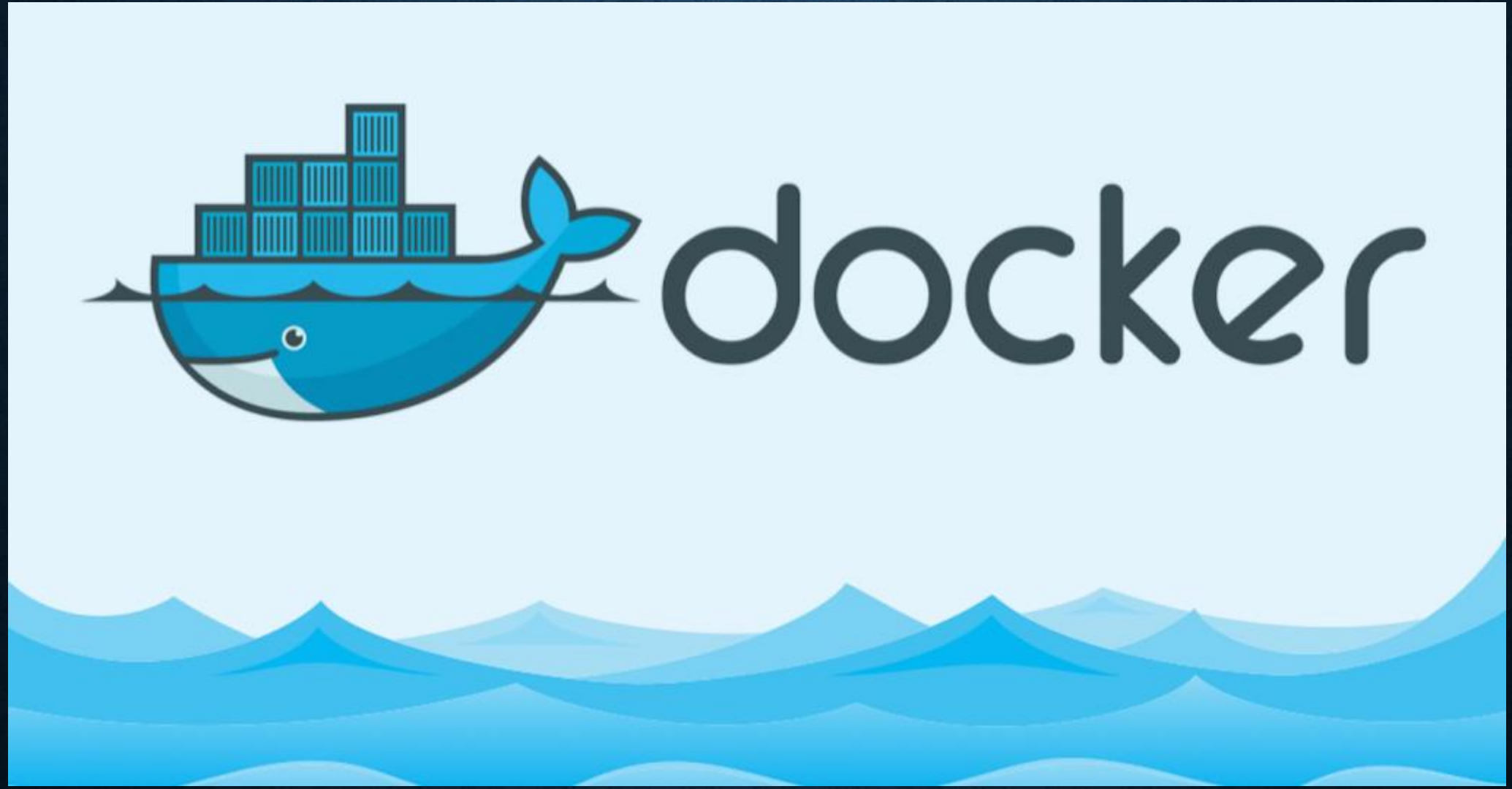
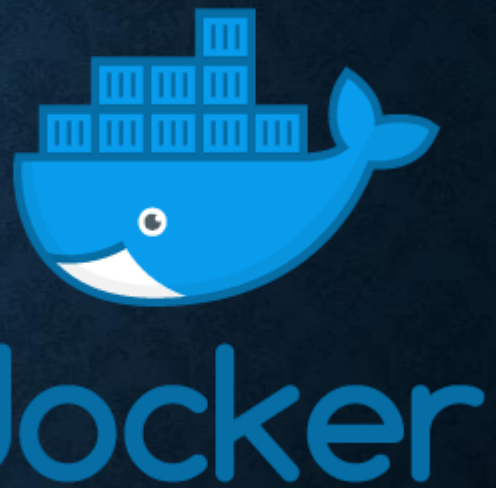

docker

Important differences?

- **Container** : A container like a virtual machine
- **Docker** : Docker is a tool to create those virtual machines

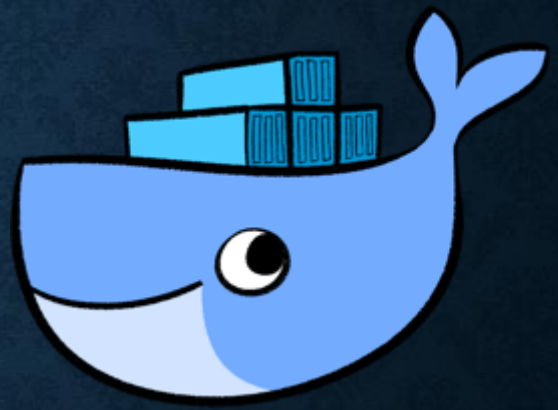
What is Docker??

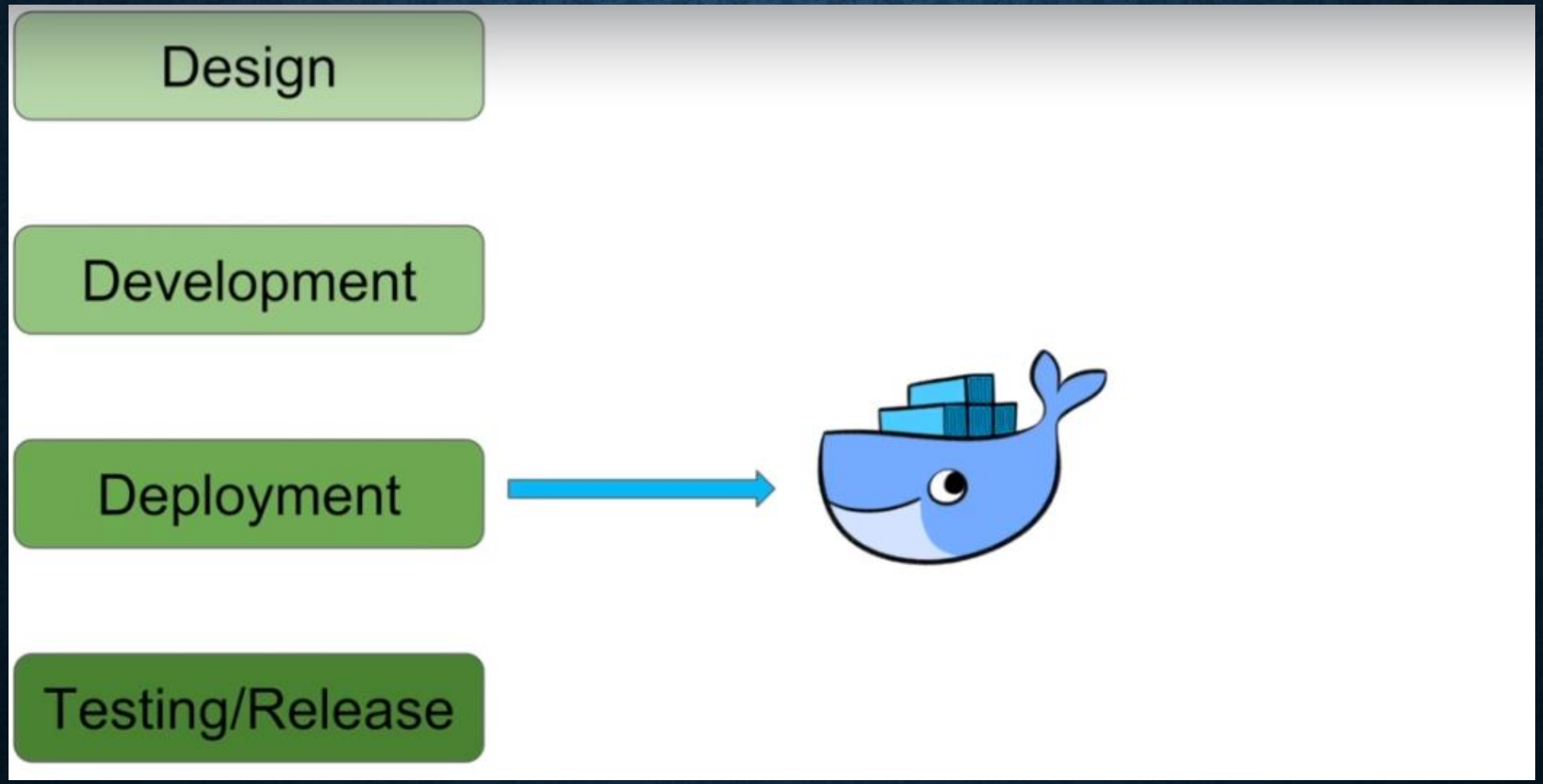
- Docker is a tool that performs operating-system-level virtualization, also known as "containerization". It was first released in 2013 and was developed by Docker, Inc
- Docker is tool used to create virtual machines called "containers".

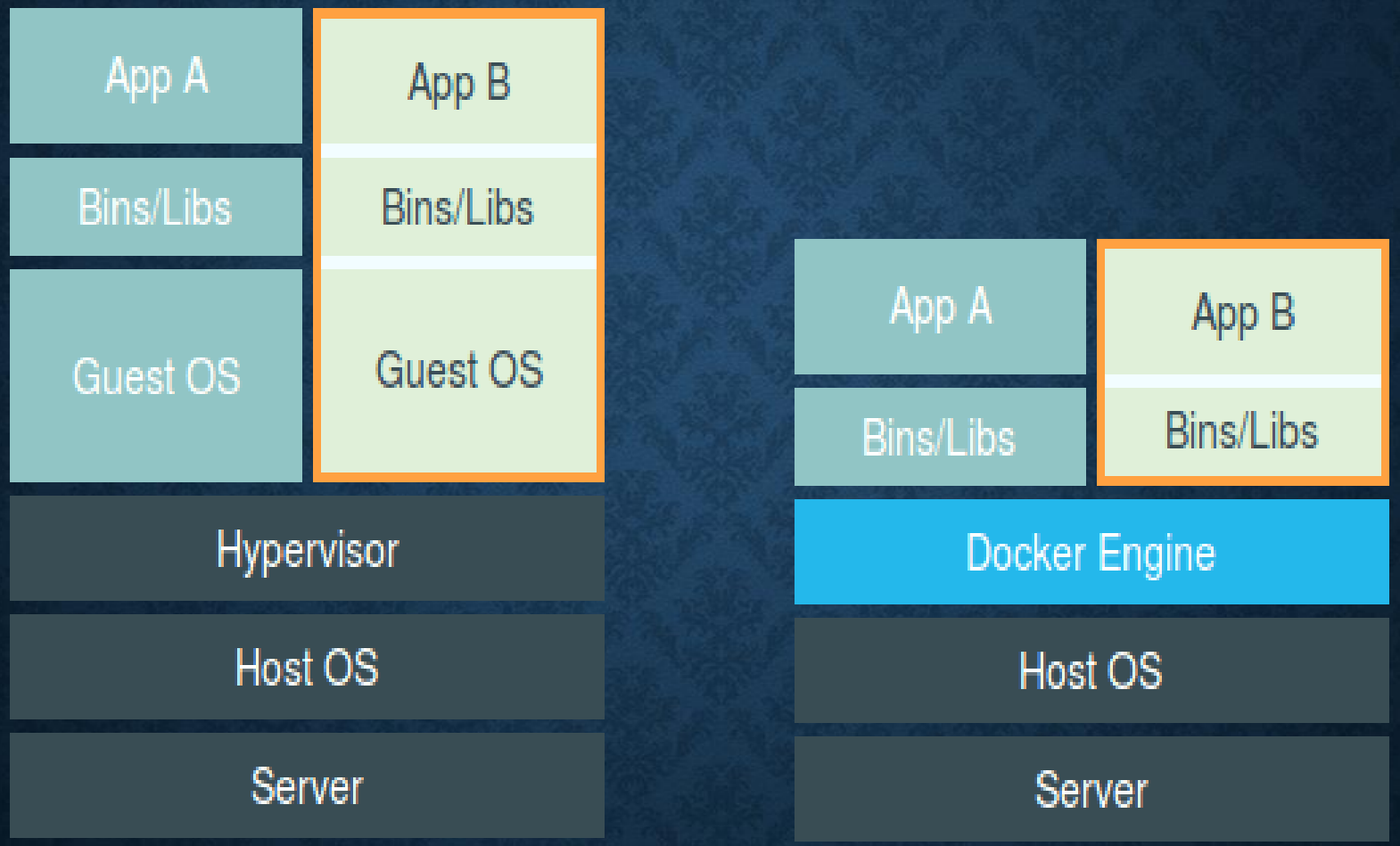


Docker

- Took from shipping containers.
- Docker is a tool designed to make containers in which we can deploy any type of applications easily.
- Docker uses union file system (layered)
- Docker performs os-level virtualization




Design
Development
Deployment
Testing/Release



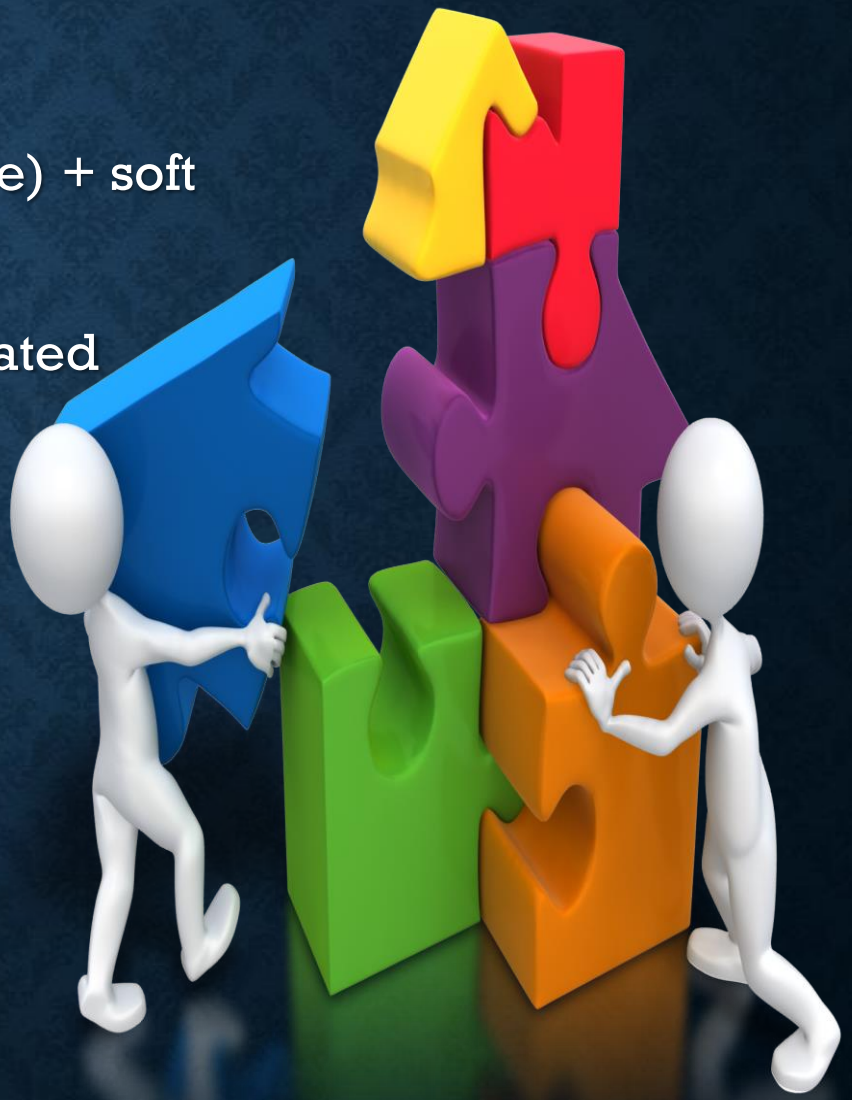
VMware **Docker**

Docker benefits

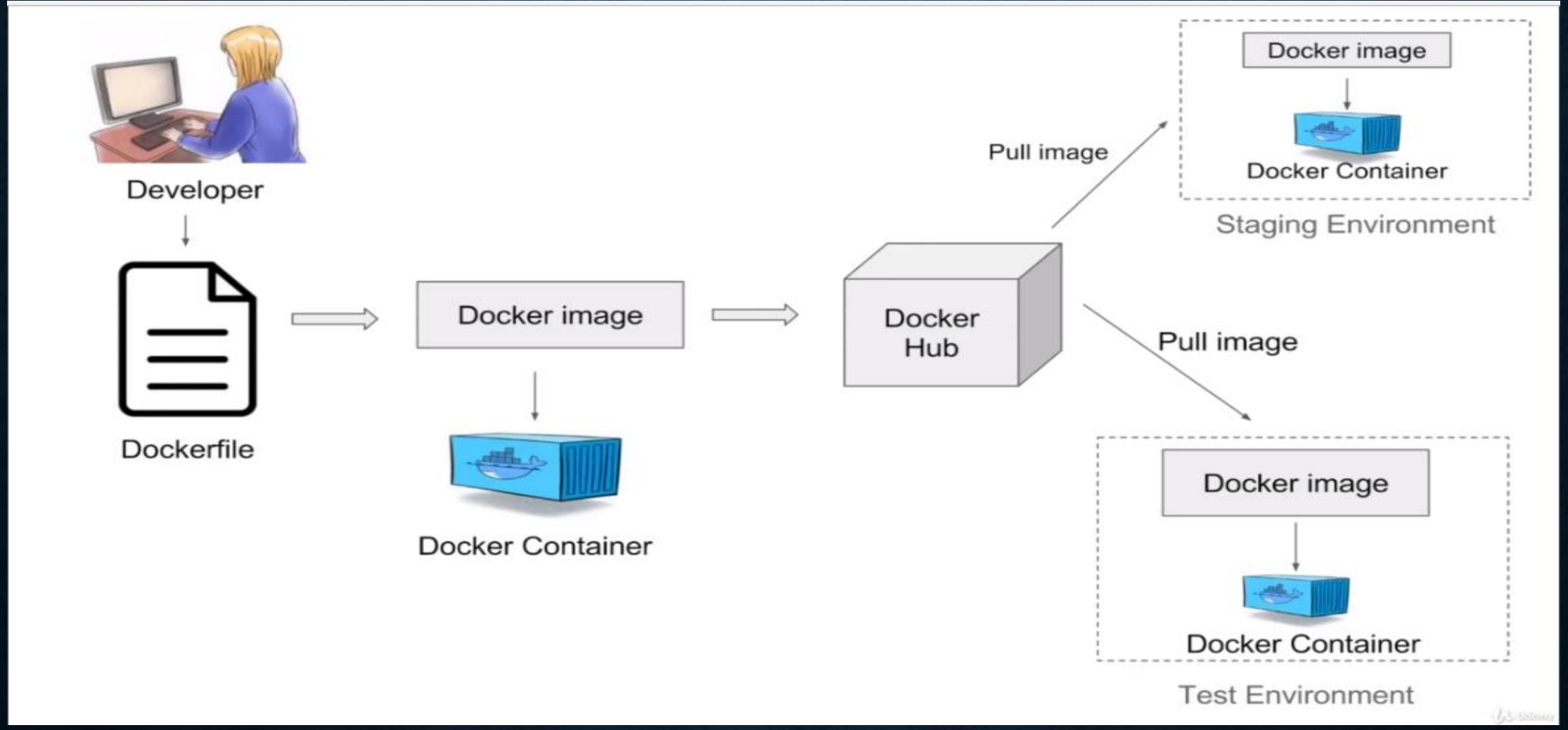
- Containerization (OS level virtualization)(No need guest OS)
- No pre-allocation of RAM
- Can replicate same environment
- Less cost
- Less weight (MB's in size)
- Fast to fire up
- Can run on physical/virtual/cloud
- Can re-use(same image)
- Can create machines in less time.

Docker components

- Docker image: Contains OS(very small)(almost negligible) + soft wares
- Docker Container: Container like a machine which is created from Docker image.
- Docker file: Describes steps to create a docker image.
- Docker hub/registry: Stores all docker images publicly.
- Docker daemon: Docker service



Docker work flow



Ways to create Docker Images

- Take image from Docker hub
- Create image from existing docker containers
- Create image from docker file

Docker Download & Install

- wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
- yum update -y
- yum install docker-io -y
- service docker start
- chkconfig docker on
- which docker

Docker Installation

- uname -r (OS)
- yum update -y
- yum install docker -y
- docker (to verify)
- docker --version (to verify)
- docker info (running or not)
- service docker start
- docker info (running or not)
- docker images (to see all images in our machine)
- docker ps (to see only running containers)
- docker ps -a (to see all containers)

How to create container?

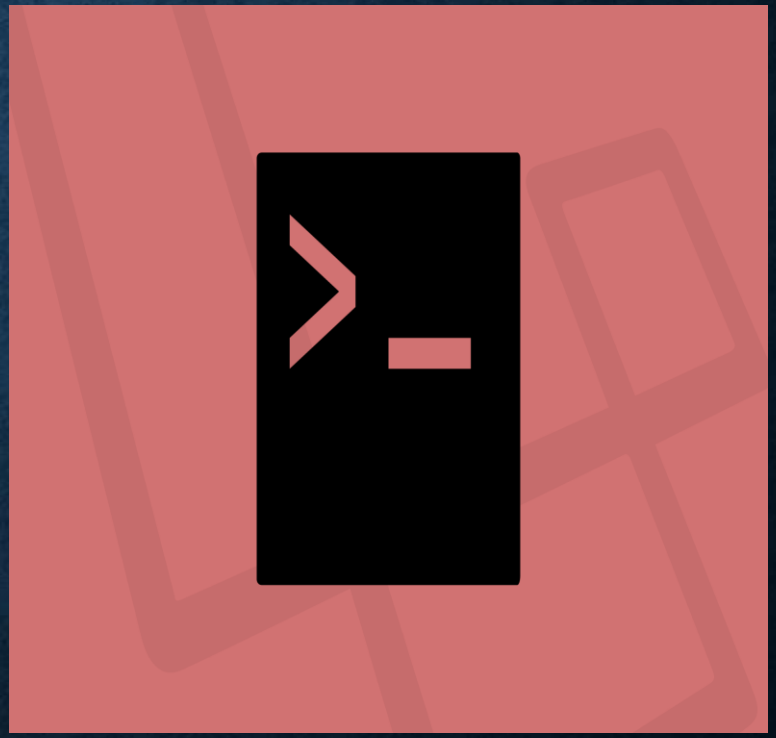
- To create container
  - docker run -i -t ubuntu /bin/bash
    - i -> Interactive mode
    - t -> Terminal
  - hostname
  - cat /etc/os-release
  - exit

Docker Basic Commands (Images)

- To see all images present in your local machine
  - docker images
- To search image in online docker registry
  - docker search ubuntu
- Download image from online docker registry to your machine
  - docker pull jenkins
  - docker pull chef/chefdk (create container from this image)
- To rename a container
  - docker rename container new_name

Docker Basic Commands

- Create container by giving container name
  - docker run --name saicontainer -it ubuntu /bin/bash
- To start container
  - docker start saicontainer
- To go inside container
  - docker attach saicontainer
- To see all containers
  - docker ps -a
- To see only running containers
  - docker ps
- To stop container
  - docker stop saicontainer
- To delete container
  - docker rm saicontainer

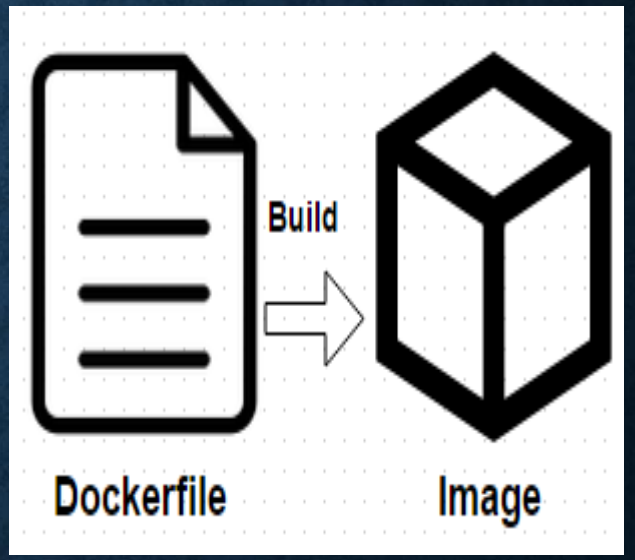


Create container from our own image

- Create container
  - docker run --name saicontainer -it ubuntu /bin/bash
  - touch /tmp/myfile
  - docker diff saicontainer (run outside of container)
- To create image from container (Be inside base machine)
  - docker commit saicontainer
- To give a name to your image
  - docker tag <imageID> saiimage
  - docker images
- To give image name while creating image
  - docker commit saicontainer <image name>
- Create container from our image
  - docker run --name haricontainer -it saiimage /bin/bash
  - ls /tmp

Dockerfile

- Dockerfile :
  - A text file with instructions to build image
  - Automation of Docker Image Creation
    - FROM
    - RUN
    - CMD
- Step 1 : Create a file named Dockerfile
- Step 2 : Add instructions in Dockerfile
- Step 3 : Build dockerfile to create image
- Step 4 : Run image to create container



Dockerfile

- vi Dockerfile
  - FROM ubuntu
  - RUN echo "Hi Sai" > /tmp/testfile
- To create image out of Dockerfile
  - docker build -t test . (-t : tag to image)
  - docker ps -a
  - docker images
- Create container from the above image
  - docker run -it --name testcontainer test /bin/bash
  - cat /tmp/testfile

Dockerfile

- vi Dockerfile

  - FROM ubuntu
  - WORKDIR /tmp
  - RUN echo "Hi Sai" > /tmp/testfile
  - ENV myname sai (echo $myname)
  - COPY testfile1 /tmp (testfile1 must be in current directory in EC2)
  - ADD test.tar.gz /tmp (test must be in current directory in EC2 in tar,gz format)

Volumes

- Volume is a directory inside your container
- First declare directory as a volume and then share volume
- Even if we stop container, still we can access volume
- Volume will be created in one container
- You can declare a directory as volume only while creating container
- You can't create volume from existing container
- You can share one volume across any no of containers
- Volume will not be included when you update an image
- Map volumes in two ways
  - Share host – container
  - Share container - container

Volumes (declaration by using docker file)

- Declare volume
  - FROM ubuntu
  - VOLUME ["/data"]

- docker build -t myimg .
- docker run -it --name mycont1 myimg /bin/bash
- ls (you can see data folder)
- touch /data/myfile
- exit

Volumes (container-container)

- To share volume while creating another container
  - docker run -it --name mycont2(new) --privileged=true --volumes-from mycont1(old) ubuntu /bin/bash
- ls /data/ (can see myfile)
- what ever u do in one volume, can see from other volume
  - touch /data/test
- docker start mycont1
- docker attach mycont1
- ls /data/ (can see test file)
- exit

Volumes(declaration by using command)

- docker run -it --name mycont3 -v /sai ubuntu /bin/bash
- ls (can see sai folder. can mount like above. same)

Volumes(host-container)

- Verify file in /home/ec2-user
- docker run -it --name nitcont -v /home/ec2-user:/raj --privileged=true ubuntu /bin/bash
- cd raj
- ls (can see all files created in your host)
- touch rajfile (in container /raj)
- exit
- ls (in host /home/ec2-user)
- Can see rajfile

- To see full details of container (even if it is in stopped state)
  - docker inspect nitcont

Port expose

- docker run -td --name webserver -p 80(host):80(container) ubuntu (d-daemon)
- docker port webserver
- docker exec -it webserver /bin/bash (to go inside container)
- apt-get update
- apt-get install apache2 -y
- cd /var/www/html
- echo "Hello" > index.html
- service apache2 restart
- In browser, PublicIP:80

(Can change host port. But not container port. Make sure, you are taking default port for container)

- Try with jenkins image (8080:8080) (Open 8080 port in SG of EC2 machine)
  (To unmap the port, stop the container or delete the container)

Important docker commands

- Stop all running containers: `docker stop $(docker ps -a -q)`
- Delete all stopped containers: `docker rm $(docker ps -a -q)`
- Delete all images: `docker rmi -f $(docker images -q)`

CMD

- vi Dockerfile
  - FROM ubuntu:16.04
  - CMD echo 'Welcome to docker'

  - docker build -t app .
  - docker run -it app

- can see Welcome to docker

- docker run -it app echo 'Welcome to AWS'
  - can see Welcome to AWS

- i.e run time command will overwrites the argument mentioned in CMD in docker file

CMD

- vi Dockerfile
  - FROM ubuntu:16.04
  - CMD echo 'Welcome to docker'
  - CMD echo 'Welcome to Python'
- docker build -t app1 .
- docker run -it app1

- can see Welcome to Python (i.e takes latest)

Enrtypoint

- vi Dockerfile

```
FROM ubuntu:16.04
ENTRYPOINT ["echo","Hi Docker"]
```

  - docker build -t app3 .
  - docker run -it app3

  (can see Hi Docker)

- ENTRYPOINT: We cann't overwrite ENTRYPOINT instructions by runtime command

  - docker run -it app3 echo 'Hi Hari'

- can see Hi Docker echo Hi Hari

- i.e run time command will overwrites the argument mentioned in CMD instruction in docker file
- i.e run time command will not overwrites the argument mentioned in ENTRYPOINT instruction in docker file

Entrypoint

- Combinely we can use

```
FROM ubuntu:16.04
ENTRYPOINT ["echo"]
CMD ["Hi Docker"]
```

  - docker build -t app4 .
  - docker run -it app4

- Can see Hi Docker

  - docker run -it app4 'Hi Hari'

  (Can see Hi Hari)

docker build -t mydemoimg -f saifile .

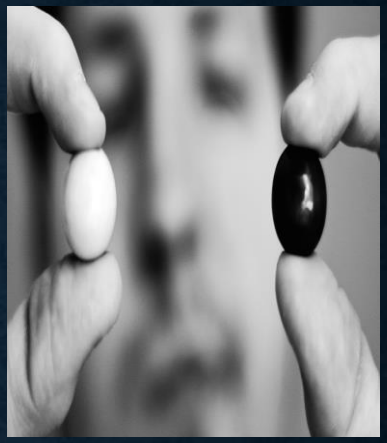
docker build -t mydemoimg -f saifile /home/ec2-user/

(delete webserver container first)

Delete container to unmap the port

docker build -t <ImageName> . -f <dockerfilename>

CMD, RUN & ENTRYPOINT



- The difference between the RUN and CMD instructions is that a RUN instruction actually runs the command and commits it, whereas the CMD instruction is not executed during build time

- RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.

- CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.

- CMD : It is an instruction that executes when we run a container out of image