**docker**: containerization platform. We can utilize the memory in efficient way

**container**: pack the application along with its dependencies. Docker containers are the lightweight alternatives of the virtual machine. It allows developers to package up the application with all its libraries and dependencies and ship it as a single package.

**virtualization**: to efficint way of using physical system

**hypervier** : It can introduce the virtulization on physical system

**monolithic service**: N no.of services utilize single server and single data base.

**microservice**: N no.of services working on N no.of serveries each and every service has individual server along with database.

docker is a os level vituvalization

**Installation of docker:**

sudo dnf update -y

sudo dnf install docker -y

sudo systemctl start docker

sudo systemctl enable docker

sudo systemctl status docker

docker --version

docker version

sudo usermod -aG docker ec2-user

[ec2-user@ip-172-31-32-61 ~]$sudo usermod -aG docker ec2-user

[ec2-user@ip-172-31-32-61 ~]$ newgrp docker

[ec2-user@ip-172-31-32-61 ~]$docker pull nginx

[ec2-user@ip-172-31-32-61 ~]$ docker run -dt --name c2 -p 90:80 nginx

basic docker coomands:

======================

docker pull ubuntu: dowload the docker image

**crate container:**

#docker run -it --name cont1 ubuntu

this command can create the docker container at the same time go inside the docker container

ctrl p q--->come outside from the container with out exited mode

ctrl + d--->come outside from the container and contaner goes to exited mode

#docker run -itd --name cont2 ubuntu

this command is used create the container and the container is run in detached mode

#docker images --->this command will show the list of containers

#docker ps --->to see the list of running contaners

#docker ps -a ---->to see the list of running and stopped containers

#docker attach contanerid --->login to the container

#docker exec -it contanerid /bin/bash--->login to the container

#docker run -it --name cont -p 80:80 ubuntu --->create the containerand assign the port numner

#docker stop cont\_name

#docker start cont\_name

#docker kill cont\_name

#docker pause con\_name

#docker container ls

#docker inspect cont\_name-->this will show the detailed description of container.

#docker ps -a | grep Exit-----🡪to see the stopped containers

#docker system prune -a -----------🡪to remove all images

#docker container prune -----------🡪delete all unused containers

**dockerfile**:

A Dockerfile is a text file that contains instructions for building a container image

Docker file commands:

1.FROM:

Specifies the base image to use for the Docker container.

Ex:

FROM ubuntu:20.04

2. RUN:

Executes a command in the Docker image during the build process. Typically used to install packages or set up configurations.

RUN apt-get update && apt-get install -y python3

3. COPY:

Copies files from the host system to the container filesystem

COPY ./app /usr/src/app

4. ADD:

Similar to COPY, but can also handle remote URLs and extract archives

ADD <https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.96/bin/apache-tomcat-9.0.96.tar.gz> /opt

5. CMD

Specifies the default command to run when the container starts. Can be overridden by providing a different command at runtime.

6. ENTRYPOINT:

**Purpose:** Sets the default executable for the container. Unlike CMD, it is more rigid and difficult to override.

ENTRYPOINT ["python3", "app.py"]

7. ENV: EXPOSE Sets environment variables inside the container.

ENV APP\_HOME /usr/src/app

8. WORKDIR:

Sets the working directory for the container, similar to the cd command in a shell.

WORKDIR /usr/src/app

9. EXPOSE:

nforms Docker that the container will listen on the specified port at runtime. This does not publish the port, but it provides documentation.

EXPOSE 8080

10. VOLUME:

**Purpose:** Creates a mount point with the specified path and marks it as holding externally mounted volumes.

VOLUME /data

11. USER

Specifies the user to run the container as. By default, containers run as root.

USER www-data

12 ARG

Defines build-time variables that users can pass during the build process.

13. LABEL:

Adds metadata to an image in key-value pairs.

LABEL version="1.0" description="My app"

14. MAINTAINER

Specifies the author of the Dockerfile. This is now deprecated in favor of using LABEL

MAINTAINER Your Name [your.email@example.com](mailto:your.email@example.com)

Dockerfile for apache:

FROM ubuntu

RUN apt update -y

RUN apt install apache2 -y

COPY \* /var/www/html/

CMD ["/usr/sbin/apachectl","-D","FOREGROUND"]

git clone <https://github.com/mohanlucky2/bookmyshow.git>

mv bookmyshow/\* .

docker build -t apche .

docker run -itd –name cont-1 -p 80:80 apache

**Dockerfile for httpd:**

RUN yum update -y

RUN yum install httpd -y

COPY index.html /var/www/html/

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

**Dockerfile for jenkins.**

FROM amazonlinux:latest

RUN yum update -y && \

yum install -y java-17-amazon-corretto wget

WORKDIR /opt

RUN wget <https://updates.jenkins.io/download/war/2.481/jenkins.war>

EXPOSE 8080

CMD ["java", "-jar", "/opt/jenkins.war"]

commands:

=========

docker build -t jenkins-image .

docker run -d -p 8080:8080 --name jenkins-container jenkins-image

**Dockerfile for tomcat:**

FROM ubuntu:latest

RUN apt-get update -y && \

apt-get install -y openjdk-17-jdk wget

WORKDIR /usr/local

RUN wget <https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.96/bin/apache-tomcat-9.0.96.tar.gz>

RUN tar xzf apache-tomcat-9.0.96.tar.gz && \

rm apache-tomcat-9.0.96.tar.gz

COPY tomcat-users.xml apache-tomcat-9.0.96/conf/tomcat-users.xml

COPY context.xml apache-tomcat-9.0.96/webapps/manager/META-INF/context.xml

RUN mv apache-tomcat-9.0.96 tomcat

EXPOSE 8080

CMD ["/usr/local/tomcat/bin/catalina.sh", "run"]

docker build -t tomcat-ubuntu-image .

docker run -d -p 8080:8080 --name tomcat-ubuntu-container tomcat-ubuntu-image

---------------------------------------------------------------------------------------------------------------------

**Vim tomcat-users.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!--

Licensed to the Apache Software Foundation (ASF) under one or more

contributor license agreements. See the NOTICE file distributed with

this work for additional information regarding copyright ownership.

The ASF licenses this file to You under the Apache License, Version 2.0

(the "License"); you may not use this file except in compliance with

the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

-->

<tomcat-users xmlns="http://tomcat.apache.org/xml"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"

version="1.0">

<!--

By default, no user is included in the "manager-gui" role required

to operate the "/manager/html" web application. If you wish to use this app,

you must define such a user - the username and password are arbitrary.

Built-in Tomcat manager roles:

- manager-gui - allows access to the HTML GUI and the status pages

- manager-script - allows access to the HTTP API and the status pages

- manager-jmx - allows access to the JMX proxy and the status pages

- manager-status - allows access to the status pages only

The users below are wrapped in a comment and are therefore ignored. If you

wish to configure one or more of these users for use with the manager web

application, do not forget to remove the <!.. ..> that surrounds them. You

will also need to set the passwords to something appropriate.

-->

<!--

<user username="admin" password="<must-be-changed>" roles="manager-gui"/>

<user username="robot" password="<must-be-changed>" roles="manager-script"/>

-->

<!--

The sample user and role entries below are intended for use with the

examples web application. They are wrapped in a comment and thus are ignored

when reading this file. If you wish to configure these users for use with the

examples web application, do not forget to remove the <!.. ..> that surrounds

them. You will also need to set the passwords to something appropriate.

-->

<!--

<role rolename="tomcat"/>

<role rolename="role1"/>

<user username="tomcat" password="<must-be-changed>" roles="tomcat"/>

<user username="both" password="<must-be-changed>" roles="tomcat,role1"/>

<user username="role1" password="<must-be-changed>" roles="role1"/>

-->

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<user username="tomcat" password="tomcat" roles="manager-gui,manager-script"/>

</tomcat-users>

**Context.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<!--

Licensed to the Apache Software Foundation (ASF) under one or more

contributor license agreements. See the NOTICE file distributed with

this work for additional information regarding copyright ownership.

The ASF licenses this file to You under the Apache License, Version 2.0

(the "License"); you may not use this file except in compliance with

the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

-->

<Context antiResourceLocking="false" privileged="true" >

<CookieProcessor className="org.apache.tomcat.util.http.Rfc6265CookieProcessor"

sameSiteCookies="strict" />

<Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Number|String)|org\.apache\.catalina\.filters\.CsrfPreventionFilter\$LruCache(?:\$1)?|java\.util\.(?:Linked)?HashMap"/>

</Context>

**Network Commands**

* docker network ls
* docker network create my\_custom\_network
* docker network rm my\_network
* docker network inspect bridge
* docker network inspect my\_network
* docker network create --driver bridge my\_bridge\_network
* docker network rm my\_network
* docker network connect my\_network my\_container
* docker network disconnect my\_network my\_container
* docker network prune
* docker run --network my\_network nginx

**Network Types in Docker:**

Bridge Network

Host Network:

Overlay Network

None

1. Bridge Network (Default for standalone containers):

 Used for communication between containers on the same host.

 Containers on different bridge networks cannot communicate unless you use custom configurations.

docker network create --driver bridge my\_bridge\_network

1. Host Network:

Bypasses Docker’s networking stack and attaches the container directly to the host network.

docker run --network host nginx

1. Overlay Network:

Used for communication between containers across multiple Docker hosts (used in Swarm mode).

docker network create --driver overlay my\_overlay\_network

1. None:

 Containers are not connected to any network.

docker run --network none nginx

**DOCKER VOLUMES:**

* docker volumes are used store data inside the container
* docker volume is a simple directory inside the container
* containers uses host resources like cpu,ram and rom
* single volume can be shared to multiple containers

creation of the container with volume in different ways:

vim myimage

FROM ubuntu

VOLUME /myvol

docker build -t image1 .

docker run -it --name cont1 image1

cd myvol/

touch abc.txt

cat>abc.txt

hi this is rammohan.

Ctrl p q

**Method-1:**

* docker run -it --name cont2 --volumes-from cont1 --privileged=true ubuntu

**method-2:**

* docker run -it --name cont3 -v myvol1 ubuntu
* cd myvol1/
* touch file{1..5}
* docker run -it --name cont4 --volumes-from cont3 --privileged=true ubuntu

**method-3:volume mounting:**

* docker volume ls ---->list of volumes
* docker volume create volume-1 --->to create volume
* docker volume inspect volumename-->to get the docker volume information
* docker run -it --name cont5 --mount source=volume-1,destination=/volume-1 ubuntu
* docker volume rm volumename
* docker volume prune volumename: to delete unused containers

**docker system commands:**

thes are used to know the complete information of docker container

* docker system df -v
* docker inspect cont3 grep volume -image
* docker system prune: to remove unused objects of docker

**docker memory management:**

* containers uses host resources .by default we don't have any limits for containers
* we need to set it
* docker run -itd --name cont6 --memory="200mb" --cpus="0.2" ubuntu
* docker inspect cont6
* docker stats cont6

**Docker compose:**

Installation:

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

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

3.sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

4.docker-compose –version

vim Dockerfile

FROM ubuntu

RUN apt update -y

RUN apt install apache2 -y

COPY index.html /var/www/html/

CMD ["/usr/sbin/apachectl","-D","FOREGROUND"]

git clone https://github.com/abburimohan/html.git

cd index.html

mv index.html /root

docker build -t registr:v1 .

docker build -t recharge:v1 .

docker build -t dth:v1 .

docker build -t train:v1 .

vim docker-compose.yml

version: '3'

services:

register:

image: register:v1

ports:

- "81:80"

recharge:

image: recharge:v1

ports:

- "82:80"

dth:

image: dth:v1

ports:

- "83:80"

train:

image: train:v1

ports:

- "84:80"

docker-compose up -d---->to create and start containers

docker-compose down--->to stop and delete

docker-compose stop--->to stop the containers

docker-compose kill--->to kill the containers

docker-compose start---> to start the containers

docker-compose pause--->to pause the containers

docker-compose unpause--->to unpause the containers

docker-compose ps--->to show the containers created with compose file

docker-compose logs--->to show compose logs

docker-compose image--->to show images used for compose

docker-compose file supported names:

docker-compose.yml, docker-compose.yaml, compose.yml, compose.yaml

advantages:

we can create multiplecontainers at a time,reduce the mannual work

we can reuse the docker-compose file

**IP address of the docker container:**

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container-id

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' my-nginx-container

docker inspect <container\_id\_or\_name> | grep "IPAddress"

**Docket swarm**:

docker swarm:

**prerequisites:**

3 servers(1-master,2-workers),give all traffic enable

stop-1:

**on master**

hostname master

bash

**on worker1**

hostname worker1

bash

**on worker2**

hostname worker2

bash

step-2:

**on all nodes**

yum install docker -y

service docker start

**on master only**

docker swarm init

**copy the token and enter it in slaves**

goto master and enter the following command

docker node ls

**services**:

to copy the container from master to worker node we need to create service

service is a way of exposing application

we can manage multiple containers with the help of services

docker service create --name train --replicas 6 -p 81:80 ramu478/train:v1

docker service ls

docker service scale train=10

docker service ps train

docker service scale train=4

docker service inspect train

docker service rollback train

docker service logs train

**to detach node from cluster:**

docker swarm leave

**to remove node completely:**

docker node rm nodeid--->on master

**to retrieve the join token:**

docker swarm join-token manager

**difference between docker swarm and k8s:**

Kubernetes (K8s) and Docker Swarm are both container orchestration tools, but they differ in complexity and features:

* **Kubernetes** is more complex, offering advanced features like auto-scaling, load balancing, and high availability, with a strong ecosystem. It’s ideal for large-scale, production-grade deployments.
* **Docker Swarm** is simpler, integrated with Docker, and easier to set up, making it suitable for smaller deployments with basic orchestration needs.

In short: **Kubernetes is feature-rich and powerful but complex**, while **Docker Swarm is simpler and quicker to deploy** but limited in functionality.