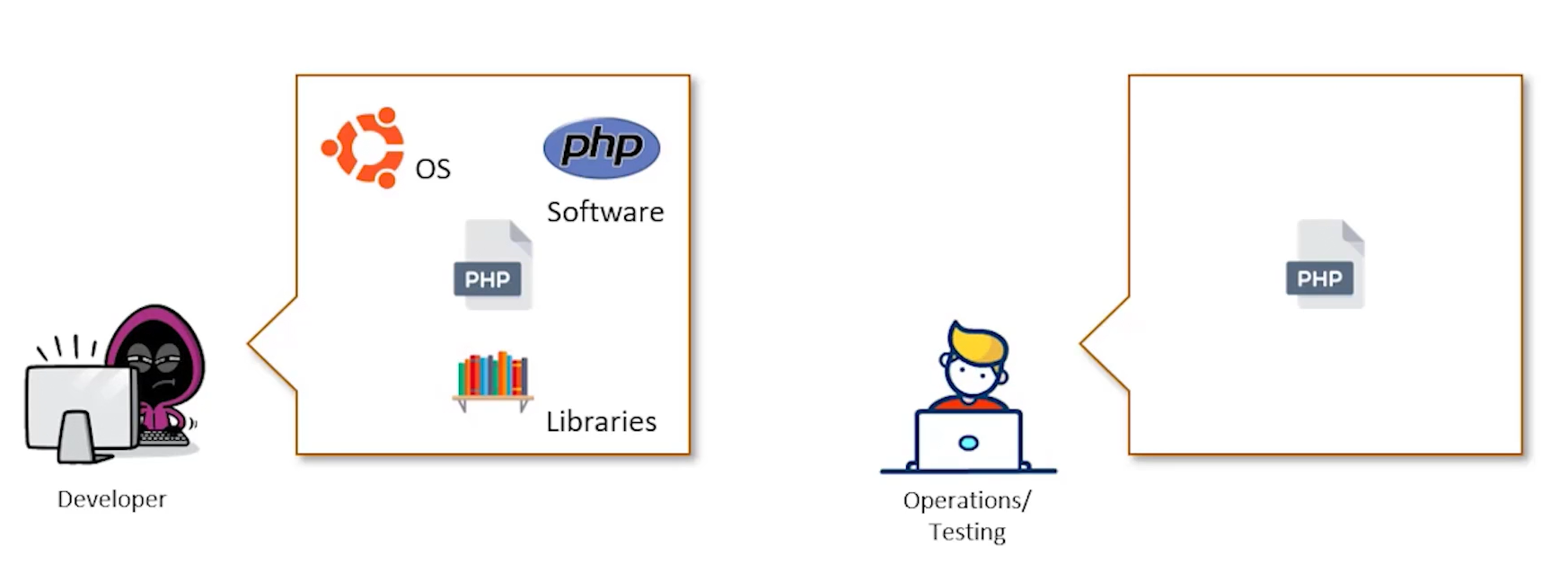
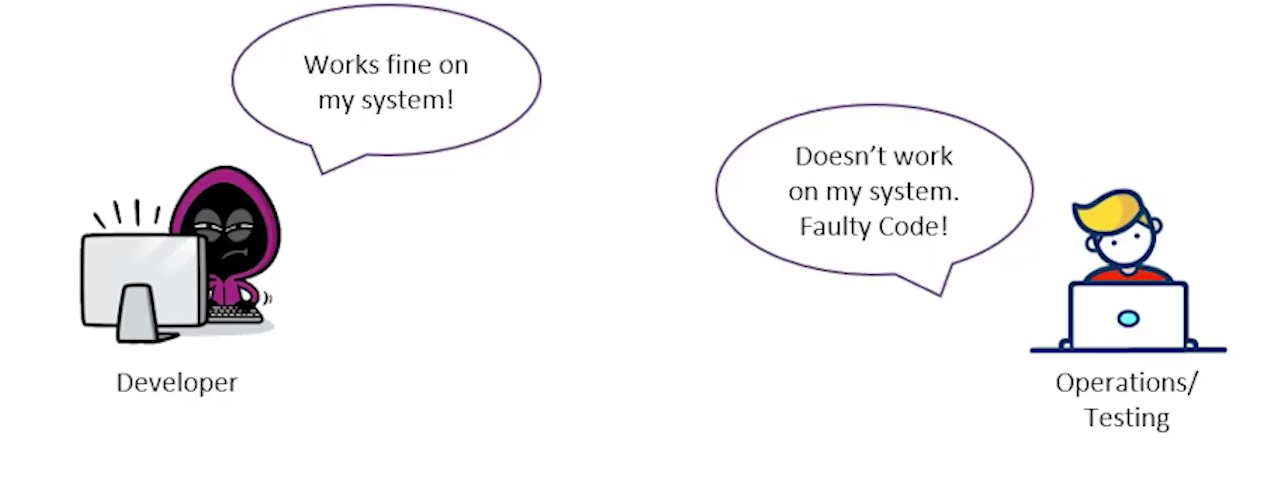
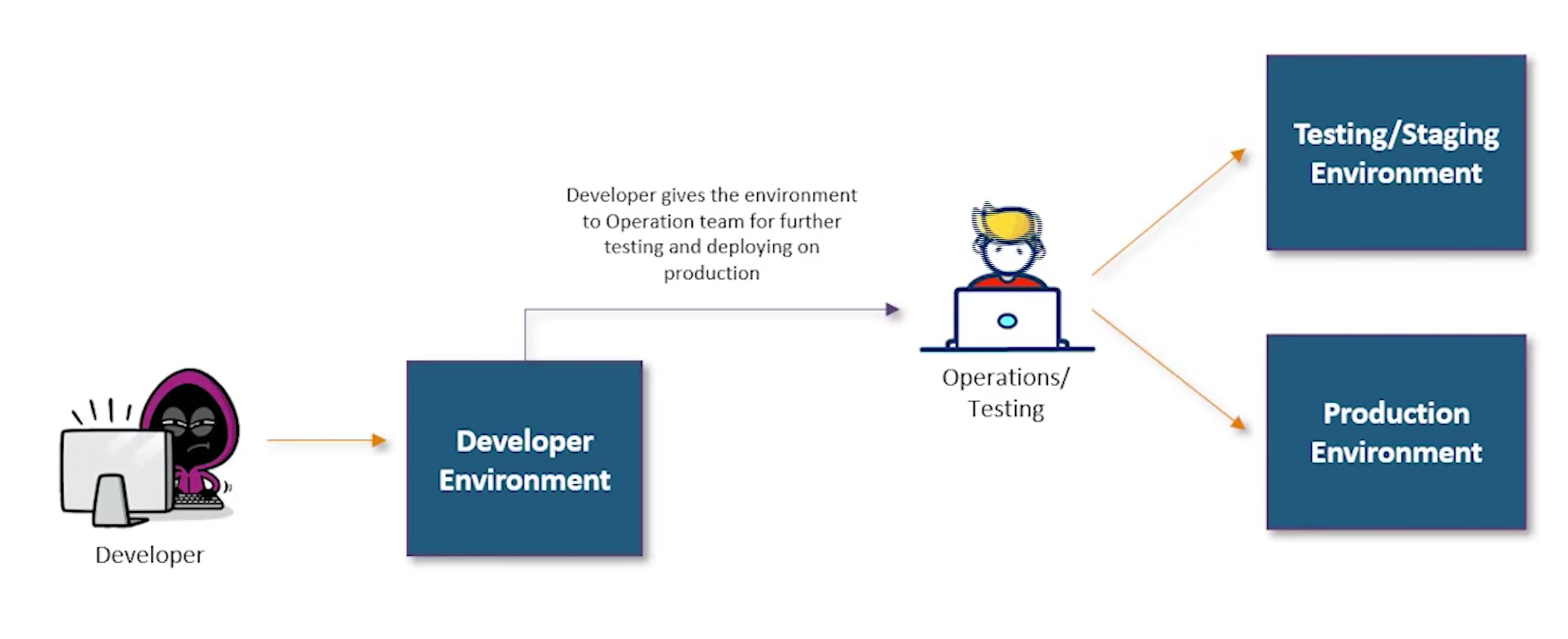
**What is Docker :**

Developers used to run the code on their system , it would run perfectly . But the same code did not run on the testing/operation team system. The issue may be because of the php version or any dependent software/binaries/libraries.

We need an entity which can contain all the software dependencies and can be ported on to other computers as plug and play





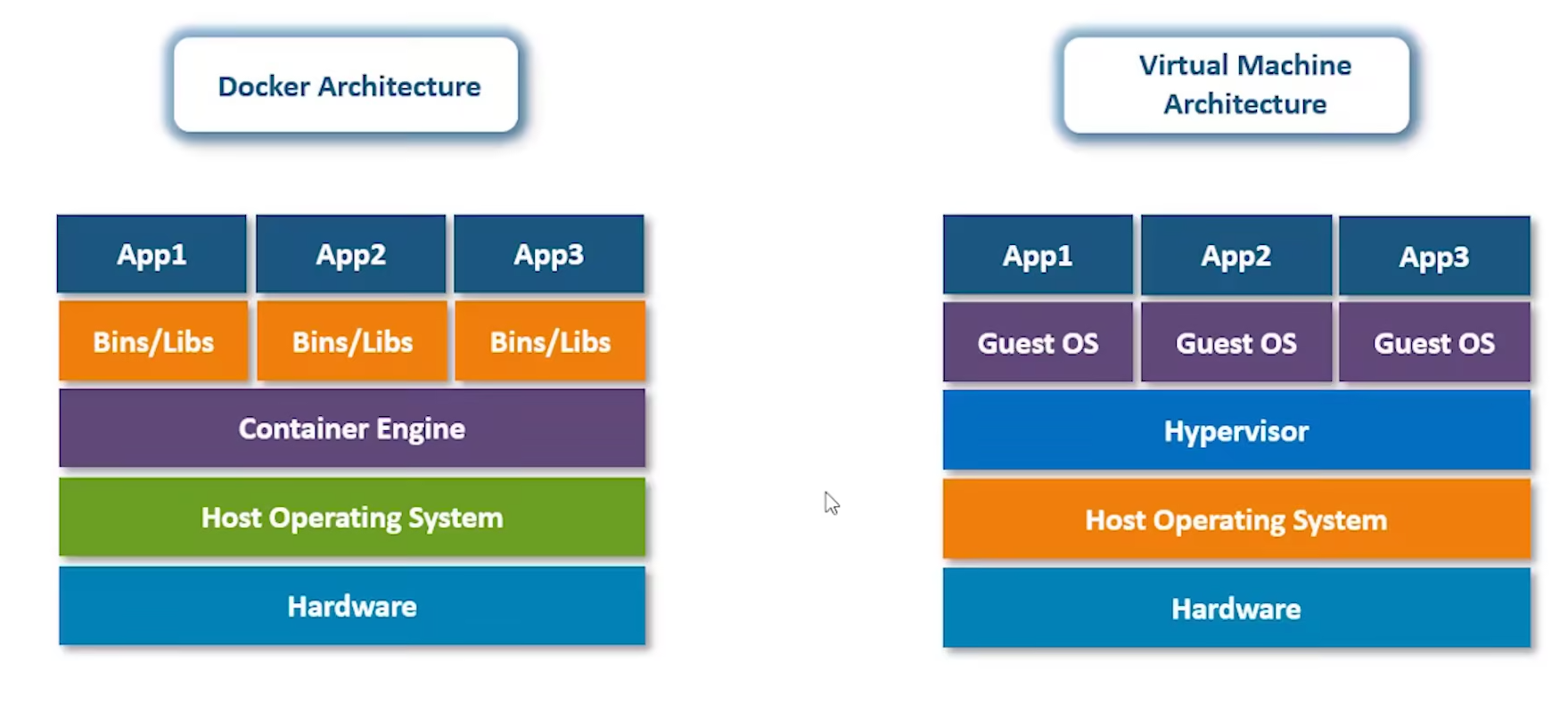


Docker is a computer program that performs operating system level virtualization, also known as containerization. It was first released in 2013 and developed by docker inc. Docker is used to run software packages called containers.

Application containerization is an OS level virtualization method used to deploy and run distributed application without launching entire virtual machine VM for each app.

Docker architecture :

Hardware 🡺 Operating System 🡺 Container Engine 🡺 Bins/libs 🡺 application



VM architecture :

Hardware 🡺 HOST OS 🡺 Hypervisor 🡺 Guest OS 🡺 Application

Installation :

Download docker tool box

On aws instance

sudo yum update

sudo yum install docker

docker –version

Advantages of Docker container:

* Take less space as compared to VM as it does not contain Guest OS.
* Runs reliably on any OS.
* Better resource utilization. It requires less hardware resulting reduction in cost.
* App isolation. Although all application deployed in one container all applications are isolated with each other.
* Container orchestration is solved.

Docker vs VM

* one example, let say you are creating an image for cent os which is Linux os.
* you have Linux machine and above that you are using docker to create centos image.
* so docker will create only binaries or libraries (bin/lib) to run internal or external commands.
* this way docker is not copying the whole OS.
* it is just using some of its resources.

Your app 🡺 Docker file (dockerize app) 🡺 Docker image 🡺 Repository 🡺 deploy into container

Docker container life cycle :

Docker hub 🡸 🡺 pull/push 🡸 🡺 docker engine 🡺 docker images 🡺 run/stop/delete

Docker hub : Docker hub the repo like git hub.

Docker engine : Docker engine is installed in your OS

Docker images : Docker image you download from docker hub to docker engine.

Container stages : If you run the image, it will be container.

Install Docker 🡺

On aws instance

sudo yum update

sudo yum install docker

#checking the version of docker 🡺

docker –version

**#**Pull the image from docker hub 🡺

**docker pull <image-name>**

**ex : docker pull ubuntu**

**#**list out all docker images downloaded into your system 🡺

**docker images**

**#**running containers from the image name🡺

**docker container run <image-name>**

**#**listing all the containers which are running in the system **=>**

**docker container ls**

**#**list out all container including stopped 🡺

**docker container ls -all**

**#** for logging into or accessing the container🡺

**docker exec <container-id>**

**#** for stopping a running container🡺

**docker container stop <container id>**

**#**kill the container🡺

**docker container kill <container id>**

* this command kills the container by stopping the execution immediately.
* the difference between docker kill and docker stop is that docker stop gives the container

time to shutdown gracefully, in situations when it is taking too much time for getting the container to stop, one can opt to kill.

**#**to remove a stopped container from system 🡺

**docker rm <container id>**

**#** to remove an image from the system🡺

**docker rmi <image id >**

**#** login to docker hub 🡺

**docker login**

**#** pushing you container on docker hub🡺

**docker push <docker hub- username>/<image-name>**

**#**you can verify the push on dockerhub🡺

**docker exec -it <container-id> bash**

**saving changes to a docker container. You can save the container as image and reuse image there on**

* run the container : **docker container run -it -d <image-name-example:ubuntu>**
* access the container : **docker exec -it <containerid> bash**
* do any operation
* exit from the container
* save that container as image so that next time you can run the image and use the container
* docker commit <containerid> <username>/<container name>

**Installing Apache in docker 🡺**

* Run a container
* docker container run -it -d <image-name>
* go to the container

docker exec -it <containerid> bash

* **install Apache**

apt-get update

apt-get install apache2

* map the port of your machine(the server on which docker is running) to the docker container
* exit from the container
* run the container as below

sudo docker run -it -p 82:80 -d <imagename>

* you can access the Apache page outside by browsing ipaddess-of-server:82

**#Pushing a container to dockerhub 🡺**

ubuntu container 🡺any changes on container 🡺commit the changes 🡺new\_image\_with\_cmmitted\_saved\_operations 🡺push 🡺docker HUB

sudo docker login

**Dockerfile**

* Docker can build images automatically by reading the instructions from a Dockerfile
* A Dockerfile is a text file of instructions which are used to automate installation and configuration of a Docker image . Dockerfiles make it easy to deploy multiple Docker containers without having to maintain the same image across multiple virtual machines.
* Using docker build users can create an automated build that executes several command line instructions in succession.
* Dockerfile 🡺 docker image 🡺 docker container 🡺 docker hub 🡺 staging server 🡺 production server

|  |  |
| --- | --- |
| FROM | First line of a docker file is always from  The FROM keyword is used to define the base image on which we will be building  FROM ubuntu |
| ADD | The ADD keyword is used to add files to the container being built.  ADD <source > <destination>  ADD . /var/www/html |
| RUN | the RUN keyword is used to add layers to the base image, by installing components. Each RUN statement , adds a new layer to the docker image.  EX : RUN apt-get -y install apache2 |
| CMD | The CMD keyword is used to run commands on the start of the container. These commands run only when there is no argument specified while running the container.  CMD apachectl -D FOREGROUND |
| ENTRYPOINT | ENTRYPOINT keyword is used strictly run commands when the container initializes.  ENTRYPOINT apachectl -D FOREGROUND |
| ENV | The ENV keyword is used to define environment variables in the container run time.  ENV name <variable-name> <value-of-variable> |
| WORKDIR | WORKDIR /path |
| EXPOSE | you can specify the port number |
| MAINTAINER |  |
| USER |  |
| VOLUME |  |

**Creating a docker file 🡺**

* mkdir docker 🡺 First create a folder docker
* sudo nano Dockerfile 🡺 create the docker file and add the content

FROM ubuntu

RUN apt-get update

RUN apt-get-y install apache2

ADD . /var/www/html

ENTRYPOINT apachectl -D FOREGROUND

ENV name Devops Intellipaat

* sudo nano index.html 🡺 create one more file index.html to test
* docker build <directory of docker file > -t <name of the container> 🡺 execute the docker file

docker build /home/nihar -t container-name

* docker run -it -p81:90 -d <container-name> 🡺 Run the built image
* echo $name 🡺 login to container and check variable name

**ADD /home/source /home/destination**

RUN apt-get update

RUN apt-get -y install apache2 >>> used to execute any command

CMD apachectl -D FOREGROUND > cmd is used to run a command without any argument specified and runs when container starts

ENTRYPOINT : same as cmd but entrypoint will run irrespective of argument

ENV :

ENV name name-of-valriable value-of-variable

**Ex :**

**FROM ubuntu**

**RUN apt-get update**

**RUN apt-get -y install apache2**

**ADD . /var/www/html**

**ENTRYPOINT apachectl -D FOREGROUND**

**ENV name Devops Intellipaat**

**Docker storage :**

by default all the data of a container is stored on a writable container layer. This layer has the following properties

* data only exists while container is active, if the container no longer exists, the data is also deleted along with the container.
* The writable container layer is tightly coupled with the host machine, hence not portable.
* The data on the writable layer in the container is written using a storage driver.

To persists data inside the container, even after it is deleted , we have two options

* Docker volumes
* Bind Mounts

**Docker volumes :**

A docker volumes is a mountable entity which can be used to store data in docker file system .

it creates a docker volume which can be attached or detached from docker

**docker volume create <volumename>**

**docker run -it --mount source=<source folder>,destination=<destination>**

**Bind mount**

Bind mounts mount a directory of the host machine to the docker container

**docker run -it -v<sourcedirectory >:<destination-directory> -d <container-name>**

**sudo docker run -it -v /home/ubuntu/mount:/demo -d ubuntu**

**Linking to Docker container**

* linking is a legacy feature of docker, which is used to connect multiple containers
* with linking containers can communicate among each other

**$ docker run –it --name <name-of-container> --link <container-name> -d ubuntu**

**docker run -it --link <name of conatiner > -d <imagename>**

**ex :**

**docker run -it --name cont2 --link cont1 -d ubuntu**

here while running cont2 from image ubuntu it is being linked to container 1

**Microservices**

**Monolithic applications and its disadvantages:**

* A Monolithic application is a single tiered software application in which different components are combined into a single program which resides in a single platform.
* basically monolithic application means, all present in single source code.
* application is large and complex to understand
* entire application must redeployed on an application update
* bug in any module, can bring down entire applications
* has a barrier to adopting new technologies

**Microservices :**

* Microservices are a software development architectural style that structures an application as a collection of loosely coupled services .
* application is distributed, hence is easy to understand
* the code of only the microservice which is supposed to be updated is changed
* bug in one service does not affect other services
* no barrier to any specific technologies

**Docker Compose**

* to resolve the problem of monolithic application, we came up with microservices. And microservices can be deployed without any manual intervention by using docker compose.
* Compose is a tool for defining and running multi-container docker application.
* With compose you use a YAML file to configure your applications services. Then with a single command you create and start all the services from your configuration.

**install :**

sudo curl -L https://github.com/docker/compose/releases/download/1.29.2/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

**YAML file :**

YAML is a superset of a json file. There are

maps : maps are key value pair

<key>:<value>

when we map a key to a value in YAML files they are termed MAPS.

Name:Intellipaat

course:Devops

**Lists : YAML Lists are a sequence of objects**

args

-arg1

-arg2

-arg3

ex

args

- sleep

- "1000"

- message

**runnig a sample docker file 🡺**

* create a folder called docker
* sudo mkdir docker
* cd docker
* create a yml file named docker-compose.yml

version: '2'

services:

sample1:

image:'httpd'

ports:

- "80:80"

sample2:

image: 'nginx'

* **build the docker compose file**

docker-compose up -d

docker ps

**to down all containers created by docker compose file 🡺**

docker-compose down

**Example 2: setting up wordpress project**

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: somewordpress

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

depends\_on:

- db

image: wordpress:latest

ports:

- "8000:80"

restart: always

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

db\_data: {}

**Container Orchestration**

* Applications are typically made up of individually containerized components, often called microservices.
* That must be organized at the networking level for the application to run as intended.
* The process of organizing multiple containers in this manner is known as container orchestration.

**Docker swarm :**

* Docker swarm is clustering and scheduling tool for docker containers. With swarm, IT administrators and developers can establish and manage a cluster of docker nodes as single virtual system.

**Diagram

Description automatically generated**

**Creating a Docker Swarm Cluster :**

* docker swarm comes pre installed with docker .

docker swarm init --advertise-addr=<ip-address-of-leader>

**🡺it will convert the node into master and generate one connection link which we need to execute on slave node**

docker node ls  **🡺 this will show the master slave nodes**

sudo docker leave --force  **🡺 to leave the swarm, if execute in the node, it will be disconnected**

**Services in docker swarm :**

* containers on the cluster are deployed using services on docker swarm.
* A service is a long running docker container that can be deployed to any node worker

**Diagram

Description automatically generated with medium confidence**

docker service create --name <name-of-service> --replicas <number-of-replicas> <image-name>

docker service create -name <name-of-service> --replicas <no-of-replicas> --publish <port-mapping> <imagename>

docker service ls  **🡺 list out all services**

docker service rm <service-name> 🡺**to delete any service**

**example :**

* **create a service for nginx webserver**
* **there should be 3 replicas of this service running on the swarm cluster**
* **try accessing the service from master ip and slave ip**

**docker service create --name nginx --replicas 3 -p 80:80 nginx**

**docker service ls**

**Docker Network**

* **in order to have interactions between docker containers we need docker networks.**
* **One of the reasons Docker containers and services are so powerful is that you can**

**connect them together or connect them to non-Docker workloads. And this can be accomplished using Docker Networks**

**website container 🡸🡺 database container**

**Docker network type 🡺**

* **bridge :**
* **host :**
* **overlay:**
* **macvlan:**
* **none:**

**<html>**

**<head>**

**<title>Docker Sample App</title>**

**<?php**

**if($\_SERVER['REQUEST\_METHOD'] == "POST")**

**{**

**$servername = "db";**

**$username = "root";**

**$password = "intelli";**

**$dbname = "docker";**

**$name=$\_POST["name"];**

**$phone=$\_POST["phone"];**

**// Create connection**

**$conn = new mysqli($servername, $username, $password, $dbname);**

**// Check connection**

**if ($conn->connect\_error) {**

**die("Connection failed: " . $conn->connect\_error);**

**}**

**$sql = "INSERT INTO emp (name, phone)**

**VALUES ('".$name."', '".$phone."')";**

**if ($conn->query($sql) === TRUE) {**

**echo "New record created successfully";**

**} else {**

**echo "Error: " . $sql . "<br>" . $conn->error;**

**}**

**$conn->close();**

**}**

**?>**

**</head>**

**<body>**

**<form action="index.php" method="POST">**

**<input type="text" name="name">**

**<input type="text" name="phone">**

**<input type="submit" name="submit">**

**</form>**

**</body>**

**</html>**

**How you can save changes to a container :**

|  |  |
| --- | --- |
| Pull the docker container | docker pull ubuntu |
| run the container | docker run -it -d ubuntu |
| Access the container using the command | docker exec -it <container-id> bash |
| install apache 2 on this container | sudo apt-get update  sudo apt-get install apache2 |
| Exit the container and save the container. The saved container will be converted to an image | docker commit container\_id username/imagename |
|  |  |

**Pushing a container to Docker Hub :**

|  |  |
| --- | --- |
| sudo docker login |  |
| sudo docker push user-name/image-name |  |

**Docker File :**

Docker file is a text document that contains all the commands a user could call on the command line to assemble an image.

Using docker build users can create an automated build that executes several command line instructions in succession.

Dockerfile 🡺 docker image 🡺 docker container 🡺 docker hub 🡺 staging server 🡺 production server

|  |  |
| --- | --- |
| FROM | First line of a docker file is always from  The FROM keyword is used to define the base image on which we will be building  FROM ubuntu |
| ADD | The ADD keyword is used to add files to the container being built.  ADD <source > <destination>  ADD . /var/www/html |
| RUN | the RUN keyword is used to add layers to the base image, by installing components. Each RUN statement , adds a new layer to the docker image.  EX : RUN apt-get -y install apache2 |
| CMD | The CMD keyword is used to run commands on the start of the container. These commands run only when there is no argument specified while running the container.  CMD apachectl -D FOREGROUND |
| ENTRYPOINT | ENTRYPOINT keyword is used strictly run commands when the container initializes.  ENTRYPOINT apachectl -D FOREGROUND |
| ENV | The ENV keyword is used to define environment variables in the container run time.  ENV name <variable-name> <value-of-variable> |
| WORKDIR | WORKDIR /path |
| EXPOSE | you can specify the port number |
| MAINTAINER |  |
| USER |  |
| VOLUME |  |

EX:

FROM ubuntu

RUN apt-get update

RUN apt-get -y install apache2

ADD . /var/www/html

ENTRYPOINT apachectl -D FOREGROUND

Difference between CMD and ENTRYPOINT :

The difference between CMD and ENTRYPOINT is , NETRYPOINT will run irrespective of the argument is specified or not.

**Creating a docker file :**

|  |  |
| --- | --- |
| First create a folder docker | mkdir docker |
| create the docker file and add the content | sudo nano Dockerfile  FROM ubuntu  RUN apt-get update  RUN apt-get-y install apache2  ADD . /var/www/html  ENTRYPOINT apachectl -D FOREGROUND  ENV name Devops Intellipaat |
| create one more file index.html to test | sudo nano index.html |
| execute the docker file | docker build <directory of docker file > -t <name of the container>  docker build /home/nihar -t container-name |
| Run the built image | docker run -it -p81:90 -d <container-name> |
| login to container and check variable name | echo $name |
|  |  |

|  |  |
| --- | --- |
| Common Docker Comands | |
| docker build |  |
| docker –version | Display the docker version |
| docker pull <image\_name> | Pull images from central docker repository  Ex : docker pull ubuntu |
| docker images | List out all installed docker images in the system |
| docker run <image\_name> | it helps running containers from the image name  Ex: docker run ubuntu |
| docker run -it -d <image\_name> | -it option is for interactive |
| docker ps | List out containers which are running in the system |
| docker ps -a | list out container which are running and stopped in the system |
| docker exec <container id> | logging into the container . container id can be get from ps command  ex: docker exec 1236h438 |
| docker stop <container id> | for stopping a running container. It gives time to container to stop gracefully. |
| docker kill <container\_id> | This command kill the container by stopping its execution immediately. |
| docker rm <container id> | remove the stopped container |
| docker rmi | remove the image from the system |
| **docker hub** | |
| docker login | login to docker hub account |
| docker push <username>/<containerid> | pushing container on docker hub  Ex: docker push nihar/123gh56k |
| docker exec -it <containerid>bash | verify the push on docker hub |
| docker pull <username>/<container> | pull the container from hub |
| Docker File | |
|  |  |

Docker stop gives time to container shutdown gracefully. In situation where docker stop is taking time to stop, once can opt to kill it.

Docker Image 🡺

used to run code in a docker container

docker images are like templates

an image is similar to a snapshot.

an image is a collection of files/layer that contain all of the components required to setup a fully functional container.

Why should we use docker image 🡺

* deployed in a docker environment , it can be used to run asa a docker container,
* docker images are reusable assets.
* does not have to recrate an image from scratch.

Container 🡺

* unit of software that packages up code and all its dependencies.
* container is a runnable docker image instance.

docker images vs container 🡺

* snapshots of containers that are used to create new containers.
* run the image, it turns into a container
* images can be created only once
* containers can be created unlimited number of times
* we can share docker images but not docker containers.

**Docker important questions**

**what is the difference between docker stop and docker kill ?**

Ans : Docker stop command gives time to stop the container however docker kill does not give any time. It immediately stop the container.