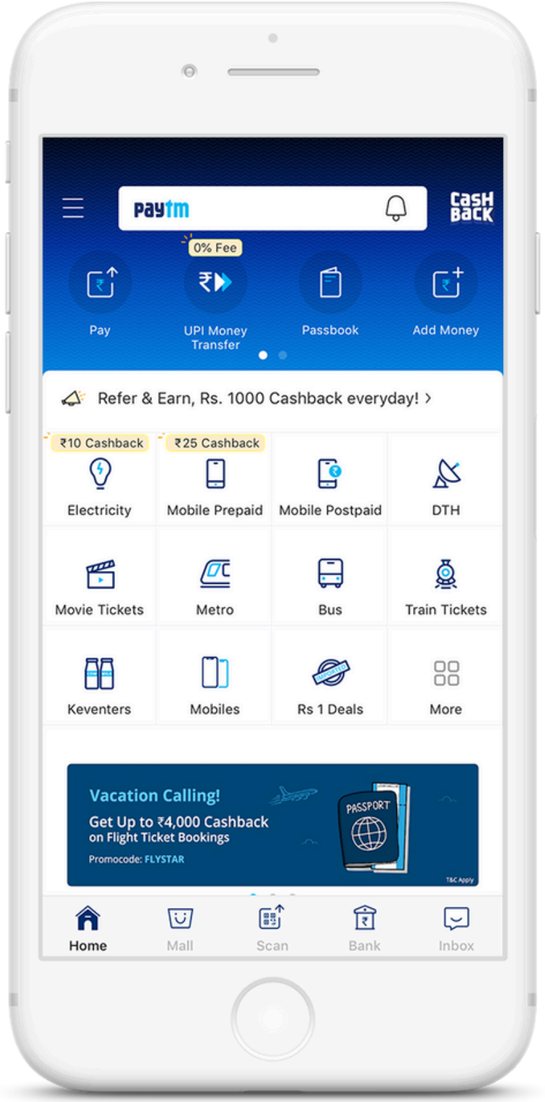
DOCKER

why we use it?

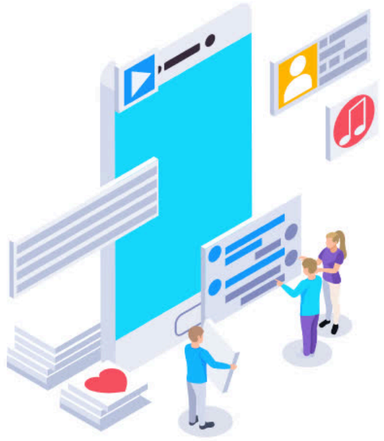
MONOLITHIC

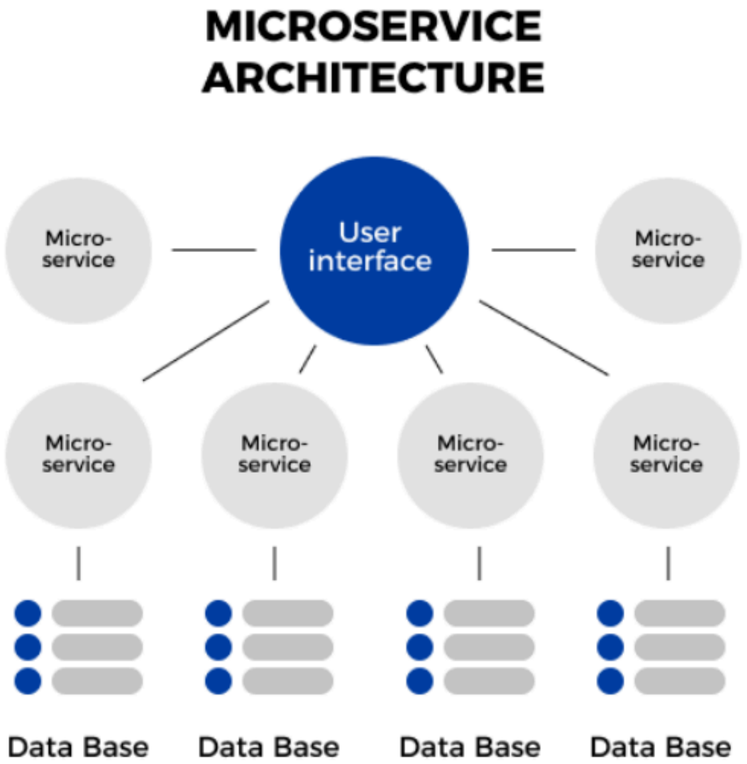
If an application contains N number of services (Let's take Paytm has Money Transactions, Movie Tickets, Train tickets, etc..) If all these services are included in one server then it will be called Monolithic Architecture. Every monolithic Architecture has only one database for all the services.



MICRO SERVICE

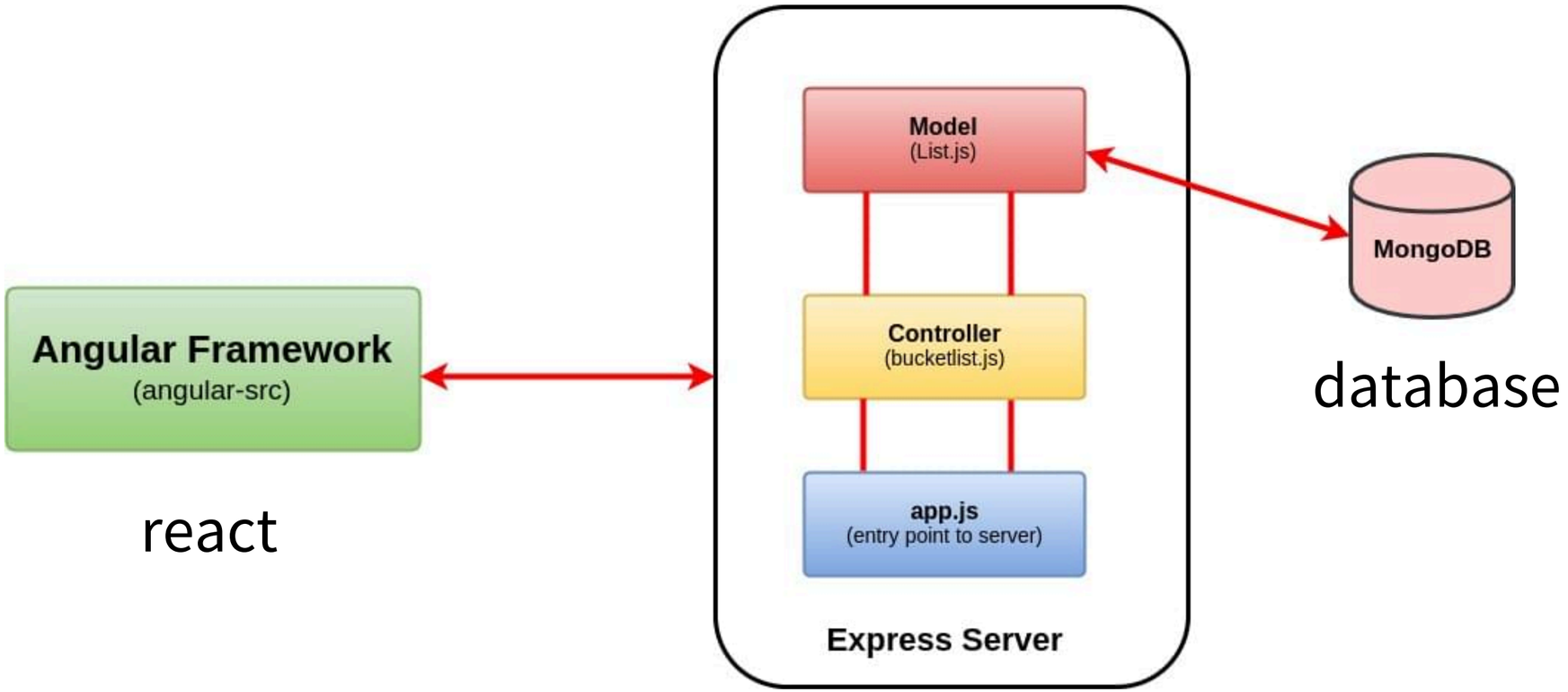
If an application contains N number of services (Let's take Paytm has Money Transactions, Movie Tickets, Train tickets, etc..) if every service has its own individual servers then it is called microservices. Every microservice architecture has its own database for each service.





WHY DOCKER

let us assume that we are developing an application, and every application has front end, backend and Database.

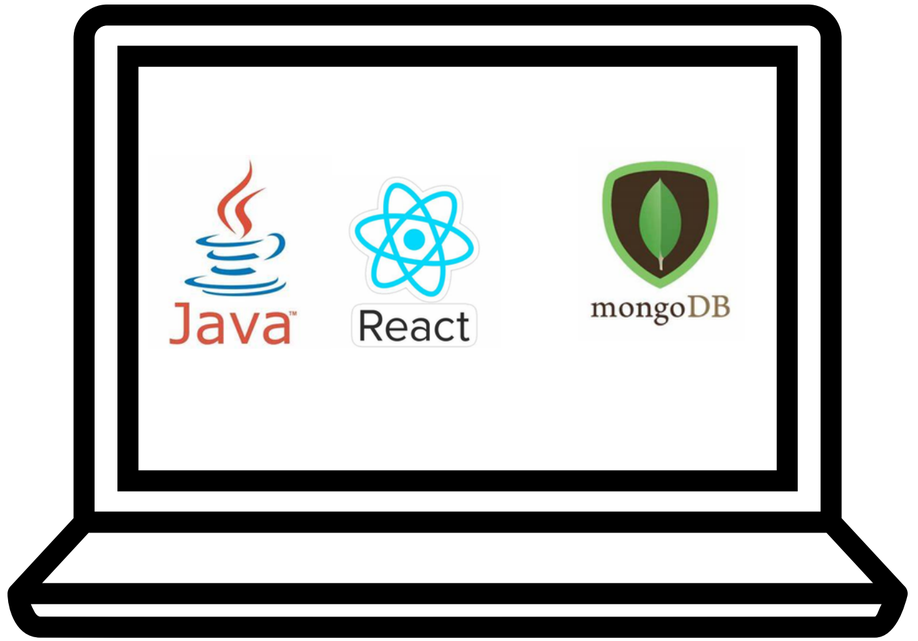


To develop the application we need install those dependencies to run to the code.

So i installed Java11, ReactJS and MongoDB to run the code.
After some time, i need another versions of java, react and mongo DB for my application to run the code.

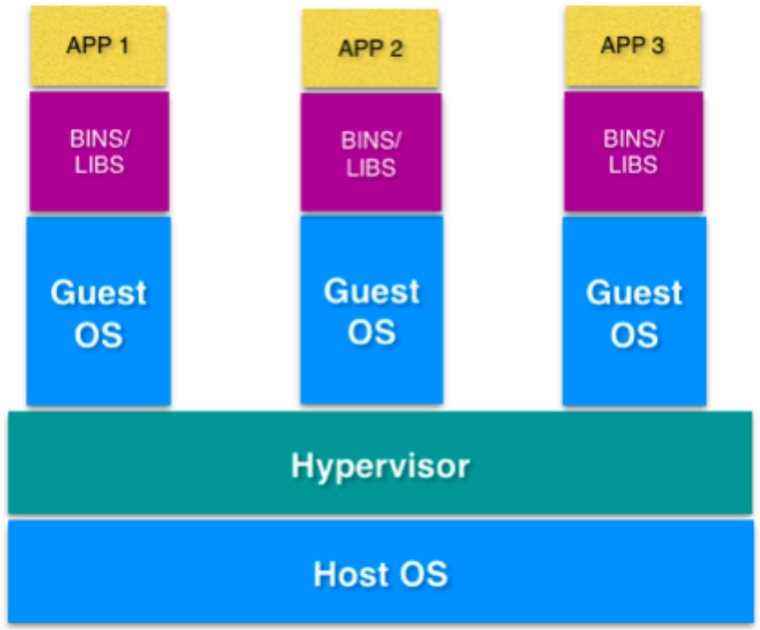
So its really a hectic situation to maintain multiple versions of same tool in our system.

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 hots guest OS in our machine.
by using this Guest OS we can run multiple application on same machine.
Hypervisor is used to create the virtualization.



VIRTUAL MACHINE ARCHITECTURE

DRAWBACKS:

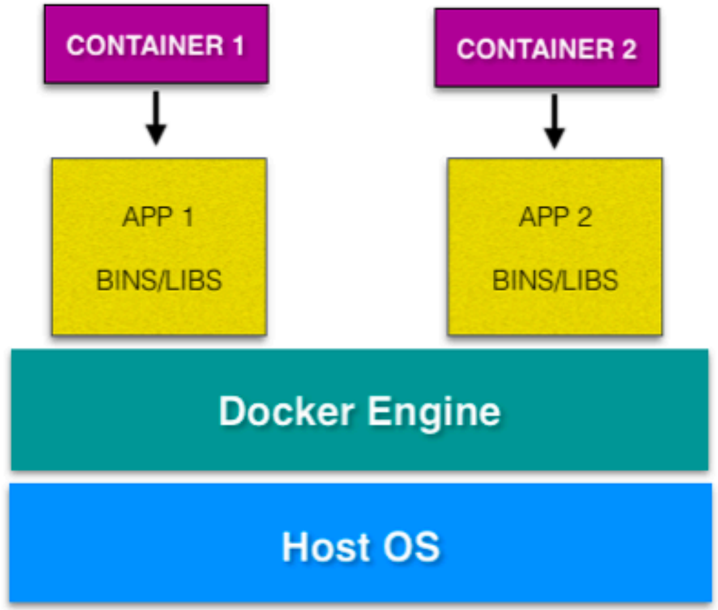
- It is old method.
- If we use multiple guest OS then the performance of the system is low.

CONTAINERIZATION:

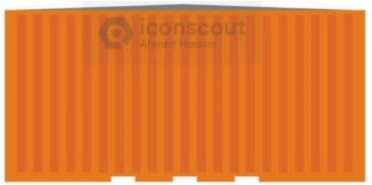
- It is used to pack the application along with its dependencies to run the application.

CONTAINER:

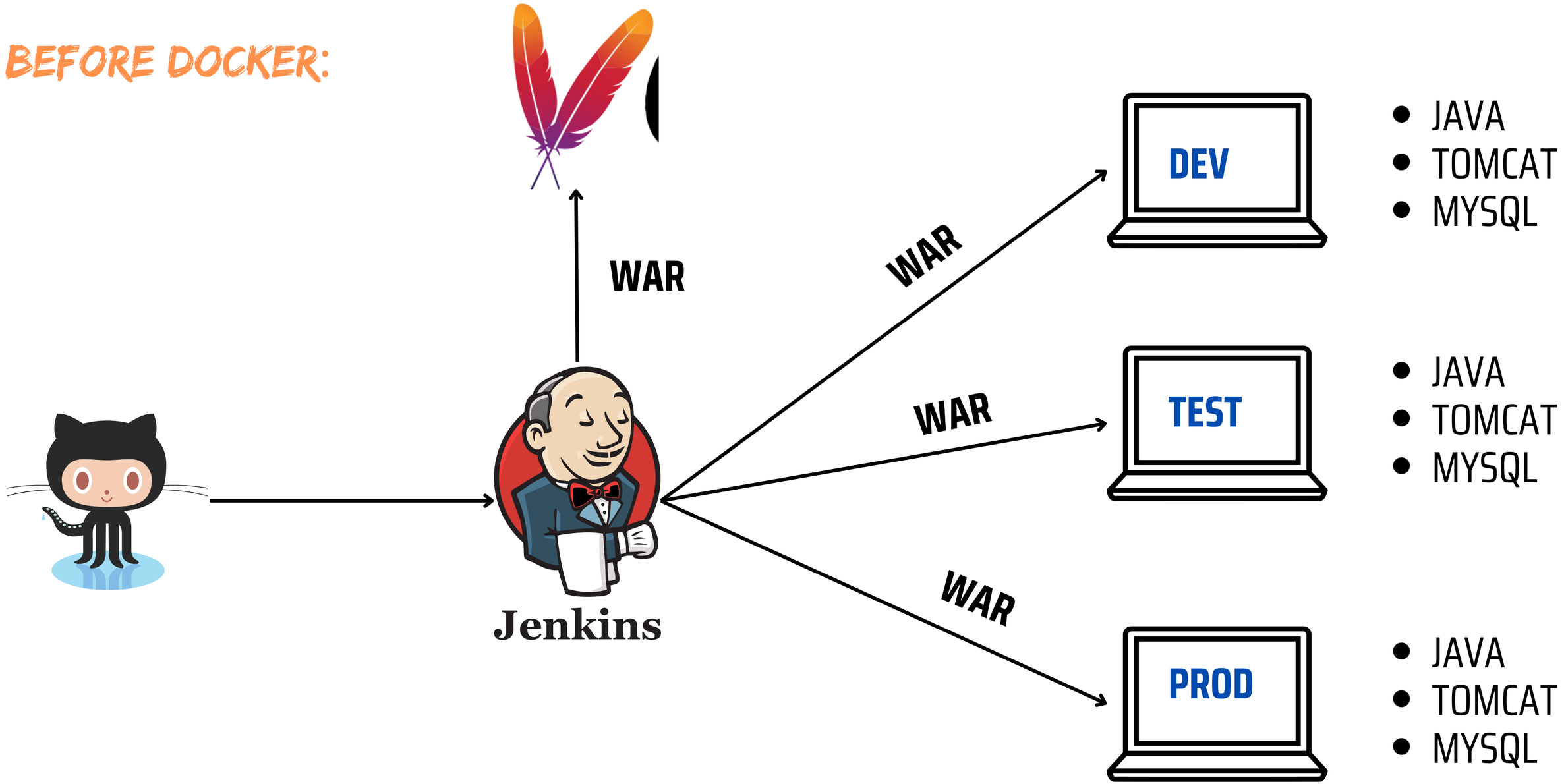
- Container is nothing but, it is a virtual machine which does not have any OS.
- Docker is used to create these containers.



DOCKER ARCHITECTURE



BEFORE DOCKER:



AFTER DOCKER:

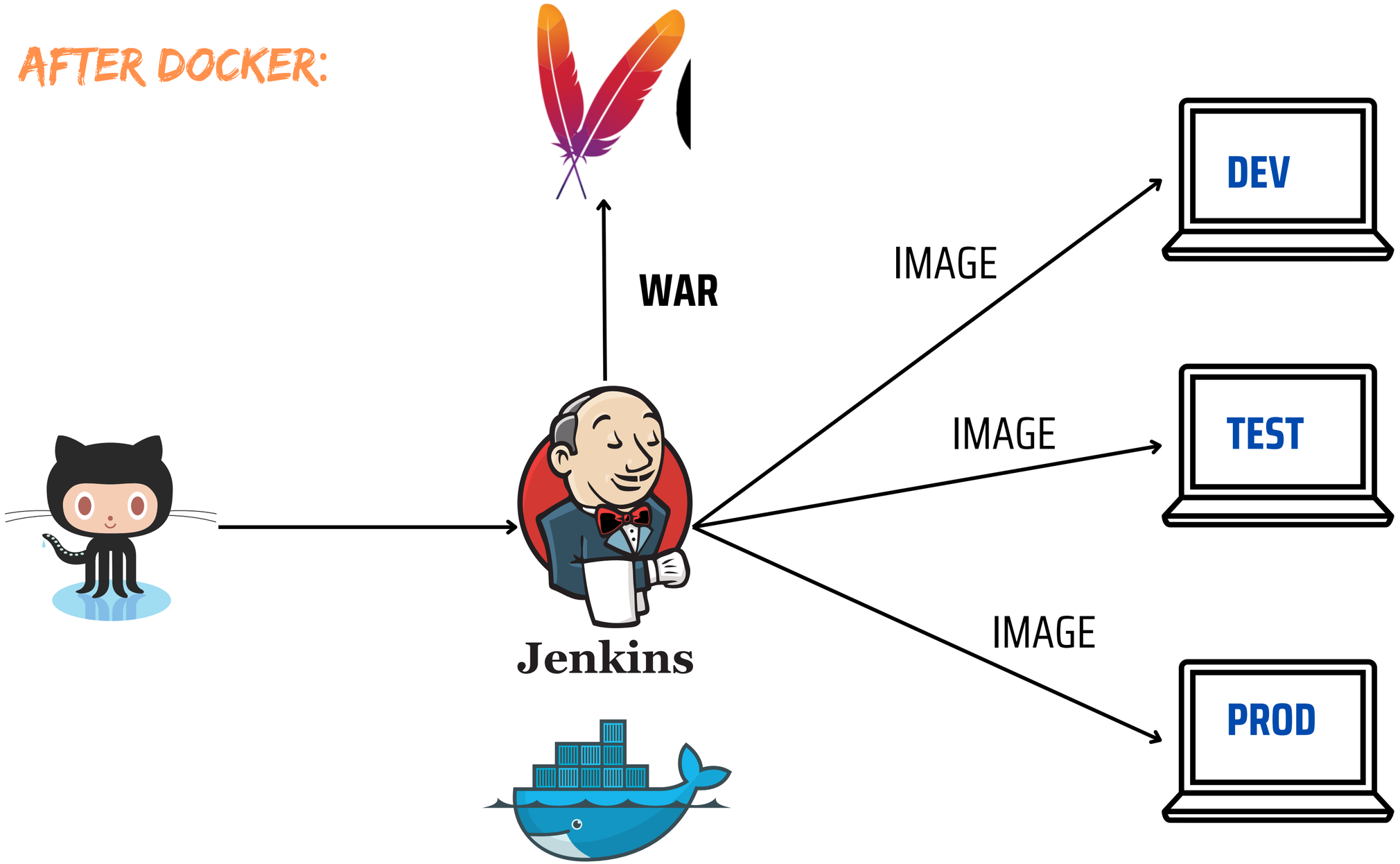
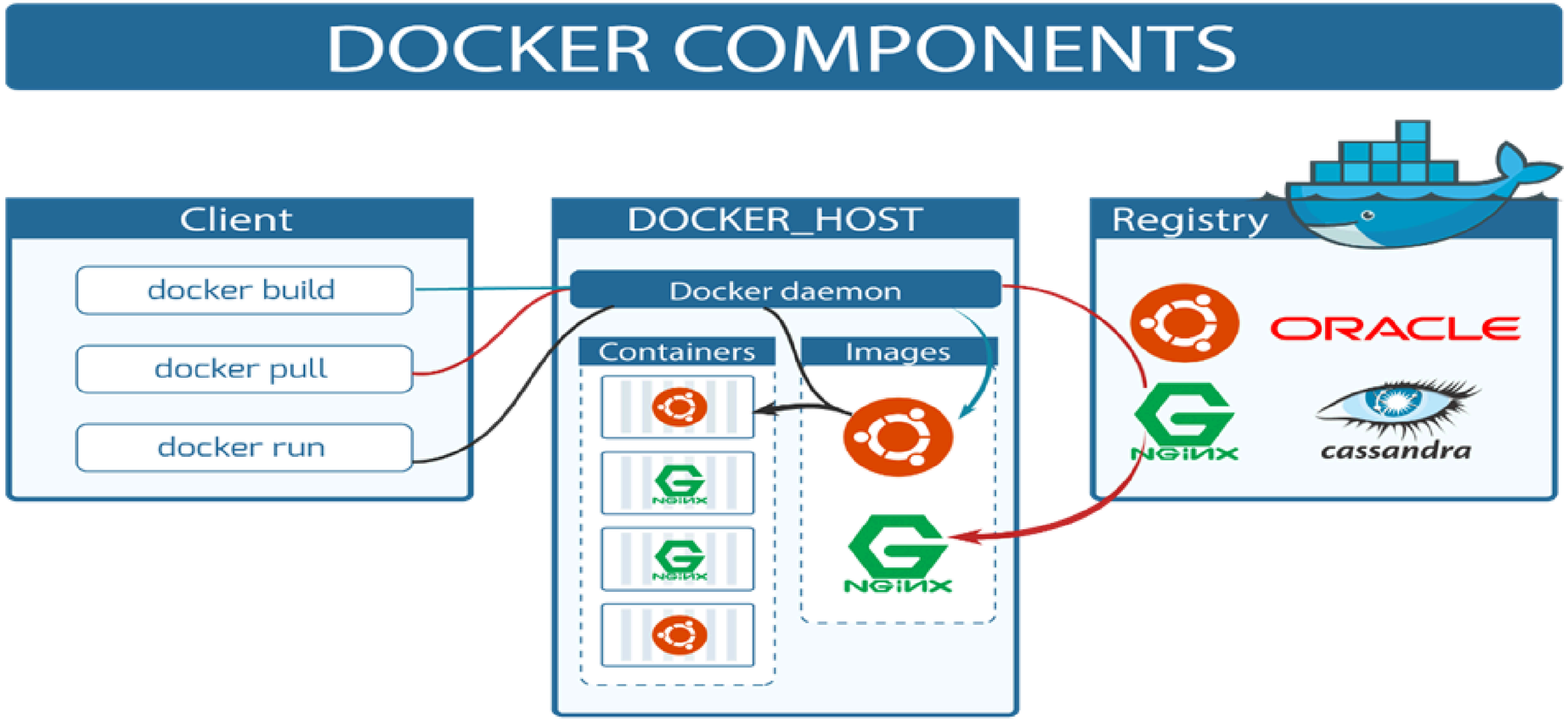


IMAGE = WAR + JAVA + TOMCAT + MYSQL

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 O.S to run applications. It allows applications to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S.
- We can install Docker on any O.S but the docker engine runs natively on Linux distribution.
- Docker performs O.S level Virtualization also known as Containerization.
- Before Docker many users face problems that a particular code is running in the developer's system but not in the user system.
- It was initially released in March 2013, and developed by Solomon Hykes and Sebastian Pahl.
- Docker is a set of platform-as-a-service that use O.S level Virtualization, where as VM ware uses Hardware level Virtualization.
- Container have O.S files but its negligible in size compared to original files of that O.S.

ARCHITECTURE:



DOCKER CLIENT:

It is the 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. The docker command uses the Docker API.

DOCKER HOST:

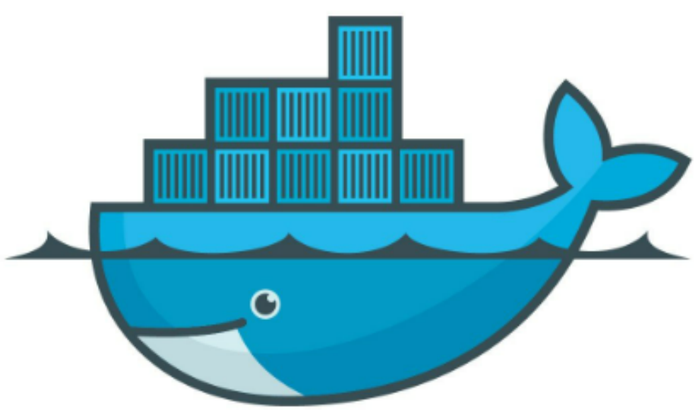
Docker host is the machine where you installed the docker engine.

DOCKER DAEMON:

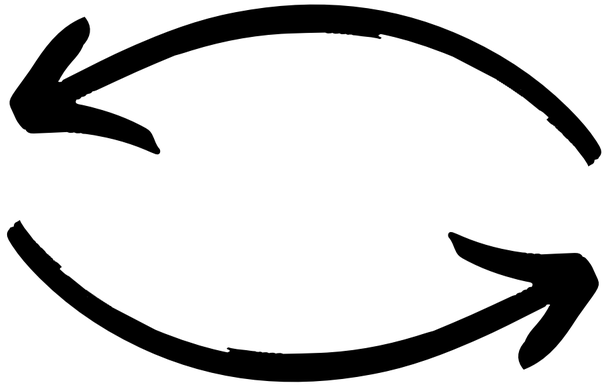
Docker daemon runs on the host operating system. It is responsible for running containers to manage docker services. Docker daemon communicates with other daemons. It offers various Docker objects such as images, containers, networking, and storage.

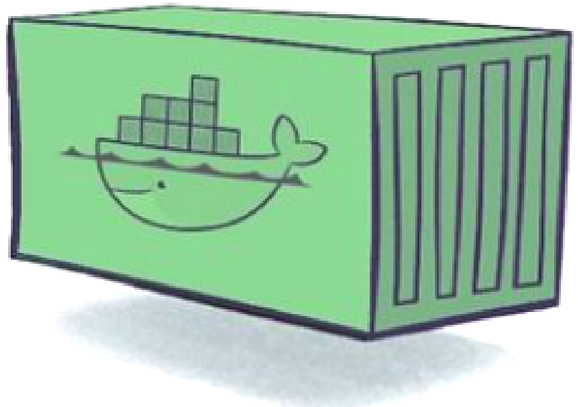
DOCKER REGISTRY:

A Docker registry is a scalable open-source storage and distribution system for docker images.



IMAGE





CONTAINER

POINTS TO BE FOLLOWED:

- You cant use docker directly, you need to start/restart first (observe the docker version before and after restart)
- You need a base image for creating a Container.
- You cant enter directly to Container, you need to start first.
- If you run an image, By default one container will create.

BASIC DOCKER COMMANDS:

To install docker in Linux : yum install docker -y
To see the docker version : docker --version
To start the docker service : service docker start
To check service is start or not : service docker status
To check the docker information : docker info
To see all images in local machine : docker images
To find images in docker hub : docker search image name
To download image from docker hub to local : docker pull image name
To download and run image at a time : docker run -it image name /bin/bash
To give names of a container : docker run -it --name raham img-name /bin/bash
To start container : docker start container name
To go inside the container : docker attach container name
To see all the details inside container : cat /etc/os-release
To get outside of the container : exit
To see all containers : docker ps -a
To see only running containers : docker ps (ps: process status)
To see only exited containers: docker ps -q -f "state=exited"
To stop the container : docker stop container name
To delete container : docker rm container name
To stop all the containers : docker stop $(docker ps -a -q)
To delete all the stopped containers : docker rm $(docker ps -a -q)
To delete all images : docker rmi -f $(docker images -q)

DOCKER RENAME:

To rename docker container: Docker rename old_container new_container

To rename docker port:

stop the container

go to path (var/lib/docker/container/container_id)

open hostconfig.json

edit port number

restart docker and start container

DOCKER EXPORT:

It is used to save the docker container to a tar file

Create a file which contains will gets stored: touch docker/password/secrets/file1.txt

TO EXPORT: docker export -o docker/password/secrets/file1.txt container_name

SYNTAX: docker export -o path container

BASIC DOCKER COMMANDS:

To see list of containers : docker container ls
To see all running containers: docker container ls -a
To see latest 2 containers : docker container ls -n 2
To see latest container : docker container ls --latest
To see all container id's : docker ls -a -q
To remove all containers : docker container rm -f $(docker container ls -aq)
To see containers with sizes : docker container ls -a -s
To stop container after some time: docker stop -t 60 cont_id

KILL VS STOP:

KILL: It passes SIGKILL signal to the container.
STOP: It passes SIGTERM signal to the container.

RUNNING A CONTAINER:

- docker run --name cont1 -d nginx
- docker inspect cont1
- curl container_private_ip:80
- docker run --name cont2 -d -p 8081(hostport):80(container port) nginx

Host Port

A port that is on the host machine that maps to the container port.

Container Port

A port that is **inside a container** where an application **listens for incoming connections**.

Port Mapping

When a **request** is made to the **host port**, it is **forwarded to** the **container port**. This makes the processes running inside the container **reachable from outside**.

docker exec:

- syntax - docker exec cont_name command
- ex-1: docker exec cont1 ls
- ex-2: docker exec cont mkdir devops
- to enter into container: docker exec -it cont_name /bin/bash **or** docker exec -it cont_name bash

LIMITS TO CONTAINER

It is used to set a memory limits to containers

docker run -dit --name cont_name --memory=250m --cpus="0.25" image_name

to check: docker inspect cont_name | grep -i memory

to check: docker inspect cont_name | grep -i nanocpu

CREATE IMAGE FROM CONTAINER:

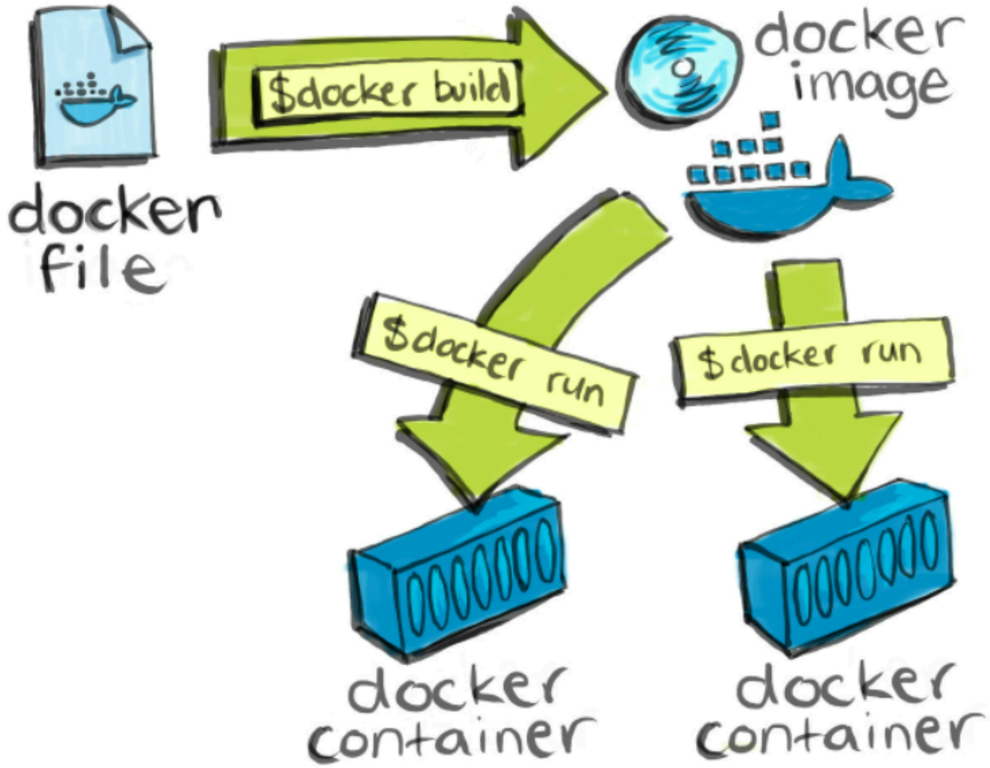
- First it should have a base image - docker run nginx
- Now create a container from that image - docker run -it --name container_name image_name /bin/bash
- Now start and attach the container
    - go to tmp folder and create some files (if you want to see the what changes has made in that image - docker diff container_name)
- exit from the container
- now create a new image from the container - docker commit container_name new_image_name
- Now see the images list - docker images
- Now create a container using the new image
- start and attach that new container
- see the files in tmp folder that you created in first container.

DOCKER FILE:

- It is basically a text file which contains some set of instructions.
- Automation of Docker image creation.
- Always D is capital letters on Docker file.
- And Start Components also be Capital letter.

HOW IT WORKS:

- First you need to create a Docker file
- Build it
- Create a container using the image



DOCKER FILE COMPONENTS:

FROM: For base image this command must be on top of the file. Ex: ubuntu, Redis, Jenkins
LABEL: Labeling like EMAIL, AUTHOR, etc.
RUN: To execute commands during image creation.
COPY: Copy files/folders from local system (docker VM) where need to provide Source and Destination.
ADD: It can download files from the internet and also, we can extract files at docker image side.
EXPOSE: To expose ports such as 8080 for tomcat and port 80 nginx etc.
WORKDIR: To set working directory for the Container.
CMD: Executes commands but during Container creation.
ENTRYPOINT: The command that executes inside of a container. like running the services in a container.
ENV: Environment Variables.

ARG argument is not available inside the Docker containers and
ENV argument is accessible inside the container

RUN: it is used to execute the commands while we build the images and add a new layer into the image.
CMD: it is used to execute the commands when we run the container.
if we have multiple CMD's only last one will gets executed.
ENTRYPOINT: it overwrites the CMD when you pass additional parameters while running the container.

COPY: Used to copy local files to containers
ADD: Used to copy files form internet and extract them

STOP: attempts to gracefully shutdown container, issues a SIGTERM signal to the main process.
KILL: immediately stops/terminates them, while docker kill (by default) issues a SIGKILL signal.

EXPOSE vs. Publish

EXPOSE	Publish
docker run command: docker run --expose <Container_PORT> Dockerfile: EXPOSE <Container_PORT>	docker run command: docker run -p <HOST_PORT>:<Container_PORT> docker compose: ports: <HOST_PORT>:<Container_PORT>
internal access	external access

DOCKER FILE TO CREATE AN IMAGE:

FROM: ubuntu
RUN: touch aws devops linux

FROM: ubuntu
RUN: touch aws devops linux
RUN echo "hello world">/tmp/file1

TO BUILD: docker build -t image_name . (. represents current directory)

Now see the image and create a new container using this image.
go to container and see the files that you created.

```
FROM ubuntu
WORKDIR /tmp
RUN echo "hello world!" > /tmp/testfile
ENV myname raham
COPY testfile1 /tmp
ADD test.tar.gz /tmp
```

```
[root@ip-172-31-83-27 ~]# touch testfile1
[root@ip-172-31-83-27 ~]# touch test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1
[root@ip-172-31-83-27 ~]# tar -cvf test.tar test
test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar
[root@ip-172-31-83-27 ~]# gzip test.tar
[root@ip-172-31-83-27 ~]# ls
Dockerfile  test  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# rm -rf test
[root@ip-172-31-83-27 ~]# ls
Dockerfile  testfile1  test.tar.gz
[root@ip-172-31-83-27 ~]# docker build -t raham .
```

docker run -it - -name container name image-name /bin/bash

```
FROM ubuntu:14.04
MAINTAINER abc "abc@abc.com"
RUN apt-get update
RUN apt-get install -y nginx
RUN echo 'Our first Docker image for Nginx' > /usr/share/nginx/html/index.html
EXPOSE 80
~
~
```

TO BUILD:

docker build -t image1 .

TO RUN:

docker run -dit --name mustafa -p 8081:80 image1 nginx -g "daemon off;"

DOCKER VOLUMES:

- When we create a Container then Volume will be created.
- Volume is simply a directory inside our container.
- First, we have to declare the directory Volume and then share Volume.
- Even if we stop/delete the container still, we can access the volume.
- You can declare directory as a volume only while creating container.
- We can't create volume from existing container.
- You can share one volume across many number of Containers.
- Volume will not be included when you update an image.
- If Container-1 volume is shared to Container-2 the changes made by Container-2 will be also available in the Container-1.

- You can map Volume in two ways:

1. Container < ------ > Container
2. Host < ------- > Container

USES OF VOLUMES:

- Decoupling Container from storage.
- Share Volume among different Containers.
- Attach Volume to Containers.
- On deleting Container Volume will not be deleted.

CREATING VOLUMES FROM DOCKER FILE:

- Create a Docker file and write
  FROM ubuntu
  VOLUME["/myvolume"]
- build it - docker build -t image_name .
- Run it - docker run -it - -name container1 ubuntu /bin/bash
- Now do ls and you will see myvolume-1 add some files there
- Now share volume with another Container - docker run -it - -name container2(new) - -privileged=true - -volumes-from container1 ubuntu
- Now after creating container2, my volume1 is visible
- Whatever you do in volume1 in container1 can see in another container
- touch /myvolume1/samplefile1 and exit from container2.
- docker start container1
- docker attach container1
- ls/volume1 and you will see your samplefile1

```
FROM ubuntu
ADD file1 /ubuntu1/file
VOLUME /ubuntu1
~
```

CREATING VOLUMES FROM COMMAND:

- docker run -it - -name container3 -v /volume2 ubuntu /bin/bash
- now do ls and cd volume2.
- Now create one file and exit.
- Now create one more container, and share Volume2 - docker run-it - -name container4 - - -privileged=true - -volumes-from container3 ubuntu
- Now you are inside container and do ls, you can see the Volume2
- Now create one file inside this volume and check in container3, you can see that file

VOLUMES (HOST TO CONTAINER):

- Verify files in /home/ec2-use
- docker run -it - -name hostcont -v /home/ec2-user:/raham - -privileged=true ubuntu
- cd raham [raham is (container-name)]
- Do ls now you can see all files of host machine.
- Touch file1 and exit. Check in ec2-machine you can see that file.

SOME OTHER COMMANDS:

- docker volume ls
- docker volume create <volume-name>
- docker volume rm <volume-name>
- docker volume prune (it will remove all unused docker volumes).
- docker volume inspect <volume-name>
- docker container inspect <container-name>
- docker system df -v :

MOUNT VOLUMES:

- To attach a volume to a container: docker run -it --name=example1 --mount source=vol1,destination=/vol1 ubuntu
- To send some files from local to container:
  - create some files
  - docker run -it --name cont_name -v "$(pwd)":/my-volume ubuntu
- To remove the volume: docker volume rm volume_name
- To remove all unused volumes: docker volume prune

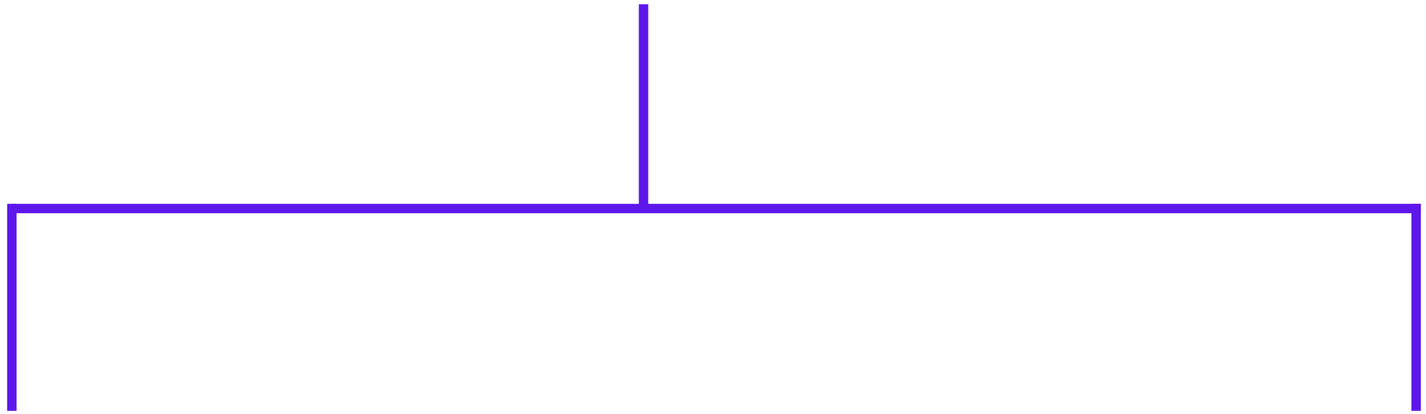
BASE VOLUMES:

STEPS

- create a volume : docker volume create volume99(volume-name)
- mount it: docker run -it -v volume99:/my-volume --name container1 ubuntu
- now go to my-volume and create some files over there and exit from container
- mount it: docker run -it -v volume99:/my-volume-01 --name container2 ubuntu

DOCKER REGISTRY:

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



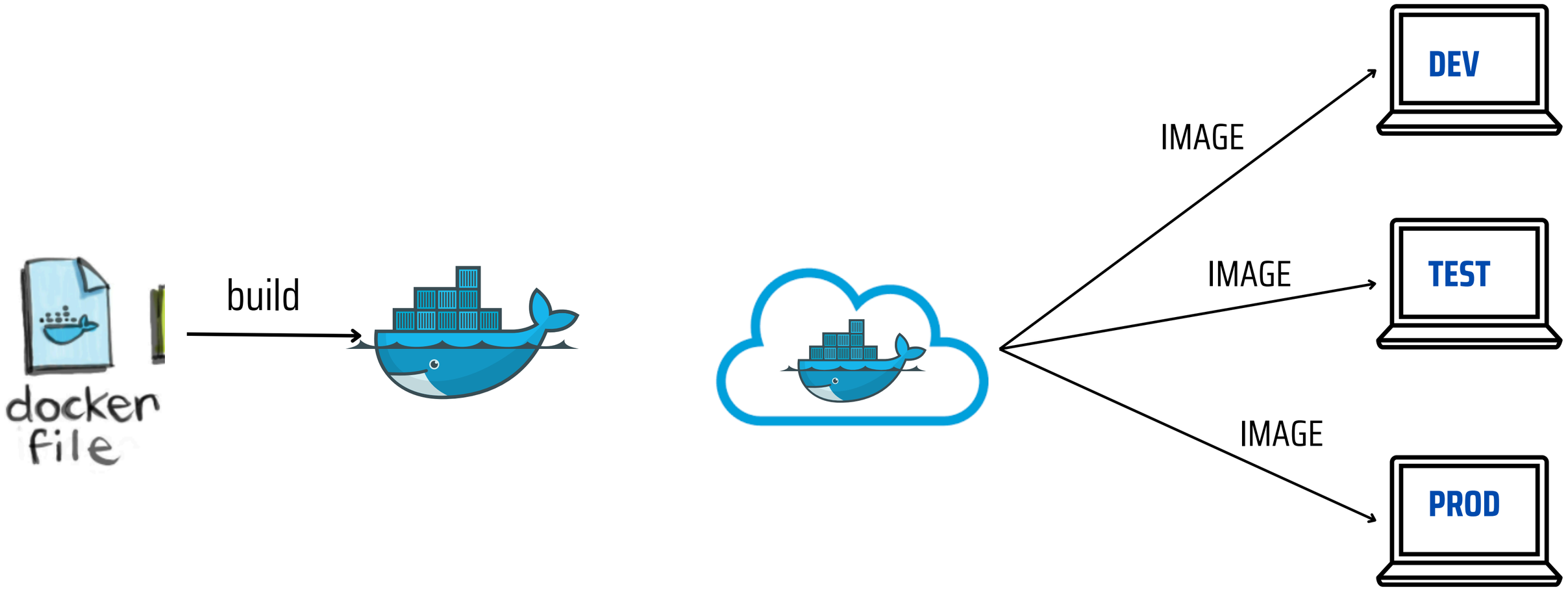
Cloud based registry

- Docker Hub
- GCR - Google Container Registry
- Amazon ECR - Elastic Container Registry

Local registry

- Nexus
- Jfrog
- DTR - Docker Trusted Registry

WHY DOCKER HUB



DOCKER PUSH:

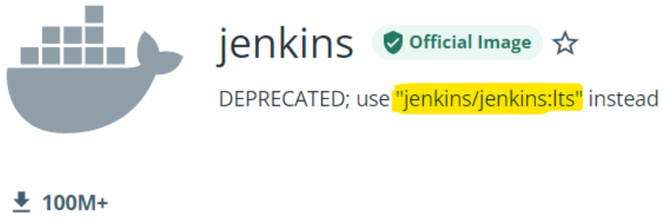
- Select an image that includes docker and S.G SSH and HTTP enable anywhere on it.
- docker run -it ubuntu /bin/bash
- Create some files inside the container and create an image from that container by using - docker commit container-name image1
- now create a docker hub account
- Go to ec2-user and log in by using docker login.
- Enter username and password.
- Now give the tag to your image, without tagging we can't push our image to docker.
- docker tag image1 rahamshaik/new-image-name (ex: project1)
- docker push rahamshaik/project1
- Now you can see this image in the docker hub account.
- Now create one instance in another region and pull the image from the hub.
- docker pull rahamshaik/project1
- docker run -it - -name mycontainer rahamshaik/project1 /bin/bash
- Now give ls and cd tmp and ls you can see the files you created.
- Now go to docker hub and select your image -- > settings -- > make it private.
- Now run docker pull rahamshaik/project1
- If it is denied then login again and run it.
- If you want to delete image settings -- > project1 -- > delete

JENKINS SETUP USING DOCKER IMAGE:

DESCRIPTION: By using the docker file, we can set up the Jenkins dashboard without installing any dependencies.

- Login to docker hub
- search for Jenkins then you will get Jenkins official image.

- Click on the image



- copy the code : docker pull jenkinsci/jenkins:lts
- run this code in docker engine
- now see docker images then you will get jenkins image
- create container using that image
- Now exit from the container and start the container again
- Inspect the container : docker inspect jenkins
- now go to jenkins image in docker hub and scroll down you will get command

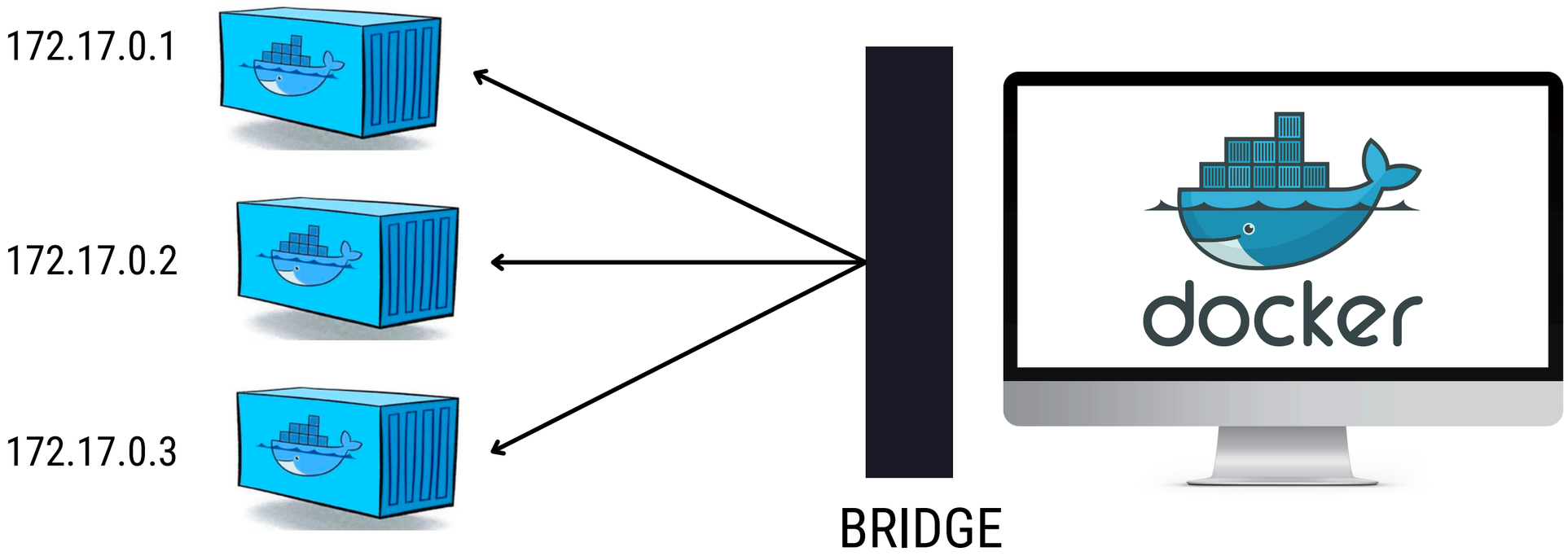
Jenkins

How to use this image

```
docker run -p 8080:8080 -p 50000:50000 jenkins
```

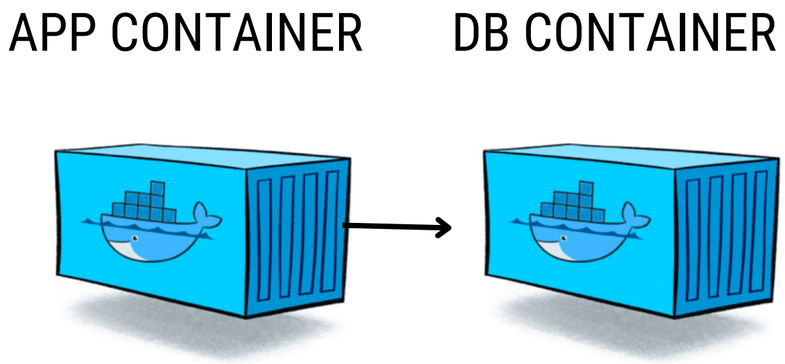
- run this command on docker engine and connect with Jenkins dashboard (public ip:8080)

DOCKER NETWORK:



WHY DOCKER NETWORK:

Lets assume we are having 2 containers like APP container 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.



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 using docker networks to create our custom network.

Docker networks are used to make a communication between the multiple containers that are running on same or different docker hosts. We have different types of docker networks.

- Bridge Network
- Host Network
- None network
- Overlay network

BRIDGE NETWORK: It is a default network that container will communicate with each other within the same host.

HOST NETWORK: When you Want your container IP and ec2 instance IP same then you use host network

NONE NETWORK: When you don’t Want The container to get exposed to the world, we use none network. It will not provide any network to our container.

OVERLAY NETWORK: Used to communicate containers with each other across the multiple docker hosts.

To create a network: docker network create network_name

To see the list: docker network ls

To delete a network: docker network rm network_name

To inspect: docker network inspect network_name

To connect a container to the network: docker network connect network_name container_id/name

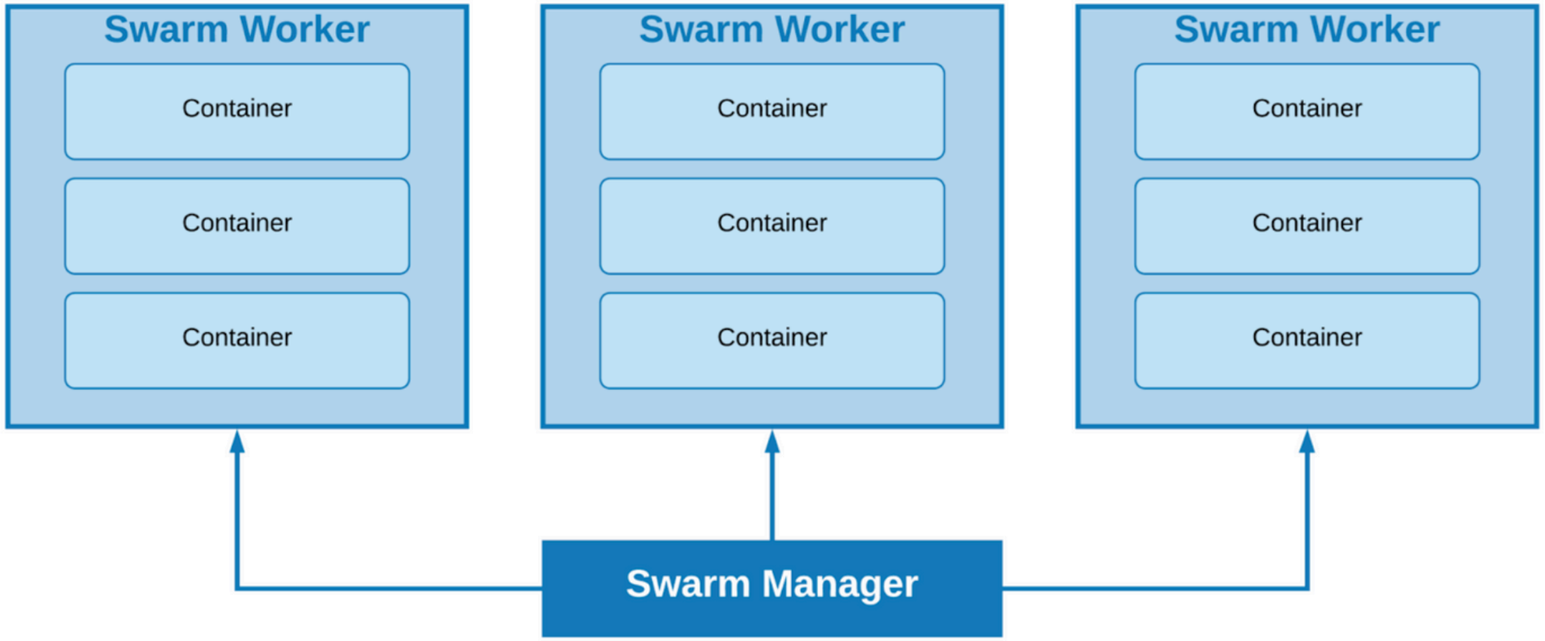
apt install iputils-ping -y : command to install ping checks

To disconnect from the container: docker network disconnect network_name container_name

To prune: docker network prune

DOCKER SWARM:

- Docker swarm is an orchestration service within docker that allows us to manage and handle multiple containers at the same time.
- It is a group of servers that runs the docker application.
- It is used to manage the 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.



- Docker Engine helps to create Docker Swarm.
- There are mainly worker nodes and manager nodes.
- The worker nodes are connected to the manager nodes.
- So any scaling or update needs to be done 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.

DOCKER SWARM COMPONENTS:

SERVICE: Represents a part of the feature of an application.
TASK: A single part of work.
MANAGER: This manages the work among the different nodes.
WORKER: Which works for a specific purpose of the service.

SETUP:

Create 3 node one is manager and another two are workers
Manager node: docker swarm init --advertise-addr (private ip)
Run the below command to join the worker nodes
To check nodes on docker swarm: docker node ls
Here * Indicates the current node like master branch on git
Now we created the docker swarm cluster
docker swarm leave : To down the docker node (need to wait few sec)
docker node rm node-id : To remove the node permenantly
docker swarm leave : To delete the swarm but will get error
docker swarm leave –force : To delete the manager forcefully
docker swarm join-token worker. : To get the token of the worker
docker swarm join-token manager : To get the token of the worker

SWARM SERVICE:

Now we want to run a service on the swarm
So we want to run a specific container on all these nodes
To do that we will use a docker service command which will create a service for us
That service is nothing but a container.
We have a replicas here when one replica goes down another will work for us.
At least one of the replica needs to be up among them.

docker service create --name raham --replicas 3 --publish 80:80 httpd
raham : service name replicas : nodes publish : port reference image: apache

docker service ls : To list the services
docker service ps service-name : To see where the services are running
docker ps : To see the containers (Check all nodes once)
docker service rm service_name : To remove the service (it will come again later)
public ip on browser : To check its up and running or not
docker service rm service-name : To remove the service

To create a service: docker service create —name devops —replicas 2 image_name

Note: image should be present on all the servers

To update the image service: docker service update —image image_name service_name

Note: we can change image,

To rollback the service: docker service rollback service_name

To scale: docker service scale service_name=3

To check the history: docker service logs

To check the containers: docker service ps service_name

To inspect: docker service inspect service_name

To remove: docker service rm service_name

DOCKER COMPOSE:

- It 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.
- It used YAML file to manage multi containers as a single service.
- The Compose file provides a way to document and configure all of the application’s service dependencies (databases, queues, caches, web service APIs, etc).

COMMANDS:

- Start all services: Docker Compose up.
- Stop all services: Docker Compose down.
- Run Docker Compose file: Docker-compose up -d.
- List the entire process: Docker ps.

COMPOSE FILE:

The Docker Compose file includes Services, Networks and Volumes.
The Default Path is ./docker-compose.yml, compose.yml (yaml=yml)
It contains a service definition which configures each container started for that service.

COMPOSE INSTALLATION:

- sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
- ls /usr/local/bin/
- sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose
- docker-compose version

COMPOSE FILE:

- version - specifies the version of the Compose file.
- services - it the services in your application.
- networks - you can define the networking set-up of your application.
- volumes - you can define the volumes used by your application.
- configs - configs lets you add external configuration to your containers. Keeping configurations external will make your containers more generic.

CREATING DOCKER-COMPOSE.YML:

```
version: '3'
services:
  webapp1:
    image: nginx
    ports:
      - "8000:80"
```

vim docker-compose.yml

Version: It is the compose file format which supports the relavent docker engine

Services: The services that we are going to use by this file (Webapp1 is service name)

Image: Here we are taking the Ngnix image for the webserver

Ports: 8000 port is mapping to container port 80

Docker-compose up -d

Public-ip:8000 -- > You can see the Nginx image

Docker network ls -- > you can see root_default

Docker-compose down -- > It will delete all the Created containers

```yaml
version: '3'
services:
  webapp1:
    image: nginx
    ports:
      - "8000:80"
  webapp2:
    image: nginx
    ports:
      - "8001:80"
```

Docker-compose up -d

Public-ip:8000 & public-ip:8001-- > You can see the Nginx image on both ports

Docker container ls

Docker network ls

CHANGING DEFAULT FILE:

mv docker-compose.yml docker-compose1.yml

docker-compose up -d

You will get some error because you are changing by default docker-compose.yml

Use the below command to overcome this error

docker-compose -f docker-compose1.yml up -d

docker-compose -f docker-compose1.yml down

```
version: "3.1"
services:
  mobile_recharge:
    image: nginx
    ports:
      - "9999:80"
    volumes:
      - "mobile-recharge-volume1"
    networks:
      - "mobile-recharge-network"

  mobile_recharge2:
    image: nginx
    ports:
      - "9998:80"
networks:
  mobile-recharge-network:
   driver: bridge
~
```

docker-compose up -d - used to run the docker file

docker-compose build - used to build the images

docker-compose down - remove the containers

docker-compose config - used to show the configurations of the compose file

docker-compose images - used to show the images of the file

docker-compose stop - stop the containers

docker-compose logs - used to show the log details of the file

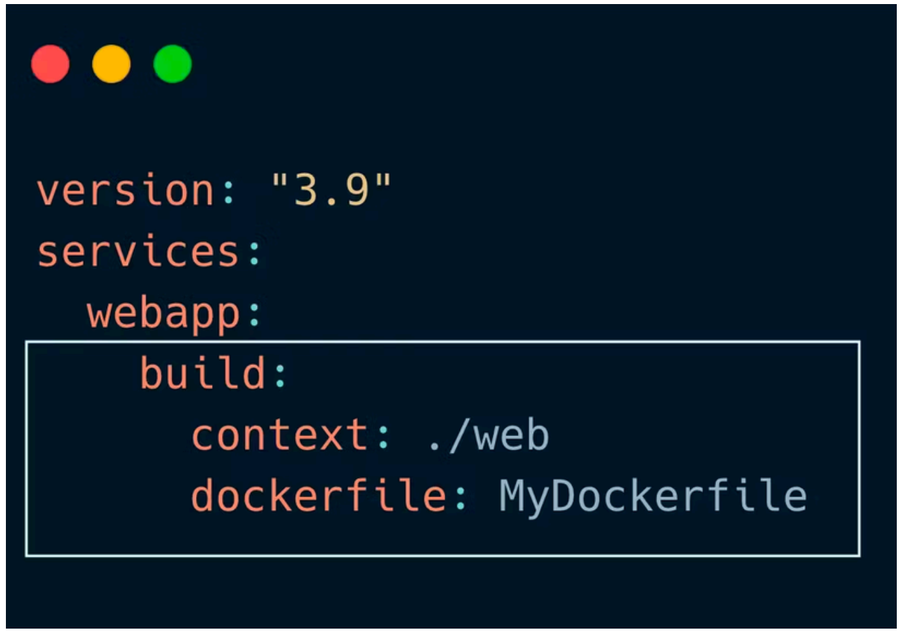
docker-compose pause - to pause the containers

docker-compose unpause - to unpause the containers

docker-compose ps - to see the containers of the compose file

6 Must-Know Docker Compose Tips

1. **Customizing Dockerfile Name and Path:** We can easily customize the name and directory of our Dockerfile using the context and Dockerfile options in Docker Compose. By default, Docker Compose looks for a Dockerfile named Dockerfile in the root directory of the context.



2. Using .env File in Docker Compose: If you're working with environment variables in Docker Compose, it's best to use the env_file option instead of hardcoding them directly in the Compose file. This approach will keep your sensitive data secure and make your Compose file more reusable.

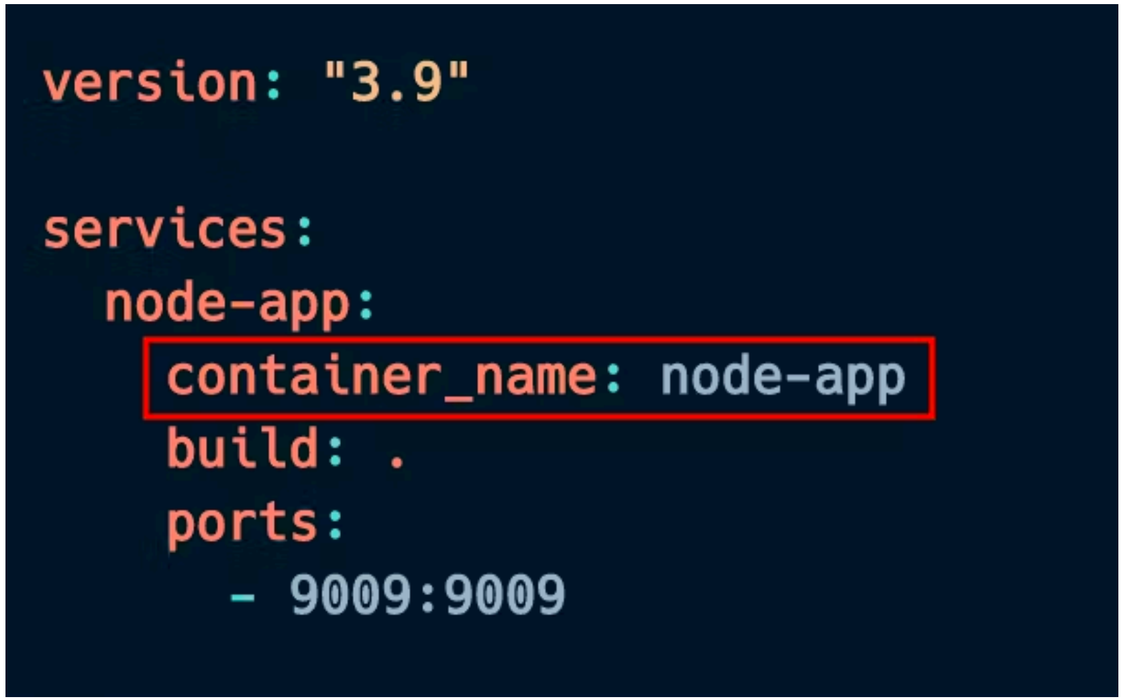
3. Building Images with Docker Compose: Not only can you use Compose to run multiple services, but you can also use it to build images and save time typing long commands on the command line with the build command.

```
docker compose build
```

4. Use the 'restart: always' option in your Docker Compose file. If the container stops for any reason, Docker Compose will automatically restart it.

```
version: "3.9"
services:
  web:
    image: nginx:latest
    restart: always
    ports:
      - "80:80"
```

5. Assigning Container Names: By default, Docker Compose assigns random names to containers created with it. However, you can assign more meaningful names to your containers by using the "container_name" property in your Docker Compose file. This allows for easier identification and management of your containers.



Depends on Property: If a container depends on another container, and the latter needs to run first, use the "depends_on" property. This ensures that the Mongo container runs before the Node app container. NOTE: This does not guarantee that the container will start first, as it depends on various factors.

```yaml
version: "3.9"

services:

  node-app:
    build: .
    ports:
      - 9009:9009
    depends_on:
      - mongo

  mongo:
    image: mongo:5.0
```

DOCKER STACK:

- Docker stack is used to create multiple services on multiple hosts. i.e it will create multiple containers on multile 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
- Ex: Lets 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

COMMAND:

TO DEPLOY : docker stack deploy --compose-file docker-compose.yml stack_name

TO LIST : docker stack ls

TO GET CONTAINERS OF A STACK : docker stack ps stack_name

TO GET SERVICES OF A STACK: docker stack services stack_name

TO DELETE A STACK: docker stack rm stack_name

DOCKER INTEGRATION WITH JENKINS

- Install docker and Jenkins in a server.
- vim /lib/systemd/system/docker.service

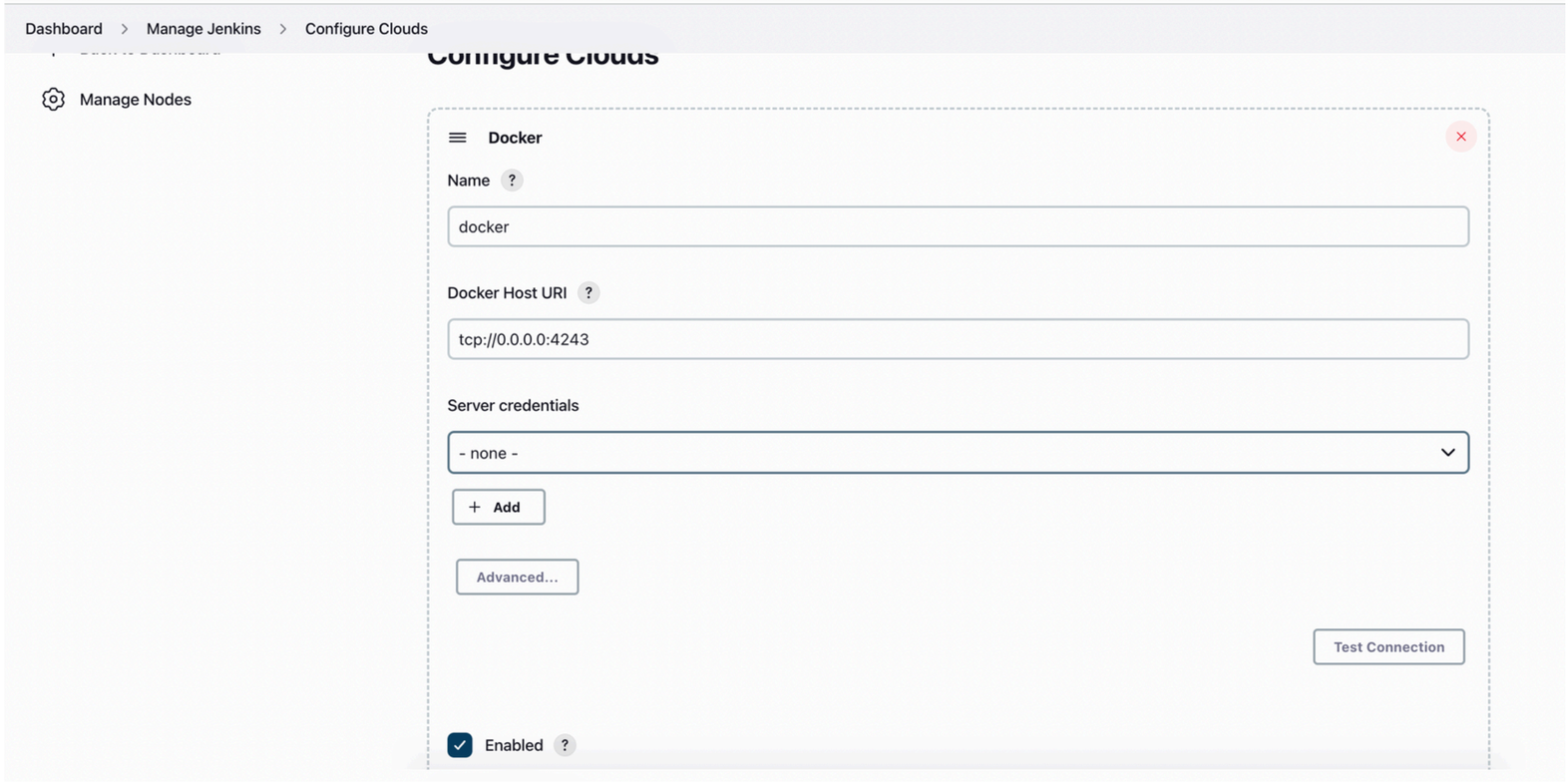
```
[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always
```

- Replace the above line with

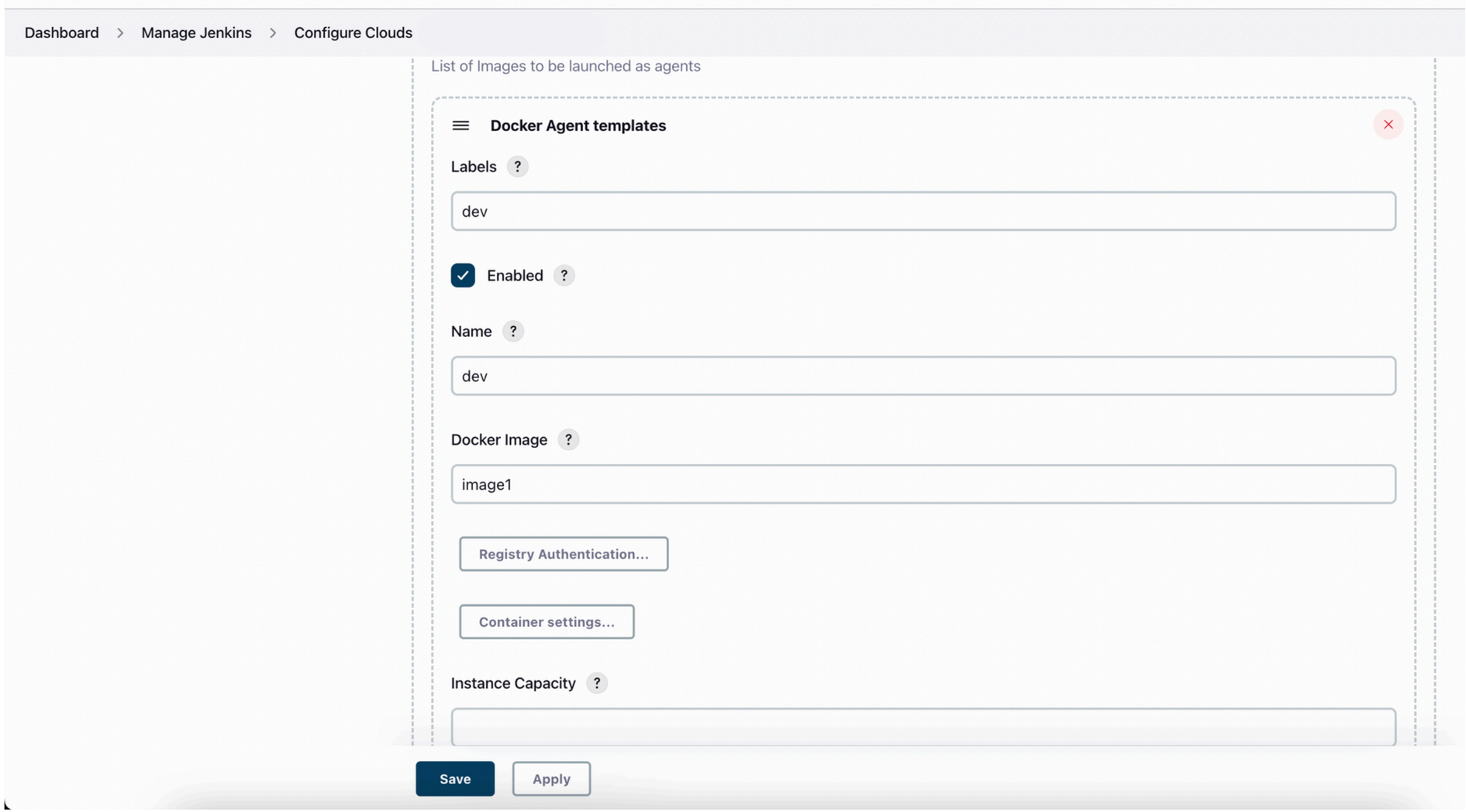
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock

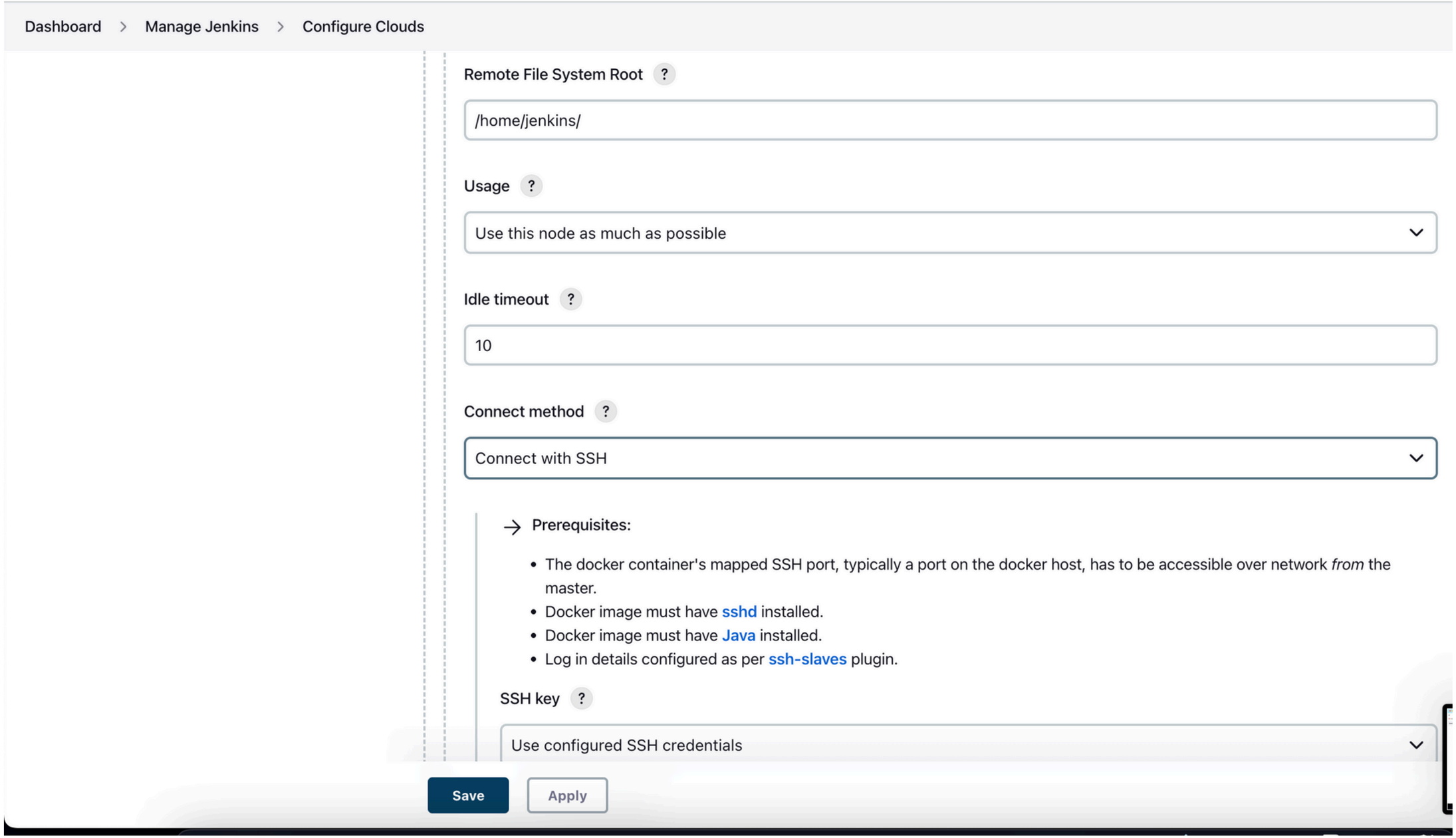
- systemctl daemon-reload
- service docker restart
- curl http://localhost:4243/version

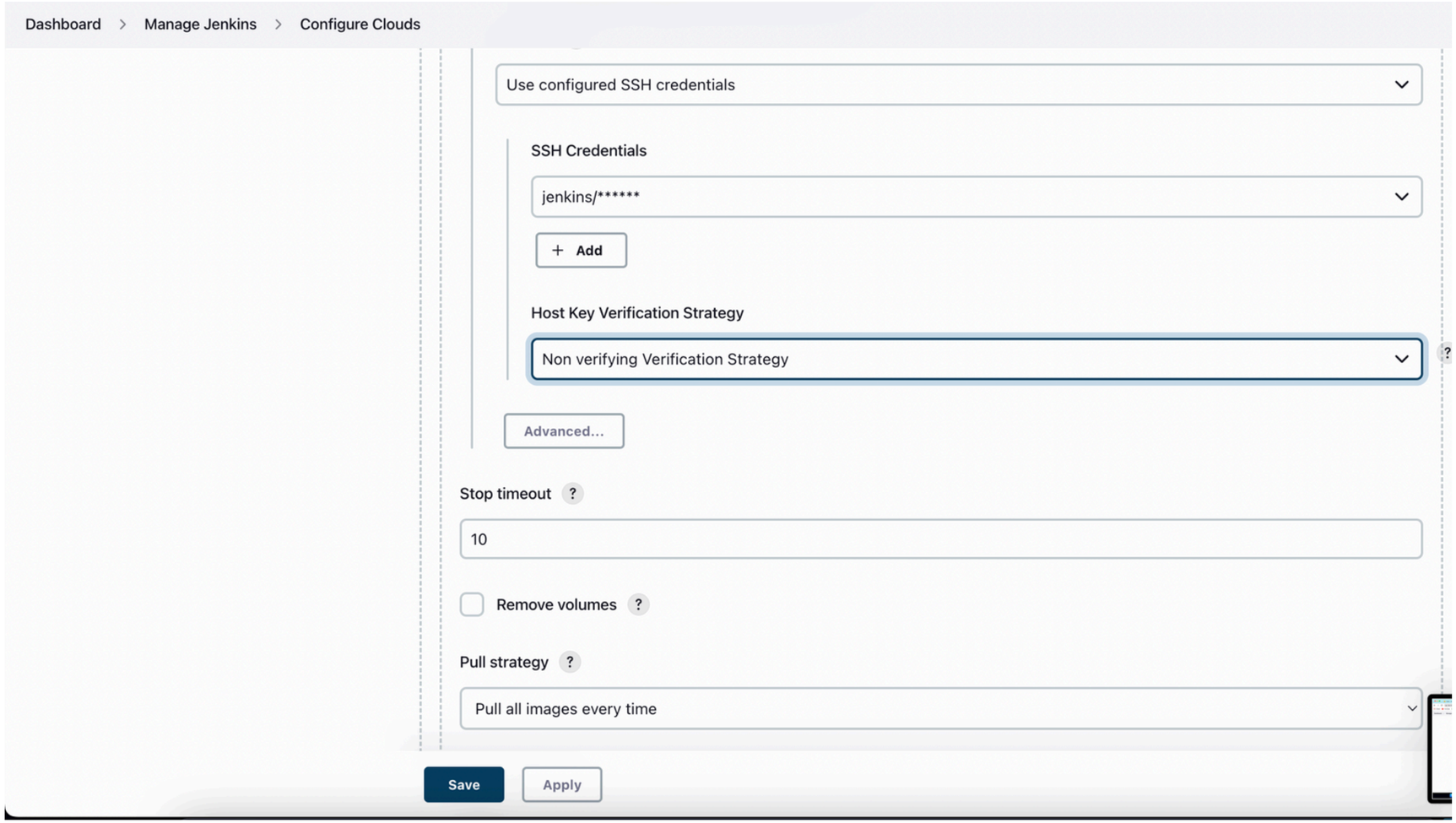
- Install Docker plugin in Jenkins Dashboard.
- Go to manage jenkins>Manage Nodes & Clouds>>Configure Cloud.
- Add a new cloud >> Docker
- Name: Docker
- add Docker cloud details.



- Add Docker Agent Template




Dashboard > Manage Jenkins > Configure Clouds
Remote File System Root ?
/home/jenkins/
Usage ?
Use this node as much as possible
Idle timeout ?
10
Connect method ?
Connect with SSH
Prerequisites:
The docker container's mapped SSH port, typically a port on the docker host, has to be accessible over network from the master.
Docker image must have sshd installed.
Docker image must have Java installed.
Log in details configured as per ssh-slaves plugin.
SSH key ?
Use configured SSH credentials
Save
Apply



- Save it and do and watch the container in Jenkins dashboard.
- Manage Jenkins>>Docker (last option)

DEPLOYMENT DOCKER FILE:

Create 2 files:

1. Dockerfile
2. index.html file

Dockerfile consists of

```
FROM UBUNTU
RUN APT-GET UPDATE -Y
RUN APT-GET INSTALL APACHE2 -Y
COPY INDEX.HTML /VAR/WWW/HTML/
CMD ["/USR/SBIN/APACHECTL", "-D", "FOREGROUND"]
```

Index.html file consists of

```
<H1>HI THIS IS MY WEB APP</H1>
```

Add these files into GitHub and Integrate with Jenkins by declarative code pipeline.

```
pipeline {
    agent any
    stages {
        stage ("git") {
            steps {
                git branch: 'main', url: 'https://github.com/devops0014/dockabnc.git'
            }
        }
        stage ("build") {
            steps {
                sh 'docker build -t image77 .'
            }
        }
        stage ("container") {
            steps {
                sh 'docker run -dit -p 8077:80 image77'
            }
        }
    }
}
```

You will get Permission Denied error while building the code.
To resolve that error you need to follow these steps:

- usermod -aG docker jenkins
- usermod -aG root jenkins
- chmod 777 /var/run/docker.sock
- systemctl daemon-reload

Now you can build the code and it will gets deployed.

DOCKER DIRECTORY DATA:

We use docker to run the images and create the containers. but what if the memory is full in instance. we have a add a another volume to the instance and mount it to the docker engine. Lets see how we do this.

- Uninstall the docker - **yum remove docker -y**
- remove all the files - rm -rf /var/lib/docker/*
- create a volume in same AZ & attach it to the instance
- to check it is attached or not - fdisk -l
- to format it - fdisk /dev/xvdf --> n p 1 enter enter w
- set a path - vi /etc/fstab (/dev/xvdf1 /var/lib/docker/ ext4 defaults 0 0)
- mkfs.ext4 /dev/xvdf1
- mount -a
- install docker - yum install docker -y && systemctl restart docker
- now you can see - ls /var/lib/docker
- df -h

PORTAINER:

- it is a container organizer, designed to make tasks easier, whether they are clustered or not.
- abel 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 two elements, the Portainer Server and the Portainer Agent.
- Both elements run as lightweight Docker containers on a Docker engine

PORTAINER:

- Must have swarm mode and all ports enable with docker engine
- curl -L https://downloads.portainer.io/ce2-16/portainer-agent-stack.yml -o portainer-agent-stack.yml
- docker stack deploy -c portainer-agent-stack.yml portainer
- docker ps
- public-ip of swamr master:9000

DOCKER RUN VS CMD VS ENTRYPOINT:

RUN: it is used to execute the commands while we build the images and add a new layer into the image.

FROM centos:centos7
RUN yum install git -y
 or
RUN [“yum”, “install”, “git” “-y”]

CMD: it is used to execute the commands when we run the container.
    It is used to set the default command.
    if we have multiple CMD’s only last one will gets executed.

FROM centos:centos7
CMD yum install maven -y
 or
CMD [“yum”, “install”, “maven”, “-y”]

If you want to overwrite the parameters:
 docker run image_name httpd (FAILED)
 docker run image_name yum install httpd -y (only httpd will gets installed)

ENTRYPOINT: it overwrites the CMD when you pass additional parameters while running the container.

FROM centos:centos7
ENTRYPOINT [“yum”, “install”, “maven”, “-y”]

If you want to overwrite the parameters:
 docker run image_name httpd (both maven and httpd will gets installed)
 docker run image_name yum install httpd -y (both maven and httpd will gets installed)

FROM centos:centos7
ENTRYPOINT [“yum”, “install”, “-y”]
CMD [“httpd”]

Bydefault it will executes httpd command, if you specify the command while running the container it will gets executed.
 docker run image_name (httpd will install)
 docker run image_name git (only git will install)
 docker run image_name git tree(both git & tree will install)

DOCKER FILE TO DEPLOY STATIC WEBSITE USING NGINX:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install nginx -y
COPY index.html /var/www/html/
EXPOSE 80
CMD ["nginx","-g","daemon off;"]
~
~
```

OR

```
FROM nginx
COPY . /usr/share/nginx/html
~
~
~
~
~
```

REFERENCE: **https://github.com/devops0014/staticsite-docker.git**

TIC-TAC-TOE GAME

REFERENCE: **https://github.com/devops0014/tic-tac-toe-docker.git**

PROJECT: USE DOCKER STACK FOR PROJECT

Take 2 nodes 1 manger & 1 worker & setup jenkins in manager

setup a cluster of docker swarm

create images and upload to docker hub

pull images to both nodes (opt)

write a compose file with replicas

write a pipeline and build it

MY PROJECT LINK : https://github.com/RAHAMSHAIK007/evedockerproject.git

memory management:

docker run -it --cpus=".5"--name cont-name ubuntu /bin/bash
docker run -d --name cont3 --memory 50M nginx : It can use now maxx 50 MB
docker stats

DOCKER FILE TO DEPLOY WAR FILE

```
FROM tomcat:8.0.20-jre8
COPY tomcat-users.xml /usr/local/tomcat/conf/
COPY target/*.war /usr/local/tomcat/webapps/myweb.war
```

DOCKER FILE TO DEPLOY NODE JS FILE:

```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8081
CMD ["node", "index.js"]
```

```
#Stage One: Development Application
FROM node:19-alpine AS base
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD npm start

#Stage Two: Create the Production image
FROM base AS final
RUN npm install --production
COPY . .
CMD npm start
```

REFERENCE: **https://github.com/devops0014/nodejs-docker.git**

DOCKER COMPOSE FILE TO DEPLOY DOCKER-APP:

```yaml
version: "3"
services:
  db:
    image: shaikmustafa/docker-app:backend
    container_name: devopsdb
    ports:
      - "3306:3306"
    environment:
      - MYSQL_ROOT_PASSWORD=devopspassword
  web:
    image: shaikmustafa/docker-app:frontend
    ports:
      - "8080:8080"
    depends_on:
      - db
```

CONTAINER LOAD BALANCER:

```
.
├── docker-compose.yml
├── nginx
│   ├── Dockerfile
│   └── nginx.conf
├── swiggy
│   ├── Dockerfile
│   └── index.html
└── zomato
    ├── Dockerfile
    └── index.html
```

DOCKER FILE FOR SWIGGY AND ZOMATO SERVICES

```
FROM nginx
COPY . /usr/share/nginx/html
~
~
```

DOCKER FILE FOR NGINX

```
FROM nginx
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d/default.conf
~
```

NGINX.CONF FILE

```
upstream loadbalancer {
server 172.17.0.1:5001 weight=6;
server 172.17.0.1:5002 weight=4;
}
server {
location / {
proxy_pass http://loadbalancer;
}}
```

COMPOSE FILE

```
version: '3'
services:
  swiggy:
    image: image1
    ports:
      - "5001:80"
  zomato:
    image: image2
    ports:
      - "5002:80"
  nginx:
    build: ./nginx
    ports:
      - "8080:80"
    depends_on:
      - swiggy
      - zomato
~
```

After writing all the file, we need to build the Dockerfiles for both the zomato and swiggy services.
command: **docker build -t image_name .**

write the docker-compose file and build it.
command: **docker-compose up -d**

access the application: **publicip:8080**

DOCKER SAVE:

```
docker image save swiggy:v1 > swiggy:v1.tar
docker image history swiggy:v1
docker rmi swiggy:v1
docker images
docker image load < swiggy\:v1.tar
```

save vs export