DOCKER

MONOLITHIC:

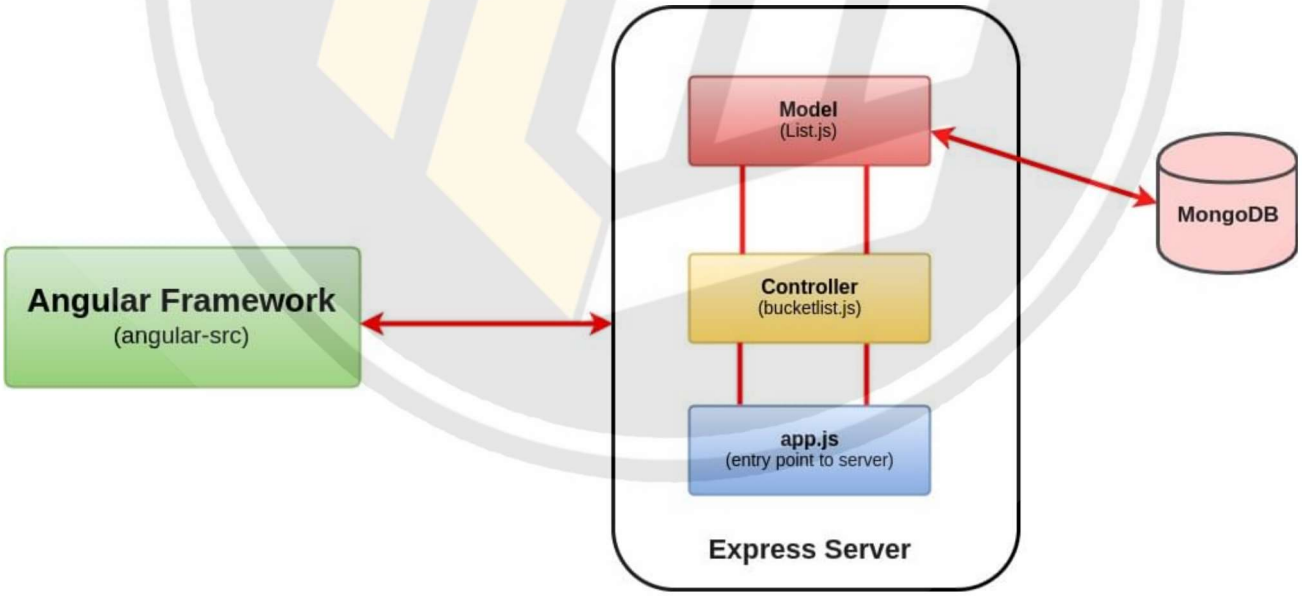
If an application contains N number of services (Let's take Paytm has Money Transactions, Movie Tickets, Train tickets, etc..) If all these services are included in one server then it will be called Monolithic Architecture. Every monolithic Architecture has only one database for all the services.

MICRO SERVICES:

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:

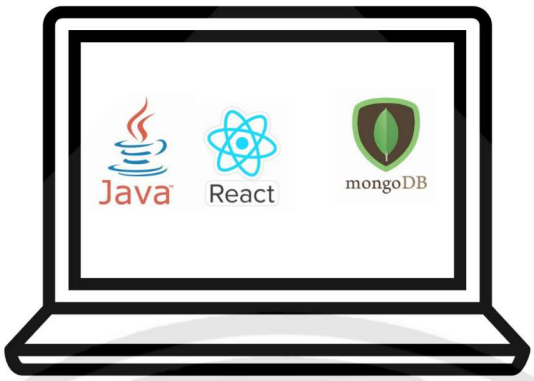
let us assume that we are developing an application, and every application has front end, backend and Database.



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.

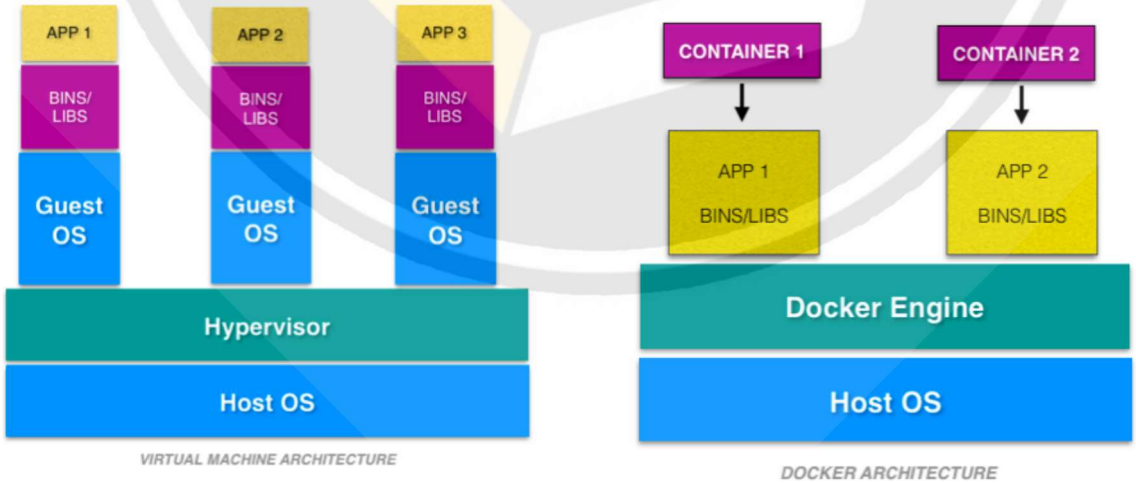


So its really a hectic situation to maintain multiple versions of same tool in our system. To overcome this problem we will use virtualization.

VIRTUALISATION:

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 virtualisation.



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.

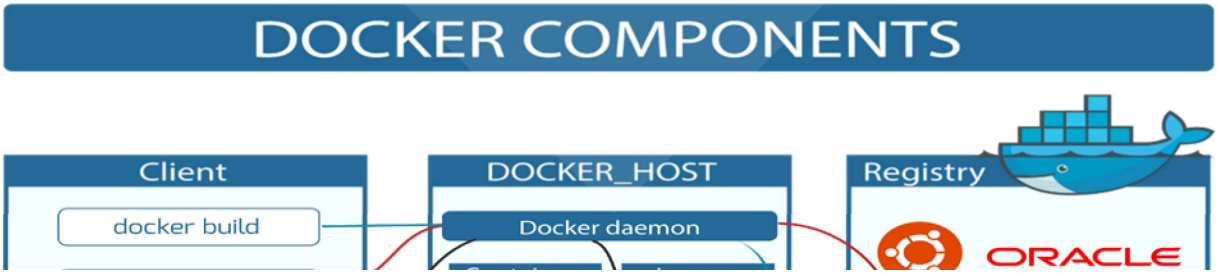
CONTAINER:

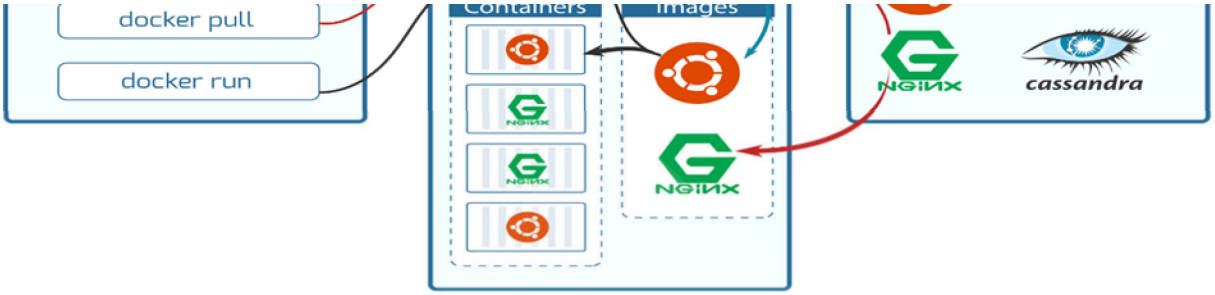
- Container is nothing but, it is a virtual machine which does not have any OS.
- Docker is used to create these containers.
- A container is like a lightweight, standalone package that contains everything needed to run a piece of software.
- It includes the application code, runtime, system libraries, and dependencies.
- To create a container we use docker.

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.

DOCKER 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 REGISTYR:

A Docker registry is a place where Docker images are stored and can be easily shared. It serves as a centralized repository for Docker images, allowing users to upload, download, and manage container images.

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.

DOCKER BASIC 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: is used to rename the container.

To rename docker container: docker rename old_container new_container

HOW TO CHANGE DOCKER PORT NUMBER:

- 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: is a command in Docker that allows you to save a Docker container as a tarball archive file. This tarball contains the container's file system and can be used to transfer the container to another system or share it with others.

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

ALTERNATE CONTAINER 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

DOCKER EXEC: is a command that allows you to run commands inside a running Docker container. You can use docker exec to execute commands, check the container's file system, troubleshoot issues, or perform various tasks within the container without the need to start a new instance of the container.

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:

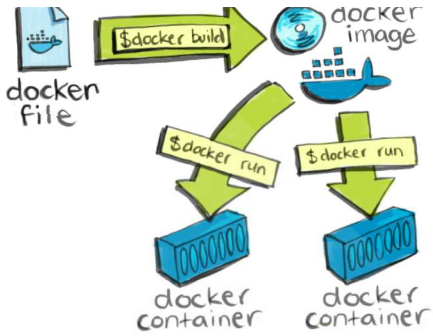
- First it should have a base image - docker run nginx
- Now create a container from that image - docker run -it --name container_name image_name /bin/bash
- Now start and attach the container
  - go to tmp folder and create some files (if you want to see the what changes has made in that image - docker 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, image description etc.
- RUN: To execute commands while we build the image.
- COPY: Copy files/folders from local system to container where need to provide Source and Destination.
- ADD: it is also used to copy the files and also It can download files from the internet and and send the files to container.
- 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: Used to pass Environment variables.

COMPARISIONS:

ENV vs ARG

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

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.
- 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 vs ADD:

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

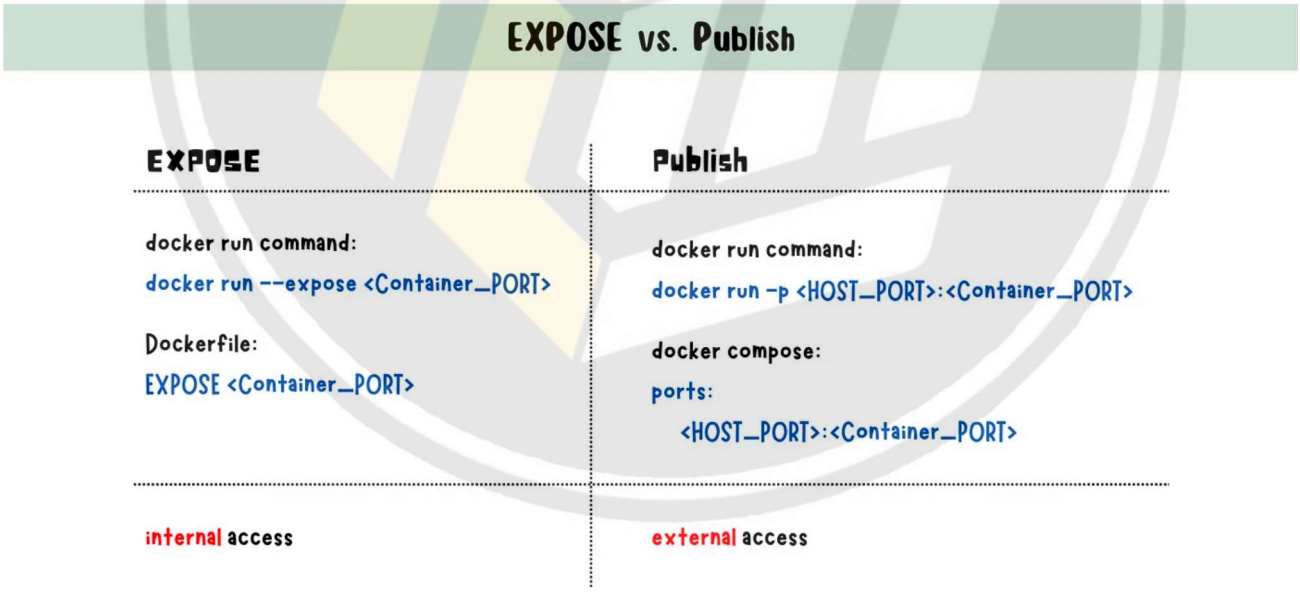
STOP vs KILL:

- 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.

PULL vs RUN:

- PULL: It is used to download the images from docker registry
- RUN: It is used to create a container.

EXPOSE VS PUBLISH:



DOCKER FILE TO CREATE AN IMAGE:

FROM: ubuntu

RUN: touch aws devops linux

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

FROM: ubuntu

RUN: touch aws devops linux

RUN echo "hello world">/tmp/file1

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 FILE TO INSTALL NGINX ON UBUNTU:

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

DOCKER FILE TO INSTALL APACHE2 ON UBUNTU:

```
FROM ubuntu
RUN apt update -y
RUN apt install apache2 -y
COPY index.html /var/www/html/
CMD ["/usr/sbin/apachectl", "-D", "FOREGROUND"]
```

DOCKER FILE TO INSTALL HTTPD ON CENTOS

```
FROM centos:centos7
MAINTAINER name mustafa
RUN yum install httpd -y
COPY index.html /var/www/html/
EXPOSE 80
```

```
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
~
~
```

DOCKER FILE TO USE NGINX IMAGE

```
EC2
FROM nginx
COPY index.html /usr/share/nginx/html/
~
```

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 A VOLUME 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

CREATING VOLUMES FROM COMMAND:

-

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 IN VOLUMES:

- 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 VOLUME:

- 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 REGISTYR:

Docker registry is a place where Docker images are stored and can be easily accessed. There are two types of docker registries

1. Cloud based registry : DockerHub, ACR (Amazon Container Registry), GCR (Google Container Registry), ACR (Azure container Registry).
2. Local registry : Jfrog, Nexus, DTR (Docker Trusted Registry).

By default docker hub is the default registry.

Docker Hub is a cloud-based platform that allows developers to store and share their Docker container images. A Docker container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.

DOCKER PUSH:

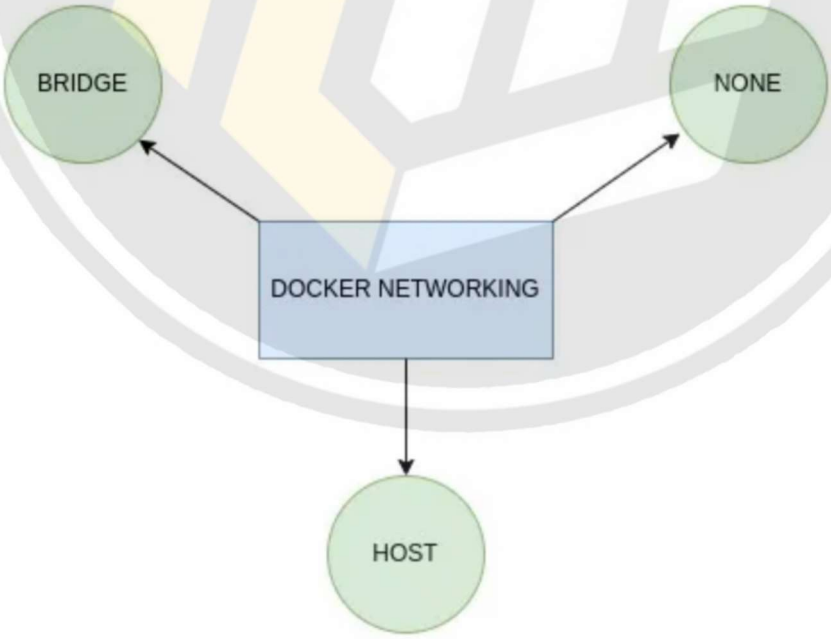
It is a command used in Docker, a containerization platform, to upload or "push" your Docker image to a container registry.

STEPS TO PUSH DOCKER IMAGE:

- Clone any repo with Dockerfile
- Build the docker file : docker build -t image1 .
- now create a docker hub account
- Now log into the docker hub 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 shaikmustafa/my-repo:image1 (ex: project1)
- docker push shaikmustafa/my-repo:image1
- 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 shaikmustafa/my-repo:image1
- docker run -it - -name mycontainer shaikmustafa/my-repo:image1 /bin/bash
- Now go to docker hub and select your image -- > settings -- > make it private.
- Now run docker pull shaikmustafa/my-repo:image1
- If it is denied then login again and run it.
- If you want to delete image settings -- > project1 -- > delete

DOCKER NETWORKS:

Docker networking is primarily used to establish communication between Docker containers and the outside world. Docker supports different types of networks.



networks

When we install Docker, it created three networks automatically.

1. *Bridge*
2. *None*
3. *Host*
4. *OverLay*

BRIDGE NETWORK:

Bridge is the default network a container gets attached to. It is a private internal network created by Docker on the host. Eah container connecting to this network get their own internal private network address. Generally IPs get assigned in the range of 172.17.x.x to the containers. The containers can access each other using the internal IP. If we want to access any of these containers from the outside world, then we need to map the ports of these containers to ports on the Docker host. The another name of Bridge network is *docker0* as it's the default one.

NONE NETWORK:

With a none network the docker container is not attached to any other network. The container can not reach to the outside world and no one from the outside world can access the container. We can use it when we want to disable the networking on a container.

HOST NETWORK:

If we want to access the containers externally, then we can attach the container to Host network. There is no network isolation between the host and the container. That means like bridge network we don't need to any additional port mapping. For example, If we deploy a web app which is listening on port 80 , then it will be accessible on port 80 on the host. But we will now not be able to run multiple web containers on the same host and the same port. If we try to run another instance of our web app that listens on the same port, it won't work as they share the host networking and two processes can't listen on the same port at the same time.

OVERLAY NETWORK:

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

DOCKER NETWORK COMMANDS:

- 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 networkName contName
- apt install iputils-ping -y : command to install ping checks
- To disconnect from the container: docker network disconnect NetworkName contName
- To prune: docker network prune

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
```

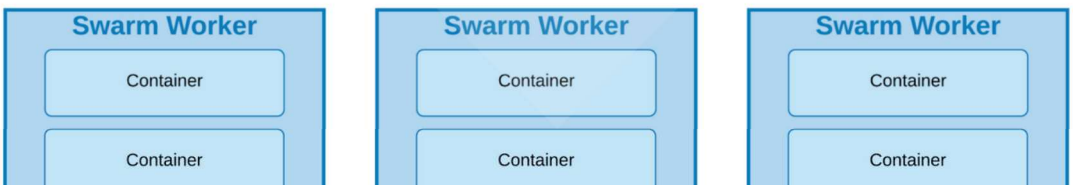
DOCKER FILE TO DEPLOY NODE JS FILE:

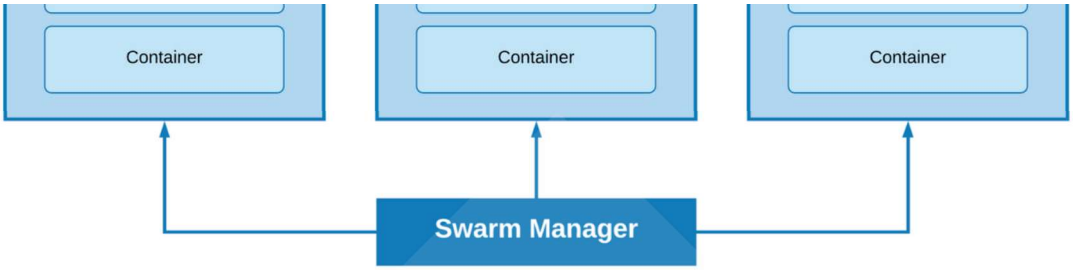
```
FROM node:16
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 8081
CMD ["node", "index.js"]
```

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

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.

KEY FEATURES OF DOCKER SWARM:

Node Clustering: Docker Swarm allows you to create a group of Docker hosts that work together as a cluster. Each host in the cluster is referred to as a "node."

Service Deployment: You can define services, which are the applications you want to run, and deploy them across the Docker Swarm cluster. Docker Swarm ensures that the services are distributed and run on the available nodes.

Load Balancing: Swarm includes built-in load balancing, distributing incoming requests among the various instances of a service running on different nodes. This helps in scaling applications and improving performance.

High Availability: Docker Swarm provides high availability by automatically rescheduling tasks (individual units of a service) to healthy nodes in case of node failures.

Scalability: You can easily scale your applications by adding or removing nodes from the Swarm cluster. This allows you to adapt to changes in demand for your services.

DOCKER SWARM SETUP AND COMMANDS:

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)
- 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

DOCKER SWARM SERVICES:

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

DOCKER SERVICE COMMANDS:

- 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 allows you to describe the services, networks, and volumes for your application in a single file, typically named docker-compose.yml.
- This file contains configurations for different services, specifying how they should be built, configured, and connected.
- 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).

KEY FEATURES OF DOCKER COMPOSE:

Services: These are the different components or containers of your application, such as a web server, a database, or any other service.

Networks: Docker Compose allows you to define custom networks for your services, enabling communication between them.

Volumes: You can specify volumes to persist data or share it between containers.

Build: Docker Compose can build custom Docker images for your services based on the configurations you provide.

DOCKER 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

CREATING DOCKER-COMPOSE.YML:

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 ------> docker compose file execution

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

EX-2:

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

EX-3:

DOCKER COMPOSE COMMANDS:

- 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

DOCKER STACK:

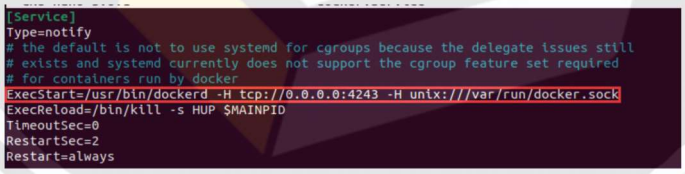
- Docker stack is used to create multiple services on multiple hosts. i.e it will create multiple containers on multiple servers with the help of compose file.
- To use the docker stack we have initialized docker swarm, if we are not using docker swarm, docker stack will not work.
- once we remove the stack automatically all the containers will gets deleted.
- We can share the containers from manager to worker according to the replicas
- 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

DOCKER STACK COMMANDS:

- 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:

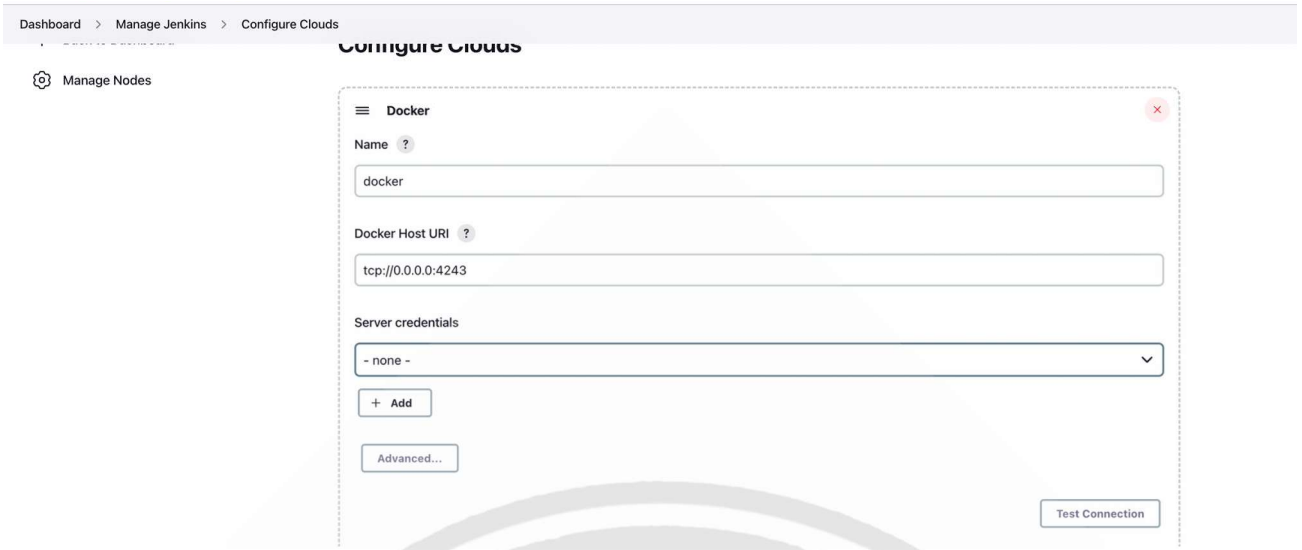
- Install docker and Jenkins in a server.
- vim /lib/systemd/system/docker.service



- 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.



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

DOCKER PORTAINER:

- it is a container organizer, designed to make tasks easier, whether they are 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 two elements, the Portainer Server and the Portainer Agent.
- Both elements run as lightweight Docker containers on a Docker engine
- Must have swarm mode and all ports enable with docker engine

DOCKER PORTAINER COMMANDS:

- 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

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