
Step-2 : Write the basic code inside Docker file



```
FROM ubuntu
RUN touch file1
```

Step-3 : Build the Docker file

- docker build -t AnyImageName .
  - docker build -t sandeep .
  - Here '.' represents the path of the Docker file. i.e. Right now, Docker file is in current directory. So, we gave '.' Otherwise, we give different path
- After successfully build, you will get like this in the given below image



```
[root@ip-172-31-2-172 ~]# docker build -t image .
Sending build context to Docker daemon  10.75kB
Step 1/2 : FROM ubuntu
 ---> c6b84b685f35
Step 2/2 : RUN touch file1
 ---> Running in 8c78988fd1b3
Removing intermediate container 8c78988fd1b3
 ---> d44d7f8e23a8
Successfully built d44d7f8e23a8
Successfully tagged image:latest
```

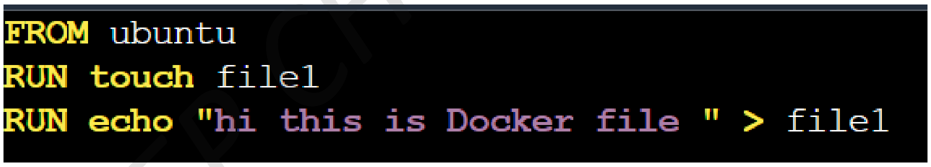
- Now, we get a image name called 'sandy'

Step-4 : Now, create the container from our created Image

- docker build -it --name cont-4 sandy
- Now, inside the container, perform 'll' command. You will get file1

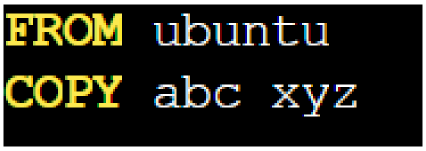
This is the way, we can create the Docker file and through that Docker file we can create images. Through that images we can create containers.

- [ ] Insert a data into a Docker file



```
FROM ubuntu
RUN touch file1
RUN echo "hi this is Docker file " > file1
```

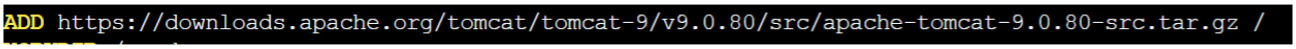
- [ ] Using COPY and ADD in a Docker file



```
FROM ubuntu
COPY abc xyz
```

In this copy first file name is from server file and 2nd file name is from container

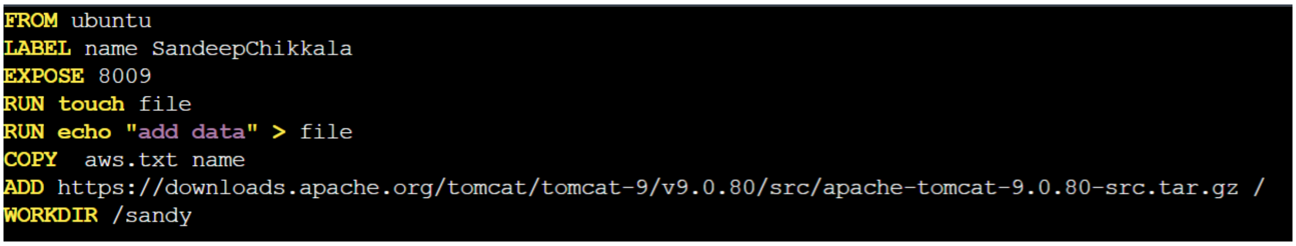
So, here we're copying server file to inside the containers



```
ADD https://downloads.apache.org/tomcat/tomcat-9/v9.0.80/src/apache-tomcat-9.0.80-src.tar.gz /
```

Add is also same like COPY but here we're downloading files from internet and copy into a container

☐ Perfect Docker file



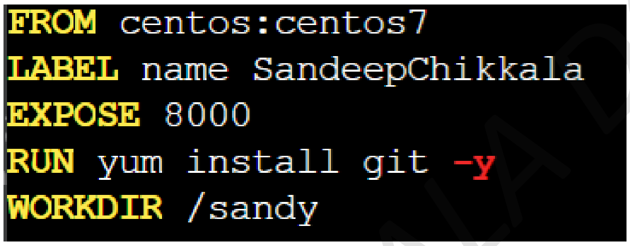
```
FROM ubuntu
LABEL name SandeepChikkala
EXPOSE 8009
RUN touch file
RUN echo "add data" > file
COPY  aws.txt name
ADD https://downloads.apache.org/tomcat/tomcat-9/v9.0.80/src/apache-tomcat-9.0.80-src.tar.gz /
WORKDIR /sandy
```

☐ Differences among RUN, CMD and ENTRYPOINT in Docker file

1st difference

When you're using RUN, you can directly build a docker file. i.e. when you perform

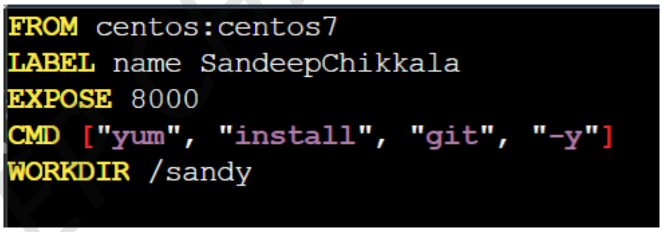
- docker build -t image .
  - Git will installed
- Through this RUN we can't install multiple tools like
  - docker run image tree httpd



```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
RUN yum install git -y
WORKDIR /sandy
```

☐ When you're using CMD, you have to build the docker file. But the package is not installed. So, you have to run the image. Then the git package is installed

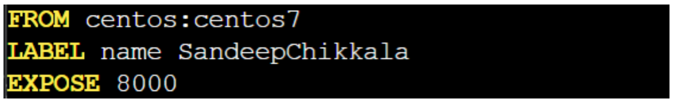
- docker build -t image .
- docker run image



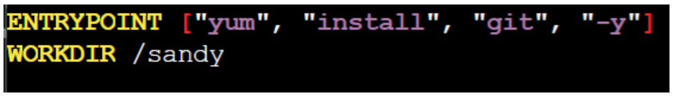
```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
CMD ["yum", "install", "git", "-y"]
WORKDIR /sandy
```

☐ When you're using ENTRYPOINT, you have to build the docker file. But the package is not installed. So, you have to run the image. Then the git package is installed

- docker build -t image .
- docker run image

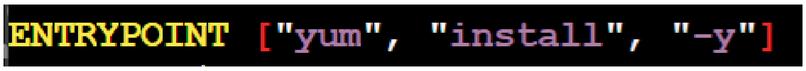


```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
```



```
ENTRYPOINT ["yum", "install", "git", "-y"]
WORKDIR /sandy
```

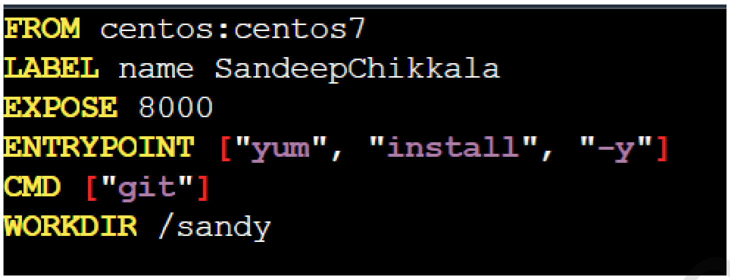
- Here, When you're using ENTRYPOINT, we can install multiple tools in runtime
    - docker run image tree httpd
- If we write code like this



```
ENTRYPOINT ["yum", "install", "-y"]
```

    - Here, command → docker run image → it won't work
    - Perform → docker run image java httpd → it works
        - that means, in Entrypoint, if you don't give any name also, In run time we can install so many tools
        - But in CMD it's not possible

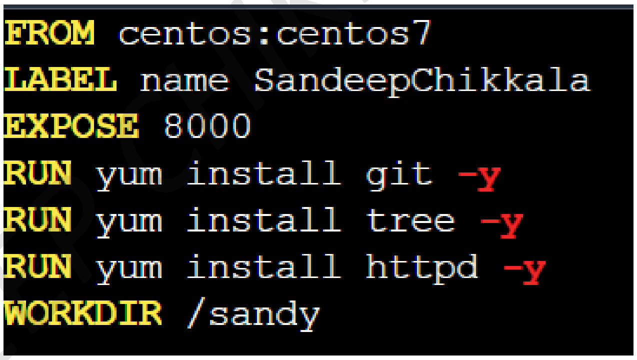
2nd difference :



```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
ENTRYPOINT ["yum", "install", "-y"]
CMD ["git"]
WORKDIR /sandy
```

- docker build -t image . → showing successfully builded
- docker run image
    - here, git installed because In entrypoint we didn't give anything. So, default we gave 'git' in CMD it will take git
- Now, I will give 'httpd' in run time i.e.
    - docker run image httpd
    - Now, only httpd installed. Because it overrides CMD, because entrypoint is having high priority

3rd Difference



```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
RUN yum install git -y
RUN yum install tree -y
RUN yum install httpd -y
WORKDIR /sandy
```

If we build means, at a time all tools installed

Now, try this code instead of 'RUN' use 'CMD' & 'ENTRYPOINT'

- Here, multiple cmd's and entrypoint not support

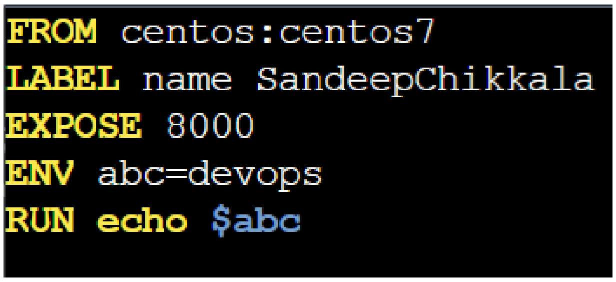
- So, multiple tools at a time if we want to run means we have to use 'RUN'

Overall, this is the main differences between RUN, CMD & ENTRYPOINT

- [ ] Difference between ARG & ENV in Docker file

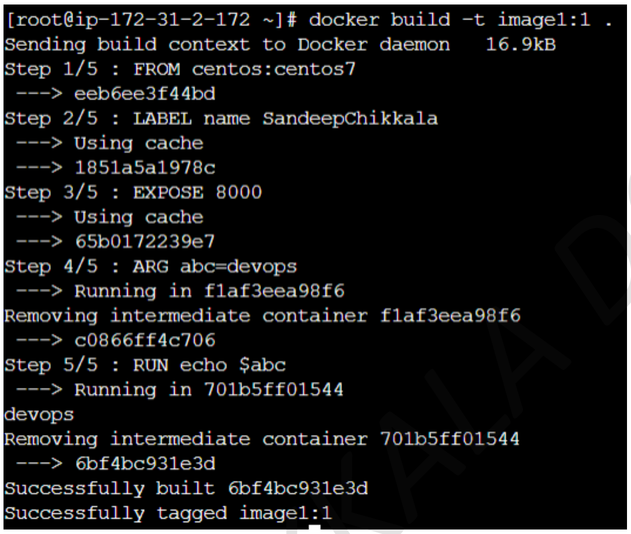
ENV :

Using ENV we can't override the value. When we build the Docker file, we can see the output in command line



```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
ENV abc=devops
RUN echo $abc
```

Output :



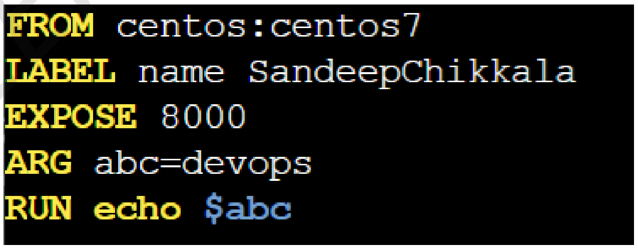
```
[root@ip-172-31-2-172 ~]# docker build -t image1:1 .
Sending build context to Docker daemon   16.9kB
Step 1/5 : FROM centos:centos7
 ---> eeb6ee3f44bd
Step 2/5 : LABEL name SandeepChikkala
 ---> Using cache
 ---> 1851a5a1978c
Step 3/5 : EXPOSE 8000
 ---> Using cache
 ---> 65b0172239e7
Step 4/5 : ARG abc=devops
 ---> Running in f1af3eea98f6
Removing intermediate container f1af3eea98f6
 ---> c0866ff4c706
Step 5/5 : RUN echo $abc
 ---> Running in 701b5ff01544
devops
Removing intermediate container 701b5ff01544
 ---> 6bf4bc931e3d
Successfully built 6bf4bc931e3d
Successfully tagged image1:1
```

ARG :

Using ARG we can override the value in runtime

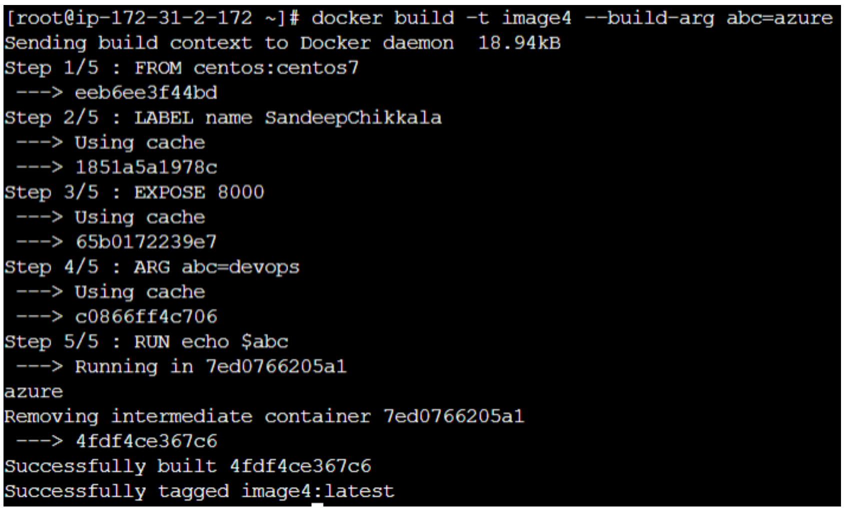
So, when we build the Docker file on that time only we can change

- docker build -t image --build-arg abc=azure



```
FROM centos:centos7
LABEL name SandeepChikkala
EXPOSE 8000
ARG abc=devops
RUN echo $abc
```

Output :



```
[root@ip-172-31-2-172 ~]# docker build -t image4 --build-arg abc=azure
Sending build context to Docker daemon  18.94kB
Step 1/5 : FROM centos:centos7
 ---> eeb6ee3f44bd
Step 2/5 : LABEL name SandeepChikkala
 ---> Using cache
 ---> 1851a5a1978c
Step 3/5 : EXPOSE 8000
 ---> Using cache
 ---> 65b0172239e7
Step 4/5 : ARG abc=devops
 ---> Using cache
 ---> c0866ff4c706
Step 5/5 : RUN echo $abc
 ---> Running in 7ed0766205a1
azure
Removing intermediate container 7ed0766205a1
 ---> 4fdf4ce367c6
Successfully built 4fdf4ce367c6
Successfully tagged image4:latest
```

Apply Tags in Images

- If we don't want to overriding the image. i.e. I want to see the same name for old image and new image. Without overriding
- Because, if we creating image with same name that image will override. So, we lose that data.
- So, if we use tags means our image will not override
- Almost, we will use tags when we build a Dockerfile
- Command is
  - docker build -t image:1 .
  - docker build -t image1:2 .

DOCKER VOLUMES

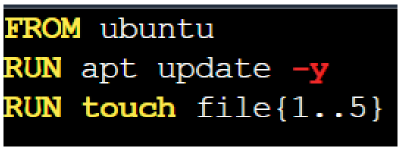
- **Volumes** are a mechanism for storing data outside containers.
- Docker volumes **provide persistent storage for your containers**.
- Docker manages the data in your volumes separately to your containers.
- All volumes are managed by Docker and stored in a dedicated directory on your host, usually /var/lib/docker/volumes for Linux systems.

If we update some data inside a container means, I want to get the same update data inside another container automatically.

So, for that purpose we're using docker volumes

Eg:

- vim Dockerfile



```
FROM ubuntu
RUN apt update -y
RUN touch file{1..5}
```

- Build the Dockerfile → docker build -t image .
- Create the container → docker run -it --name cont image → ll → we got files

- Now, create some files inside container → docker attach cont → touch a b c
- Now, create an image from the container. So, perform → docker commit cont image1
- Again create the container → docker run -it --name cont1 image1 → I got all files
- Now, again I created some files inside the cont1, I want that files in "cont". Usually we won't get

So, if you want to get that replication means we are using concept docker volumes

DIFFERENCE BETWEEN VOLUMES AND DIRECTORY

VOLUMES

- If there is any data present in the volume we can share to any other containers
- Volume will not gets deleted

DIRECTORY

- We can't share this data to another container
- directory will gets deleted

Points to be noted:

- When we create a container then volume will be created
- Volume is imply a directory inside our container
- First, we have to declare the directory volume and then share the volume
- Even if we stop/delete the container still, we can access the volume and inside the volume data
- You can declare directory as a volume, only while creating container
- We can't create volume from existing containers
- You can share one volume across many number of containers. At a time of creating a container not for existing container
- Volume will not be included, when you update an image
    - i.e. when you update the image, volume data will not update. Just it will show name volume
- If container-1 volume is shared to container-2 the changes made by container-2 will be also available in the container-1
    - i.e. In two containers, if we are having same volume. So, if we update the data in one container. automatically in another container also data got updated
- We can share our volume among different containers
- Decoupling/remove container from storage
- We can map volume in two ways
    a. container → container
    b. Host → container

We can create the Volumes in 2 ways

1. Command
2. Dockerfile

COMMAND (container → container)

- docker run -it --name containerName -v /volumeName imagename
    - docker run -it --name cont1 -v /sandy ubuntu → ll → cd sandy → touch file{1..5} → ctrl+p,q
- Now, we have to share the data. Right now, I have data in "cont1", we need to share in "cont2"

- command is → docker run -it --name NewContainerName --privileged=true --volumes-from VolumeContainerName imagename
  - docker run -it --name cont2 --privileged=true --volumes-from cont1 ubuntu
  - Here, --privileged=true means we are sharing the volume
  - Now check the files inside container → ll → cd sandy → we have files
  - Now, create files in the "cont2" and check in "cont1". So, you will get the data. that means data is replicated

How we can access the data, if the container is deleted/stopped ?

- First inspect the container → docker inspect cont1
  - Check mount, and here you see the volume path
    - cd /var/lib/docker/volumes......./data
  - Now, go to cd /var/lib/docker/volumes and see the list of volumes → ll
  - Here, we are having some big folder name → cd foldername → cd data → ll → here, we have our files

Note :

- If we deleted a container, but if we create a files inside volume "/data". Automatically in another container also changes happened, when it uses the same volume
- i.e. from local also, we can access volume data

CREATING MULTIPLE VOLUMES IN A CONTAINER

- docker run -it --name cont3 --privileged=true --volumes-from cont2 -v /sandy ubuntu
  - So, here In cont3, we get a volume from cont2 and another volume we created
  - So, like this we can maintain multiple volumes in a container

Mapping from HOST → CONTAINER

We can create a volume inside a container from server/local

Eg-1:

- docker run -it --name cont -v /home/ec2-user:/sandy --privileged=true ubuntu
  - here, /home/ec2-user is the server path
- now, Go to the path → ll → cd sandy → touch file
- Now, check in server
  - docker attach cont → ll → cd sandy → you have data

So, this process is called mounting the volumes

CREATING MANUAL VOLUMES THROUGH COMMANDS

- Create a manual/own volume → docker volume create volumeName
- Check how many volumes → docker volume ls
- Remove the volume
  - First, stop the container → docker stop $(docker ps -a -q)
  - docker volume rm volumeName
- Remove all volumes at a time

    - docker volume rm $(docker volume ls)
- Remove unused volumes
    - docker volume prune → it is asking (y/n) → type y

HOW TO ATTACH THE VOLUME TO CONTAINER FROM OUR MANUAL VOLUMES

1. docker volume create sandy
2. docker volume ls

- Now, this volume we have to attach to a container
    - docker run -it --name cont1 -v sandy:/volumeName ubuntu
    - ls → cd volumeName → touch file
    - exit

3. Go to docker default path → cd /var/lib/docker/volume/_data → ll → data is present

So, here overall we created volume separately and and that volume we attached to a container

We can't attach to the volume to existing containers. So, we call it as "base voice"

HOW TO ATTACH THE MANUAL VOLUMES TO HOST → CONTAINER

- docker run -it --name cont -v /home/ec2-user:/volumeName -v manualVolume:/volumeName ubuntu
    - docker run -it --name cont -v /home/ec2-user:/vol1 -v sandy/vol2 ubuntu
    - ll → so, we're having 2 volumes name as vol1, vol2

2. CREATING VOLUME FROM DOCKER FILE

- vi Dockerfile
        - FROM ubuntu
        - VOLUME ["/Sandeep"]
        - VOLUME ["/Chikkala"]
    - save and exit from the Dockerfile
- Build the Dockerfile
    - docker build -t image .
    - Now, in this image we are having 2 volumes. When we run this image we get the volumes inside a container
- Creating container
    - docker run -it --name cont image → ll → we are having Sandeep and Chikkala volumes

DOCKER NETWORKS

Docker Network is used to make a communication between the multiple containers that are running on same (or) different docker hosts

Why Network ?

Let's assume we are having 2 containers like APP and DB container. This App container has to communicate with DB container. So, the developer will write a code to connect the application to the DB container.

So, here the IP address of a container is not permanent. If a container is removed due to hardware failure, a new container will be created with a new IP, which can cause connection issues

To resolve this issue, we are creating our own network. i.e. we are using docker networks to create our custom/own network

Practice - Deep Dive

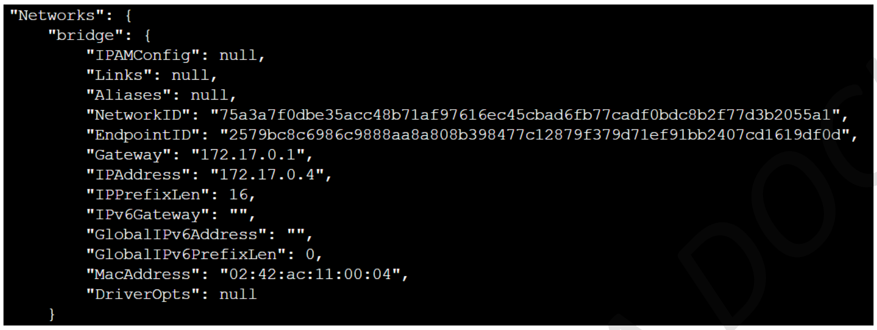
Now, Create a container and do inspect. So in inspect you can see the full networks data. i.e. you can see IP address and everything

- See the all networks → docker network ls

Each container will contain multiple networks. So, we have different types of docker networks

Bridge Network :

- It is a default network that container will communicate with each other within the same host
- Create one container, and inspect the container

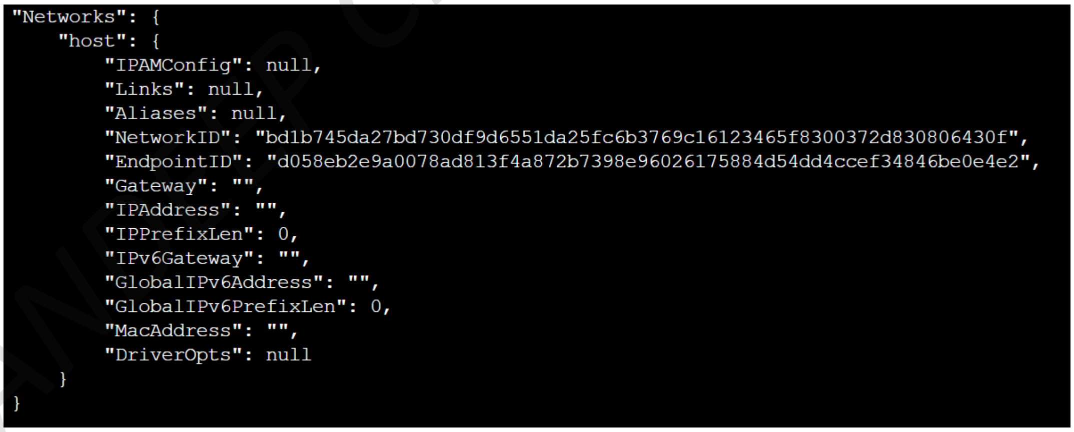


```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "75a3a7f0dbe35acc48b71af97616ec45cbad6fb77cadf0bdc8b2f77d3b2055a1",
        "EndpointID": "2579bc8c6986c9888aa8a808b398477c12879f379d71ef91bb2407cd1619df0d",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.4",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:04",
        "DriverOpts": null
    }
```

- Usually, bridge network contains IP address

Host Network :

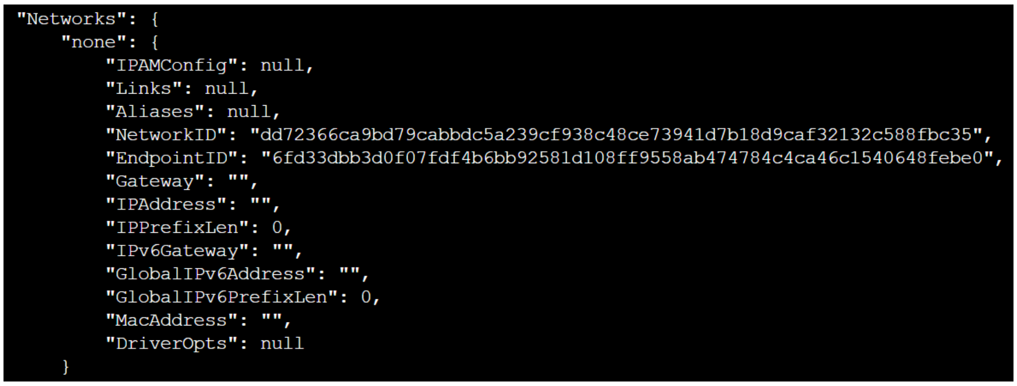
- When you need, your container IP and EC2 instance IP same than we have to use host network
  - i.e., 172.31.3.321 → host/server private IP
- Normally we are getting bridge default IP. But I want to get private IP we are using host network
  - docker run -it --name cont5 --network host ubuntu
  - Now, if you perform inspect you can see the host network



```
"Networks": {
    "host": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "bd1b745da27bd730df9d6551da25fc6b3769c16123465f8300372d830806430f",
        "EndpointID": "d058eb2e9a0078ad813f4a872b7398e96026175884d54dd4ccef34846be0e4e2",
        "Gateway": "",
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "",
        "DriverOpts": null
    }
}
```

None Network:

- When you don't want the containers to get exposed to the world, we use none network.
- It will not provide any network to our container
    - i.e. No IP address
    - docker run -it --name cont --network none ubuntu



```
"Networks": {
    "none": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "dd72366ca9bd79cabbdc5a239cf938c48ce73941d7b18d9caf32132c588fbc35",
        "EndpointID": "6fd33dbb3d0f07fdf4b6bb92581d108ff9558ab474784c4ca46c1540648febe0",
        "Gateway": "",
        "IPAddress": "",
        "IPPrefixLen": 0,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "",
        "DriverOpts": null
    }
```

Overlay Network:

- If you want to establish the connection between the different containers which are present in different servers

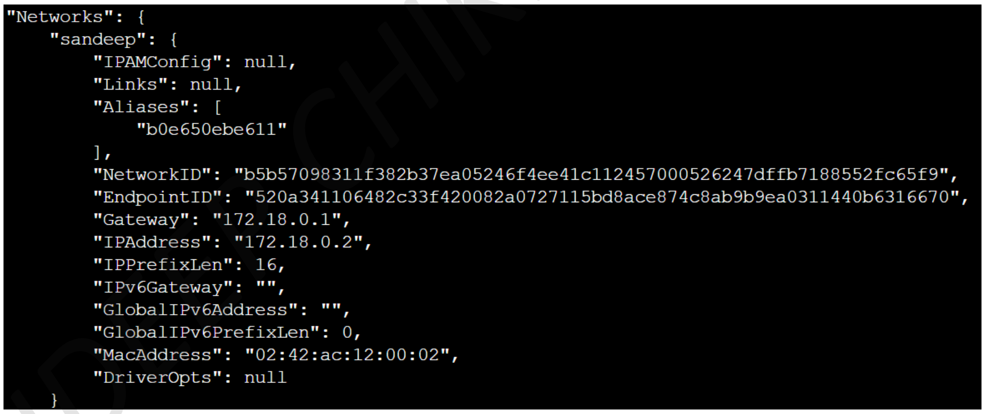
If we have multiple networks to our container means communication will increased

So, these are the Docker networks. first 3 networks are default. Normally, we're using bridge network

- Create Custom Network → docker network create sandeep
- see the list of networks → docker network ls

Now, we have to attach the custom network to our container. the command is

- docker run -it --name cont9 --network sandeep ubuntu
- docker inspect cont9
    - So, now you got one IP address for "sandeep" network



```
"Networks": {
    "sandeep": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "b0e650ebe611"
        ],
        "NetworkID": "b5b57098311f382b37ea05246f4ee41c112457000526247dffb7188552fc65f9",
        "EndpointID": "520a341106482c33f420082a0727115bd8ace874c8ab9b9ea0311440b6316670",
        "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, if you create a network inside "sandeep"
        - If we use the different networks inside a same container the IP address range is
            - IP address will - 172.18.0.1, 172.19.0.1, 172.20.0.1, ... upto 172.256.256.256
        - if we use the same network in different containers the IP address range is

        - 172.18.0.1, 172.18.0.2, 172.18.0.3, ... upto 172.256.256.256

Attach the custom network to existing container, the command is

- docker network connect customNetwork containerName
    - docker network connect sandeep cont1

Disconnect the Networks

- docker network disconnect networkName containerName
    - docker network disconnect sandeep cont99

See the Network IP address

- To get the IP address of the network
    - docker inspect sandeep

Delete the unused Networks

- If the network is not attached to any container, we can simply called unused network
- If you want to delete the network, command is
    - docker network prune

Delete the custom networks

- docker network rm sandeep
    - if that network is attached to a container we can't delete

DOCKER HUB/ DOCKER REGISTRY

It is used to store the images. Docker hub is the default registry

2 Types of registry

1. **Cloud based registry :**
    - when you want to store your images in the cloud like
        - docker hub
        - GCR - Google Container Registry
        - Amazon ECR - Elastic Container Registry
2. **Local registry :**
    - Here, we are storing the images in local like
        - Nexus
        - JFrog
        - DTR - Docker trusted registry

So, here cloud based registry is preferrable. Because in docker hub we just do the account creation and store the images. But in local registry like nexus, we have to take t2.medium and do the setup it is little bit complex.

So, we are using Docker Hub, ECR

- Without Docker hub, In another server we can't run the application

- i.e. system to system (or) server to server, we can't send the image directly. So, we need one platform for this.
- So, we are using Docker hub for that.

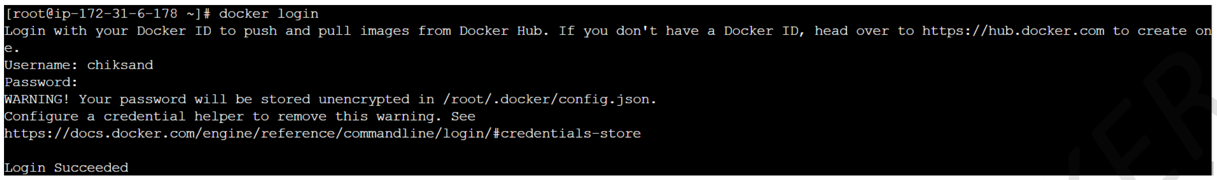
- So, Docker hub what actually does means, from server we will push image into docker. From another server we will pull that image using docker

Note:

First, Go to Google → we need to create Docker Hub account. It will ask username, mail, passwd and verification happened. then after login. that's it

Docker Hub - Practice

- First, we need to login into docker hub. When you want to upload the image in Docker hub.
- Command is → docker login
  - username and password you have to provide



```
[root@ip-172-31-6-178 ~]# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create on
e.
Username: chiksand
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
```

  - After login it will shows like succeeded
  - Without login, you can't push the image into docker hub

Step - 1 :

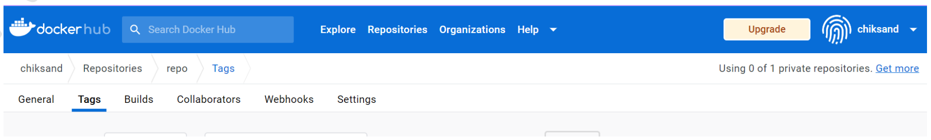
- If you want to push you have to tag that image
- For that one, just write one sample docker file and run it. You will get custom image. Now, push that image into docker hub
  - docker tag imageName dockeruserID/repositoryname
  - docker tag image chiksand/repo
  - docker images → we're having the image

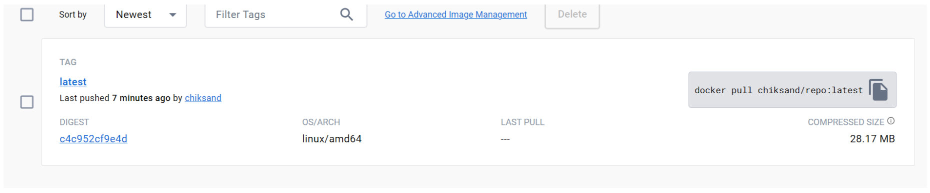
Step - 2 :

- send the image into Docker hub
  - docker push chiksand/repo
  - So, here tagged image will go to docker hub
    - At a time multiple images, we can't push. Single image only we can push
- Go to Docker hub and refresh, we got our image and we have whole docker file

Now, we need to check whether the image is working correct/not. For that we need another server. So, launch normal server and install the docker in that server

- Now, perform the command
  - FYI, the command will be present in Docker hub like below image




Sort by Newest Filter Tags Go to Advanced Image Management Delete

TAG
latest
Last pushed 7 minutes ago by chiksand

docker pull chiksand/repo:latest

DIGEST	OS/ARCH	LAST PULL	COMPRESSED SIZE
c4c952cf9e4d	linux/amd64	---	28.17 MB

- ○ docker pull chiksand/repo: latest
- ○ Now, through this image, create the container, you can access the application in browser

So, like this we can pull the image into servers and we will do our work

How to Store Multiple Images in Docker Hub

Without overriding the image we can do like this

- docker tag image:1 chiksand/repo
- docker push chiksand/repo

Now, we will get separate image in docker hub. So, like this you can store multiple images in docker hub

From Private repo

Create a new repo in private, same like first two steps upto build

- docker tag image1 chiksand/private repo
- docker push chiksand/private repo

It's worked, when you already login into docker hub in your server

- Suppose, you're logout in another server
  - ○ if you want to take the pull in private repo, you can't
  - ○ But, you can take the pull in public repo

OFFICIAL IMAGES in DOCKER HUB

There are so many images that are pre-built in docker hub. So, how the process means we are searching the particular image.

Eg: Usually, Jenkins setup is little bit hard. So, here we are just pull the Jenkins image

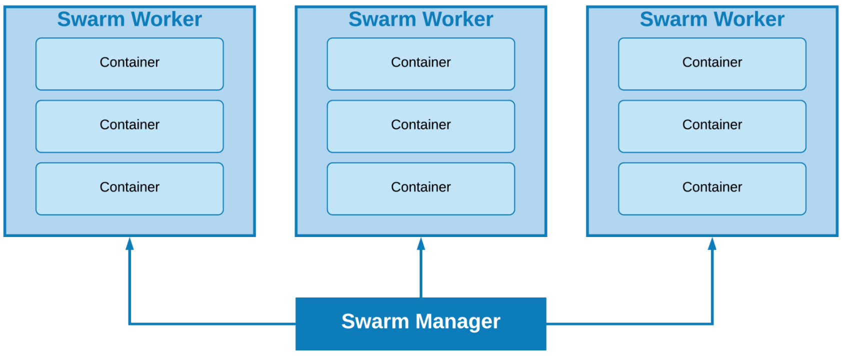
So, in official image they mentioned how to use that image

DOCKER SWARM

Docker Swarm is an Orchestration service (or) group of service

- It is similar to master-slave concept
- Within the docker that allows us to manage and handle multiple containers at the same time
- Docker Swarm is a group of servers that runs the docker application
  - ○ i.e. for running the docker application, in docker swarm we're creating group of servers
- We used to manage the multiple containers on multiple servers.
- This can be implemented by the "Cluster"

- The activities of the cluster are controlled by a "Swarm Manager" and machines that have joined the cluster is called "Swarm Worker"
- Here, it is the example of master and slave



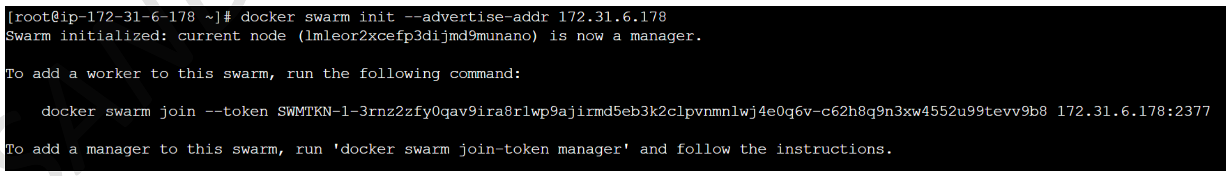
- Docker Engine helps to create Docker Swarm
    - i.e. if you want to implement Docker Swarm. In that system we have to install docker for 2 servers. i.e. Swarm Manager & Swarm Worker
- In the cluster we are having 2 nodes
    a. Worker nodes
    b. Manager nodes
- The worker nodes are connected to the manager nodes
- So, any scaling i.e. containers increase (or) updates needs to be done means first, it will go to the manager node
- From the manager node, all the things will go to the worker node
- Manager nodes are used to divide the work among the worker nodes
- Each worker node will work on an individual service for better performance
    - i.e. 1 - worker node, 1 - service

COMPONENTS in Docker Swarm

1. SERVICE
    - It represents a part of the feature of an application
2. TASK
    - A Single part of work (or) Work that we are doing
3. MANAGER
    - This manages/distributes the work among the different nodes
4. WORKER
    - which works for a specific purpose of the service

PRACTICAL

1. Take 1 normal server named as manager and inside the server install & restart the docker
2. Initializing Swarm
    - docker swarm init --advertise-addr PrivateIP



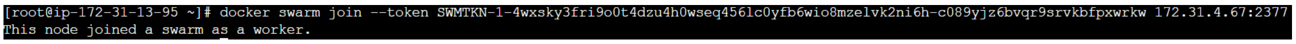
```
[root@ip-172-31-6-178 ~]# docker swarm init --advertise-addr 172.31.6.178
Swarm initialized: current node (lmleor2xcefp3dijmd9munano) is now a manager.

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

    docker swarm join --token SWMTKN-1-3rnz2zfy0qav9ira8r1wp9ajirmd5eb3k2clpvnmnlwj4e0q6v-c62h8q9n3xw4552u99tevv9b8 172.31.6.178:2377

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

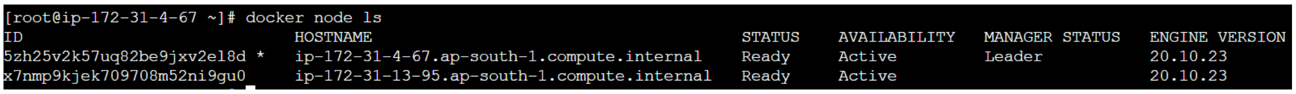
- Here, you got all details, how to connect with worker node
- i.e. in master the token is generated. If we give this token in another server it will be work as a worker node
- Now, take 2 normal servers named as worker-1,2 and install & restart the docker here in 2 servers
- Now, copy the token from manager server and paste in 2nd server. It will joined a swarm as a worker



```
[root@ip-172-31-13-95 ~]# docker swarm join --token SWMTKN-1-4wxsky3fri9o0t4dzu4h0wseq456lc0yfb6wio8mzelvk2ni6h-c089yjz6bvqr9srvkbfpxwrkw 172.31.4.67:2377
This node joined a swarm as a worker.
```

3. See the list of nodes in manager

   - docker node ls → you will get 3 nodes like 1-leader, 2-worker

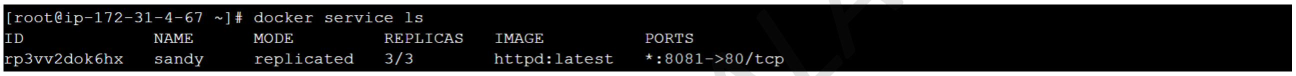


```
[root@ip-172-31-4-67 ~]# docker node ls
ID                            HOSTNAME                                      STATUS    AVAILABILITY   MANAGER STATUS   ENGINE VERSION
5zh25v2k57uq82be9jxv2e18d *   ip-172-31-4-67.ap-south-1.compute.internal    Ready     Active         Leader           20.10.23
x7nmp9kjek709708m52ni9gu0     ip-172-31-13-95.ap-south-1.compute.internal   Ready     Active                          20.10.23
```

- ☐ Now, task is - In 2 slave servers at a time we need to create a container from manager
- So, here we are creating in the service format. Here, service means container

Create Service/Container

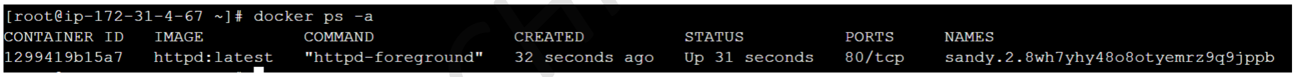
- docker service create --name sandy --replicas 3 --publish 8081:80 httpd
  - here, sandy → service name
  - replicas → duplicates i.e. if the container is stopped/deleted means automatically another container will created with the same configuration
  - 3 → duplicate containers, like how many containers you need, just give the number
- Now, 1 container is created
- ☐ See the list of Services
- docker service ls



```
[root@ip-172-31-4-67 ~]# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE          PORTS
rp3vv2dok6hx   sandy     replicated   3/3        httpd:latest   *:8081->80/tcp
```

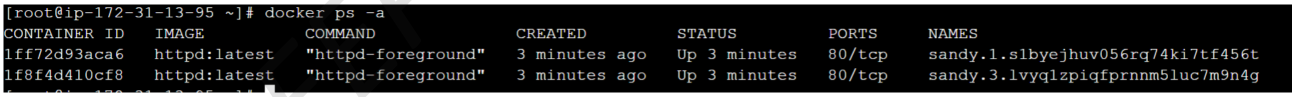
- It will work in only manager, not in slaves
- Now, if you perform → docker ps -a

Actually we get 3 but here, it will show you 1 container. So, here it acts as a master & slave



```
[root@ip-172-31-4-67 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND              CREATED          STATUS          PORTS     NAMES
1299419b15a7   httpd:latest   "httpd-foreground"   32 seconds ago   Up 31 seconds   80/tcp    sandy.2.8wh7yhy48o8otyemrz9q9jppb
```

- Now, check in slave server, you will get the remaining containers



```
[root@ip-172-31-13-95 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND              CREATED         STATUS         PORTS     NAMES
1ff72d93aca6   httpd:latest   "httpd-foreground"   3 minutes ago   Up 3 minutes   80/tcp    sandy.1.s1byejhuv056rq74ki7tf456t
1f8f4d410cf8   httpd:latest   "httpd-foreground"   3 minutes ago   Up 3 minutes   80/tcp    sandy.3.1vyq1zpiqfprnnm5luc7m9n4g
```

- Now, check it's working (or) not
  - Go to manager → copy publicIP:8081 in browser → it works

same like check in slave servers, you will get it

This is the basic example for Docker swarm

- [ ] Now, create another service with 2 replicas

- Go to Manager server → docker service create --name devops --replicas 2 --publish 8082:80 httpd
  - Now, check → docker ps -a → you are having devops.2
- Go to Worker -1 server → docker ps -a → you are having devops.1
- In Worker-2 we don't have. Because we gave only 2 replicas

Note :

Here, If the worker node contains less containers. Manager will send the containers to that worker node. It balance the work load

i.e. Now take another service with 2 replicas. This time, the container will add in Worker-2

Task -2 : Create Docker file, and we have to run the image from the Docker file

  - FROM ubuntu
  - RUN apt update -y
  - RUN apt install apache2 -y
  - RUN echo "hi this is app" > /var/www/html/index.html
  - CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
- Build the image → docker build -t image .
- Create the service based on image
  - docker service create --name sandy --replicas 3 --publish 8084:80 image
  - docker service ls
    - Here, 3 servers are present inside manager. Not in slave servers. Because, that is local image
    - Httpd, nginx, ubuntu these are all open images
    - When you check in worker-1,2. It is not there. that's why when we run this command we're seeing no such image
  - So, 3 containers are created inside manager server
  - So, httpd, nginx, ubuntu these are open images. So, these are pulling from the docker hub
  - So, present we're pushing our "image" into docker hub
- In manager server → docker login
  - docker tag image chiksand/repo
  - docker push chiksand/repo
  - docker images → we are having image "chiksand/repo"
- Now, through that image we have to create a service
  - docker service create --name swarm --replicas 3 --publish 8085:80 chiksand/repo
  - docker ps -a
    - Now, we are having 1 container in manager same like in slaves also we're having containers

So, like this through custom image also we can create the services in docker swarm

DOCKER SERVICE COMMANDS

1. See the list of services
   - docker service ls
2. Checking the how many containers inside a service
   - docker service ps ServiceName
3. If we stop/delete a container in worker node. Automatically another container will created

   - docker stop contID
   - docker ps -a → you can see container exited and created
     - Because of the Auto/self healing
4. Remove Service
   - docker service rm ServiceName
   - If we remove the service means, containers also removed
5. Check the logs
   - docker service logs ServiceName
6. Inspect Service
   - docker service inspect ServiceName
7. See the service history like Eg: how many containers, IP address, etc..
   - docker service ps ServiceName
8. removing the manager in docker swarm
   - docker swarm leave --force

Update Image From the Service

1. Update Dockerfile
2. Build the Dockerfile
3. docker tag image chiksand/repo
4. docker push chiksand/repo
5. So, right now, present service we need to update the image

- docker service update --image ImageName ServiceName
  - docker service update --image chiksand/repo swarm
  - Check in browser, whether it's working/not

Rollback to Previous image

Here, I want to go back to the previous image means. Without updating the image you can't rollback to previous image

- docker service rollback ServiceName
  - So, here the common query is we already update the image. How we can get the previous image
  - It will stores the log files. So, we can rollback to any image

SCALING in Docker Swarm

If you want to increase/decrease the replicas for containers we can use Scaling

Here Scaling is 2 types

1. Container Scaling → using docker
2. Server Scaling → using aws, Based on the users request it will increase

In Manager Server → docker service ls

- Now, I want to increase upto 5 replicas
  - docker service scale ServiceName=5
- Now, check the containers in Manager & Worker servers

SWARM COMMANDS related to NODE

1. see the list of nodes
   - In manager → docker node ls
2. If you want to remove the worker from Swarm cluster, Directly Manager can't remove. For that we have command
   - docker swarm leave → perform this command in slave servers
3. Check the nodes
   - docker node ls → here, still we have 3 containers, but the status is changed to down
4. So, now remove the node from the manager
   - docker node rm NodeID
   - docker node ls → now we have 2 containers

So, overall if you want to remove the node from the manager means, first, node will leave from that particular service, then we can perform the command

So, again you want to add the nodes, you have to do it starting process

DOCKER - COMPOSE

- In docker swarm, we created 1 container/service in multiple servers using master & Slave (or) Manager & Worker concept
- But In docker compose, we deployed multiple containers in single server
- It is complete opposite to docker swarm
  - Here, multiple containers means full application. i.e. frontend, backend, database containers present
  - So, right now these 3 containers will present in single container. For that, we are using Docker Compose
- Here, Manually we can do. But here, we're doing through "compose file"
  - So, here suppose we have 3 apps. For these, if we have to create container means first, write the docker file. then after build and get the image. After run the image you got a container. These is the manual process for all 3 apps
- But, here in Docker compose, we took docker files. For all docker files we write one compose file
- In this compose file, we are having containers related configuration is present.
  - i.e. container name, port, volume, networks
  - So, like this if we write the container related requirements in compose file and if we execute the file means. In compose files, whatever the containers we're having that all will created
  - Here, Automate happened. i.e. image build & container creation at a time happened

So, overall, In real time if developers write the code means, we are writing the docker files for that. For that docker file, Here we are writing the Compose file and will execute that

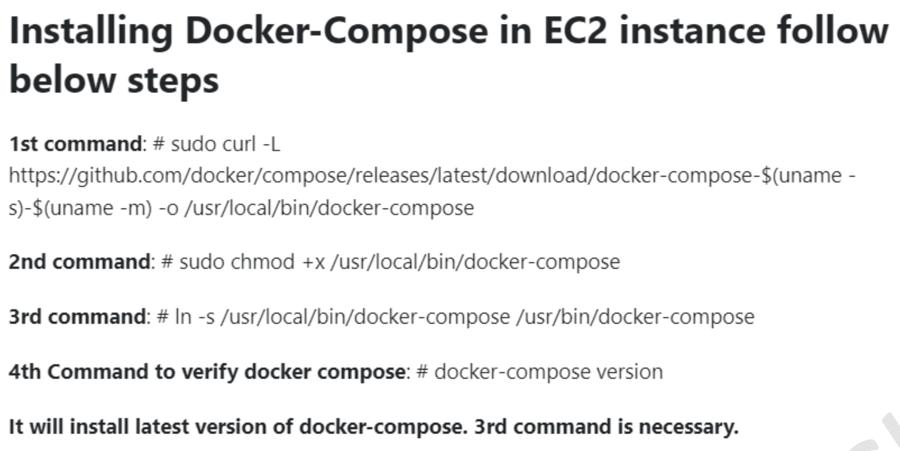
Def:

- Docker compose is a tool used to build, run and ship the multiple containers for application
- It is used to create multiple containers in a single host/server
- It used YAML file to manage multi containers as a single service
  - i.e. In docker compose file we are writing the container configurations, that should be written in YAML format and the compose file extends with ".yml"

- The compose file provides a way to document and configure all of the applications service dependencies
  - like - databases, queues, caches, web service API's, etc..,
- In one directory, we can write only one docker-compose file

PRACTICAL

1. Install & Restart the docker in a server
2. Install docker compose
   - Go to google → search docker compose install on amazon Linux → go to stack overflow



Installing Docker-Compose in EC2 instance follow below steps

1st command: # sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

2nd command: # sudo chmod +x /usr/local/bin/docker-compose

3rd command: # ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

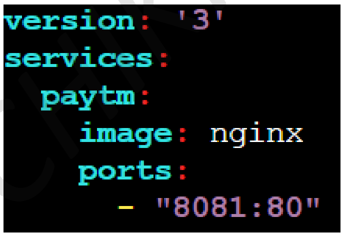
4th Command to verify docker compose: # docker-compose version

It will install latest version of docker-compose. 3rd command is necessary.

   - Perform the above steps, you can get docker-compose file in your local
- Install docker - compose
  - sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
  - sudo chmod +x /usr/local/bin/docker-compose
  - ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
  - docker-compose version

3. Write the Docker compose file

   - vim docker-compose.yml

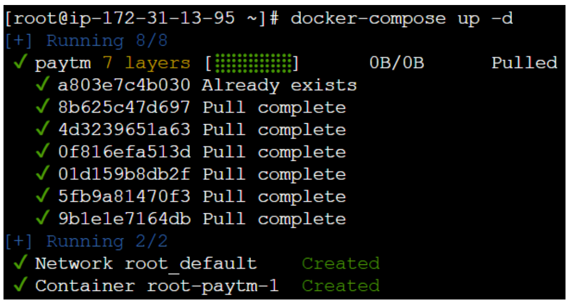


```
version: '3'
services:
  paytm:
    image: nginx
    ports:
      - "8081:80"
```

   - Version → It specifies the version of the compose file, default 3
   - Services → if you want to create a service, first mention services
   - paytm → service name
     - Here, service acts as a container
     - Inside the service we are having containers, and we have to give the image name

4. Execute the compose file

   - docker-compose up -d → here, d is for detach mode

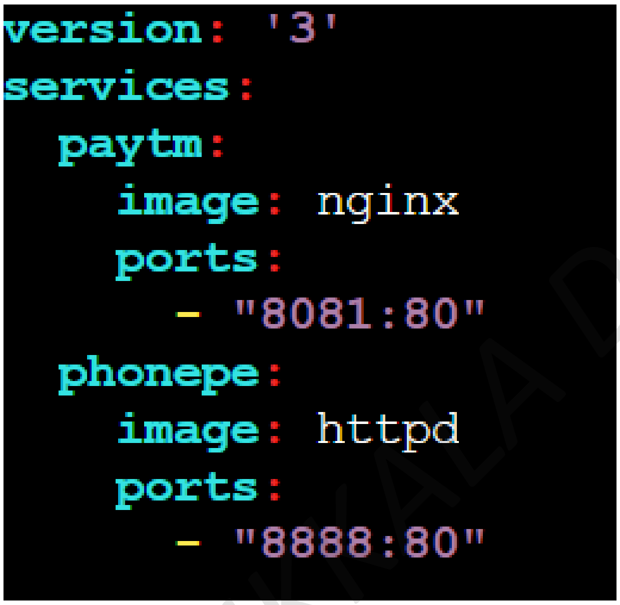

```
[root@ip-172-31-13-95 ~]# docker-compose up -d
[+] Running 8/8
 ✓ paytm 7 layers [⣿⣿⣿⣿⣿⣿⣿]      0B/0B      Pulled
   ✓ a803e7c4b030 Already exists
   ✓ 8b625c47d697 Pull complete
   ✓ 4d3239651a63 Pull complete
   ✓ 0f816efa513d Pull complete
   ✓ 01d159b8db2f Pull complete
   ✓ 5fb9a81470f3 Pull complete
   ✓ 9b1e1e7164db Pull complete
[+] Running 2/2
 ✓ Network root_default     Created
 ✓ Container root-paytm-1   Created
```

- After performing this command, automatically containers will gets created which are present in docker compose file
  - docker ps -a
  - So, here with the name of paytm, container will be created → root-paytm-1

5. Access the application in browser

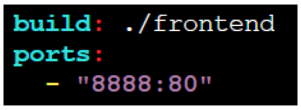
   - publicIP:8081 → you will get the output in the browser

Creating Multiple Services in Docker Compose


```
version: '3'
services:
  paytm:
    image: nginx
    ports:
      - "8081:80"
  phonepe:
    image: httpd
    ports:
      - "8888:80"
```

AUTOMATE IMAGE BUILD IN DOCKER-COMPOSE

- If I update the code in frontend i.e. in the docker file. Again we need to build the image and the new image will be placed inside the compose file
- So, whenever develop the code. We need to build that image, and update in compose.yml
- Instead of doing this, whenever developer write the code, automatically it will update in docker-compose file. For that we're using build component
  - In docker file you changed data and you built image with existing name. Image will overrided
- vim docker-compose.yml


```
build: ./frontend
ports:
  - "8888:80"
```

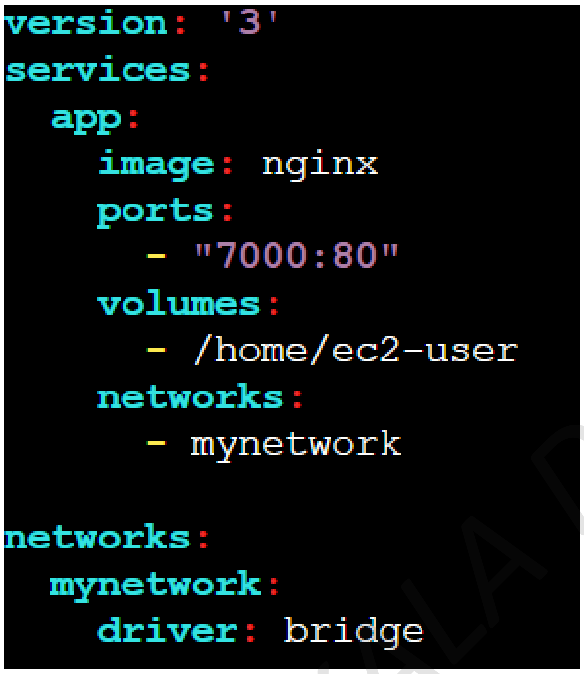
- Now, build the docker-compose for this image

  - docker-compose build
    - Now, here image is builded, then it will update into the container
  - docker-compose up -d

Now, execute and access in browser. You, will get the output

Here, in docker-compose we have to define 4 components for containers

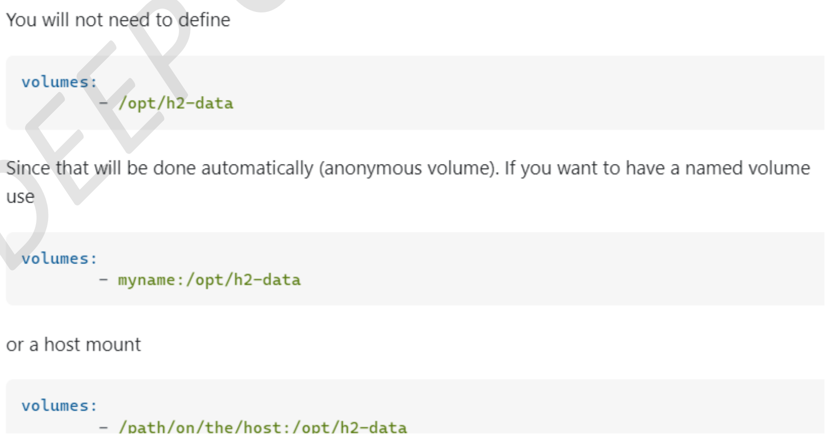
- P I V N
  - P → ports
  - I → Image
  - V → Volume
  - N → Network
- Now, we have to add these components inside a docker-compose



```
version: '3'
services:
  app:
    image: nginx
    ports:
      - "7000:80"
    volumes:
      - /home/ec2-user
    networks:
      - mynetwork

networks:
  mynetwork:
    driver: bridge
```

- Now, execute the docker-compose
  - docker-compose up -d
- perform inspect in container
  - docker inspect contID
    - Now, you will see everything
- For defining volumes in docker-compose, give like this



You will not need to define

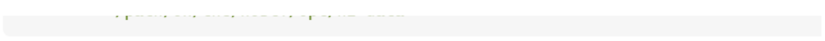
```
volumes:
        - /opt/h2-data
```

Since that will be done automatically (anonymous volume). If you want to have a named volume use

```
volumes:
        - myname:/opt/h2-data
```

or a host mount

```
volumes:
        - /path/on/the/host:/opt/h2-data
```



This is the way we can give all components inside a docker compose file

(FAQ) Suppose, in one service, I got issue in docker compose. how to resolve ?

Here, We're going into particular docker file and update the docker file. Build the compose file. Old containers are running, new containers will deploy

DOCKER-COMPOSE COMMANDS

1. Create the containers in docker compose
   - docker-compose up -d
2. Stop and remove the composed containers
   - docker-compose down
3. Stop the containers from the compose file
   - docker-compose stop
   - docker ps -a
     - start the stopped containers → docker-compose up -d
4. See the list of images in docker-compose file
   - docker-compose images
5. See the compose containers
   - docker-compose ps
   - docker ps -a → we see manual created containers
6. See the logs in docker compose. Inside the logs containers start & end details are present here
   - docker-compose logs
7. See the code configuration in compose file, not from vim editor
   - docker-compose config
8. Pause & UnPause in container
   - docker-compose pause
     - i.e. no updates will happen in that container. that means container will struck
   - docker-compose unpause
     - Used to unpause the container

Usually, we're maximum using this "docker-compose.yml" file name. If we use another name we're getting this error

Eg:

- vim docker-compose.yml (this is the standard way)
- vim sandeep.yml (docker won't execute this file, it shows error)

For that, the command is → docker-compose -f sandeep.yml up -d

This is the code we're using for another file names

DOCKER STACK

If you want to deploy multiple services in multiple servers we're using docker stack

- It is the combination of Docker Swarm + Docker Compose

- Here, first we have to initiate the Swarm, otherwise stack doesn't work here
- Here, we have to write compose files
- Docker stack is used to create multiple services on multiple hosts
  - i.e. it will create multiple containers on multiple servers with the help of compose file
- To use the docker stack we have initialized docker swarm, if we are not using docker swarm, docker stack will not work
- Once we remove the stack automatically all the containers will gets deleted
- We can share the containers from manager to worker according to the replicas
- In docker stack, we are using overlay network

Eg: Let's assume, if we have 2 servers which is manager and worker. If we deployed a stack with 4 replicas. 2 are present in manager and 2 are present in worker

- Here, manager will divide the work based on the load on a server

PRACTICAL

Take 3 servers named as manager, worker-1 & worker-2

Step -1 :

Install & restart the docker

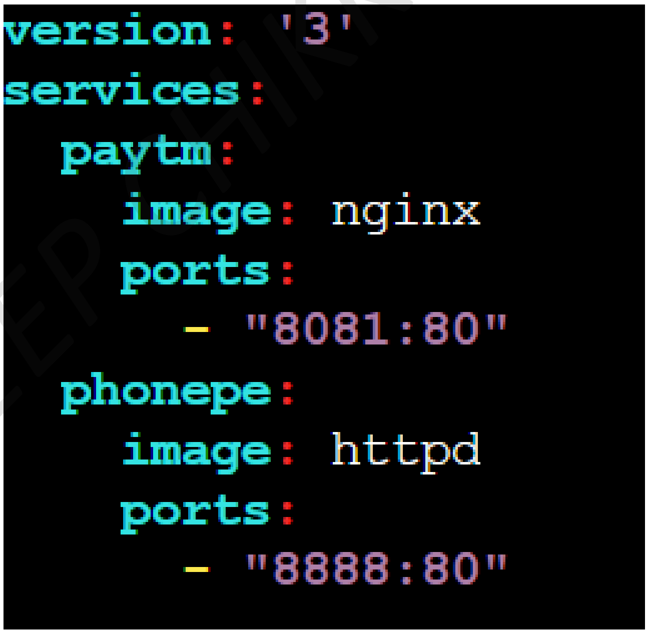
In manager

  - docker swarm init --advertise-addr privateIP
    - copy the token, paste in worker-1&2
  - See the list of nodes → docker node ls

Step -2 :

Write the docker-compose file, for that install the docker compose

- vi docker-compose.yml



```
version: '3'
services:
  paytm:
    image: nginx
    ports:
      - "8081:80"
  phonepe:
    image: httpd
    ports:
      - "8888:80"
```

Step -3: Execute this file

- docker stack deploy --compose-file docker-compose.yml stackName
    - docker ps -a → containers will be present in worker nodes

Step -4: Access the browser

- check in browser → publicIP:8888

Use case Scenario for Docker Stack

Suppose, we have paytm app. Usually, daily 1k people accessing the paytm. Due to festivals, this time 100k people use this application. Due to multiple requests, server can't handle the capacity.

So, that's why i want to run my application in multiple servers. For that thing we're using cluster

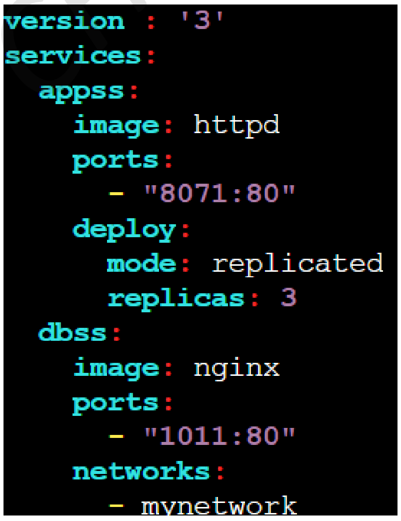
Here, if you need multiple containers, we can use replicas for this. replicas used for high availability and application performs well

COMMANDS

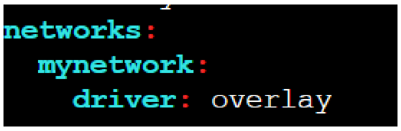
1. create replica for this server
    - docker service scale serviceName=2
2. See the list of stacks
    - docker stack ls
3. Remove the stack
    - docker stack rm stackName
4. See the stack related services
    - docker stack services stackName
5. See the commands in stack
    - docker stack

Through Docker compose file, we can do docker stack, first execute this compose file in stack

- docker stack deploy --compose-file docker-compose.yml mystack/stackName
- docker service ls
    - for creating replicas it will takes time



```
version : '3'
services:
  appss:
    image: httpd
    ports:
      - "8071:80"
    deploy:
      mode: replicated
      replicas: 3
  dbss:
    image: nginx
    ports:
      - "1011:80"
    networks:
      - mynetwork
```



```
networks:
  mynetwork:
    driver: overlay
```

PORTAINER

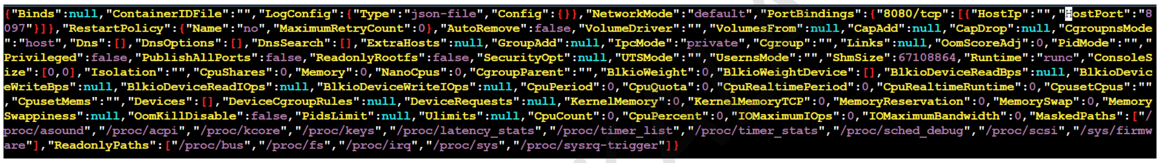
It is an Docker-GUI. It works with the help of docker stack. i.e. we can create the gui for the entire docker by using portainer concept

- It is container organizer, designed to make tasks easier, whether they're clustered (or) not.
- Able to connect multiple clusters, access the containers, migrate stacks between clusters
- It is not a testing environment. Mainly used for production routines in large companies
- Portainer consists of 2 elements.
  - The portainer Server
  - The portainer agent

Change the Container Port

Create container → docker run -itd --name cont -p 8081:80 httpd

- curl publicIP:port
  - I want to change the port number
  - First, stop the container
  - So, go to cd /var/lib/docker → ll
  - cd docker → cd containers → cd contID →
- Here, go to hostconfig.json → vi hostconfig.json



```
{"Binds":null,"ContainerIDFile":"","LogConfig":{"Type":"json-file","Config":{}},"NetworkMode":"default","PortBindings":{"8080/tcp":[{"HostIp":"","HostPort":"8
097"}]},"RestartPolicy":{"Name":"no","MaximumRetryCount":0},"AutoRemove":false,"VolumeDriver":"","VolumesFrom":null,"CapAdd":null,"CapDrop":null,"CgroupnsMode
":"host","Dns":[],"DnsOptions":[],"DnsSearch":[],"ExtraHosts":null,"GroupAdd":null,"IpcMode":"private","Cgroup":"","Links":null,"OomScoreAdj":0,"PidMode":"","
Privileged":false,"PublishAllPorts":false,"ReadonlyRootfs":false,"SecurityOpt":null,"UTSMode":"","UsernsMode":"","ShmSize":67108864,"Runtime":"runc","ConsoleS
ize":[0,0],"Isolation":"","CpuShares":0,"Memory":0,"NanoCpus":0,"CgroupParent":"","BlkioWeight":0,"BlkioWeightDevice":[],"BlkioDeviceReadBps":null,"BlkioDevic
eWriteBps":null,"BlkioDeviceReadIOps":null,"BlkioDeviceWriteIOps":null,"CpuPeriod":0,"CpuQuota":0,"CpuRealtimePeriod":0,"CpuRealtimeRuntime":0,"CpusetCpus":""
,"CpusetMems":"","Devices":[],"DeviceCgroupRules":null,"DeviceRequests":null,"KernelMemory":0,"KernelMemoryTCP":0,"MemoryReservation":0,"MemorySwap":0,"Memory
Swappiness":null,"OomKillDisable":false,"PidsLimit":null,"Ulimits":null,"CpuCount":0,"CpuPercent":0,"IOMaximumIOps":0,"IOMaximumBandwidth":0,"MaskedPaths":["/
proc/asound","/proc/acpi","/proc/kcore","/proc/keys","/proc/latency_stats","/proc/timer_list","/proc/timer_stats","/proc/sched_debug","/proc/scsi","/sys/firmw
are"],"ReadonlyPaths":["/proc/bus","/proc/fs","/proc/irq","/proc/sys","/proc/sysrq-trigger"]}
```

  - Here "HostPort": "8097", change this port number and exit
- Now, restart the docker
- Start the container

So, like this we can change the port number for container