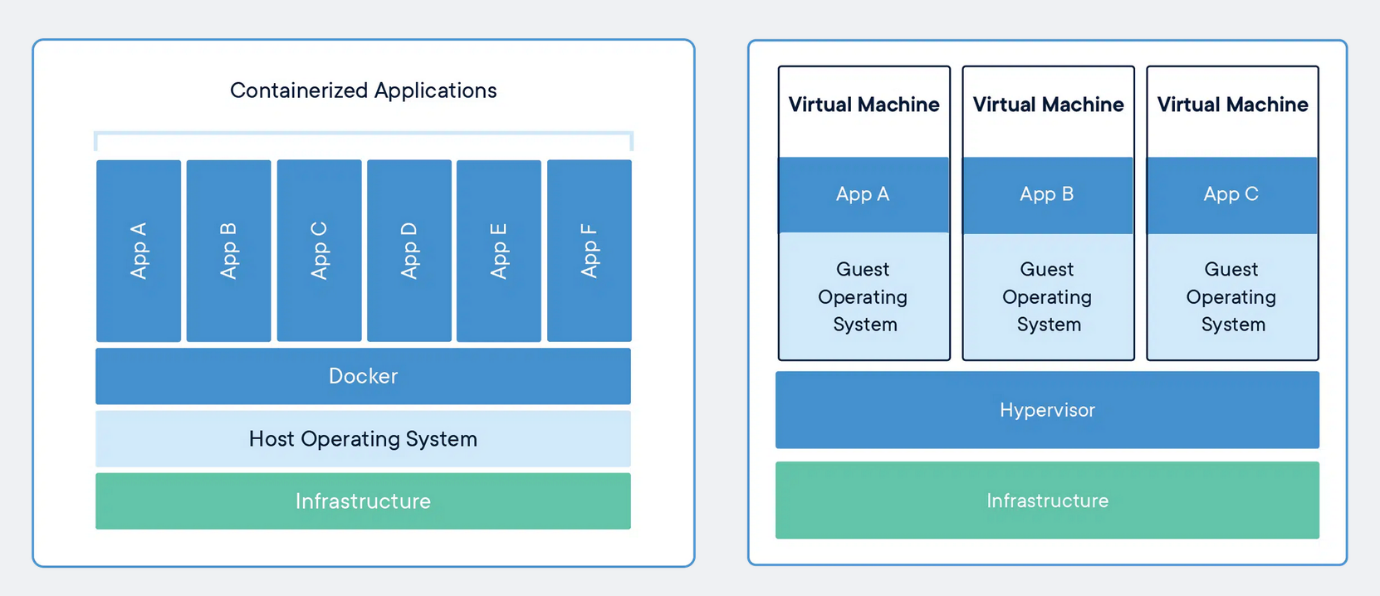
Docker

**What is a container ?**

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Ok, let me make it easy !!!

A container is a bundle of Application, Application libraries required to run your application and the minimum system dependencies.

[](https://user-images.githubusercontent.com/43399466/217262726-7cabcb5b-074d-45cc-950e-84f7119e7162.png)

**Containers vs Virtual Machine**

Containers and virtual machines are both technologies used to isolate applications and their dependencies, but they have some key differences:

1. Resource Utilization: Containers share the host operating system kernel, making them lighter and faster than VMs. VMs have a full-fledged OS and hypervisor, making them more resource-intensive.

2. Portability: Containers are designed to be portable and can run on any system with a compatible host operating system. VMs are less portable as they need a compatible hypervisor to run.

3. Security: VMs provide a higher level of security as each VM has its own operating system and can be isolated from the host and other VMs. Containers provide less isolation, as they share the host operating system.

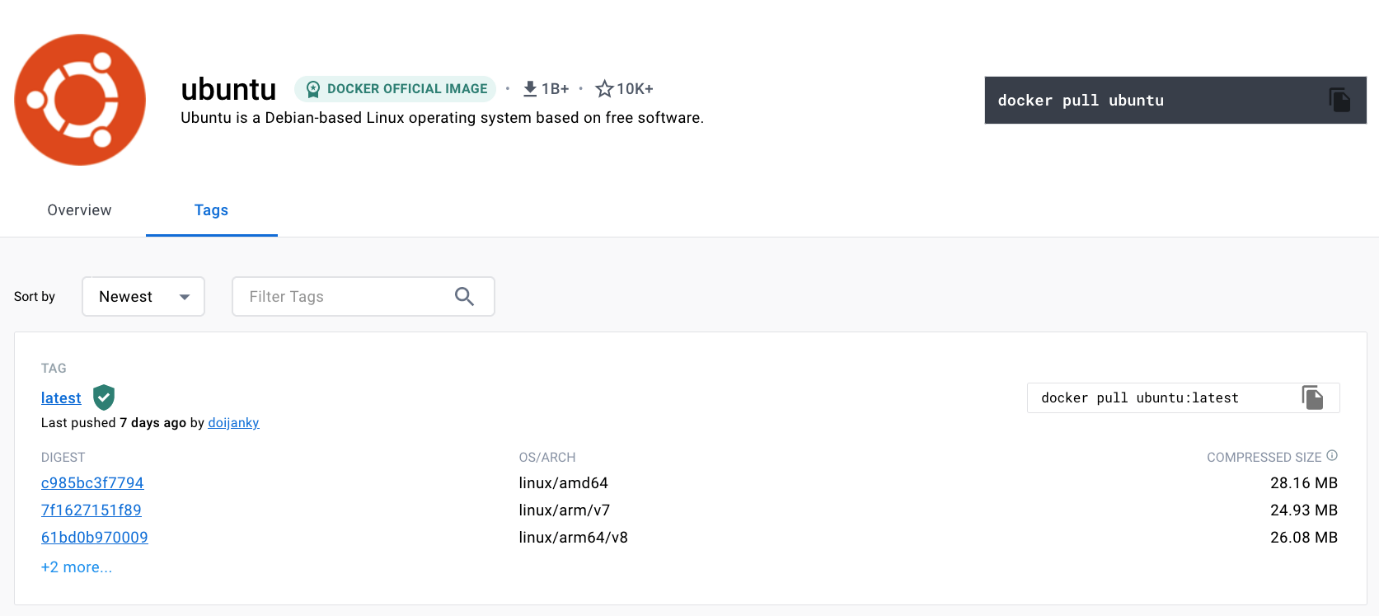
1. Management: Managing containers is typically easier than managing VMs, as containers are designed to be lightweight and fast-moving.

**Why are containers light weight ?**

Containers are lightweight because they use a technology called containerization, which allows them to share the host operating system's kernel and libraries, while still providing isolation for the application and its dependencies. This results in a smaller footprint compared to traditional virtual machines, as the containers do not need to include a full operating system. Additionally, Docker containers are designed to be minimal, only including what is necessary for the application to run, further reducing their size.

Let's try to understand this with an example:

Below is the screenshot of official ubuntu base image which you can use for your container. It's just ~ 22 MB, isn't it very small ? on a contrary if you look at official ubuntu VM image it will be close to ~ 2.3 GB. So the container base image is almost 100 times less than VM image.

[](https://user-images.githubusercontent.com/43399466/217493284-85411ae0-b283-4475-9729-6b082e35fc7d.png)

To provide a better picture of files and folders that containers base images have and files and folders that containers use from host operating system (not 100 percent accurate -> varies from base image to base image). Refer below.

**Files and Folders in containers base images**

/bin: contains binary executable files, such as the ls, cp, and ps commands.

/sbin: contains system binary executable files, such as the init and shutdown commands.

/etc: contains configuration files for various system services.

/lib: contains library files that are used by the binary executables.

/usr: contains user-related files and utilities, such as applications, libraries, and documentation.

/var: contains variable data, such as log files, spool files, and temporary files.

/root: is the home directory of the root user.

**Files and Folders that containers use from host operating system**

The host's file system: Docker containers can access the host file system using bind mounts, which allow the container to read and write files in the host file system.

Networking stack: The host's networking stack is used to provide network connectivity to the container. Docker containers can be connected to the host's network directly or through a virtual network.

System calls: The host's kernel handles system calls from the container, which is how the container accesses the host's resources, such as CPU, memory, and I/O.

Namespaces: Docker containers use Linux namespaces to create isolated environments for the container's processes. Namespaces provide isolation for resources such as the file system, process ID, and network.

Control groups (cgroups): Docker containers use cgroups to limit and control the amount of resources, such as CPU, memory, and I/O, that a container can access.

It's important to note that while a container uses resources from the host operating system, it is still isolated from the host and other containers, so changes to the container do not affect the host or other containers.

**Note:** There are multiple ways to reduce your VM image size as well, but I am just talking about the default for easy comparision and understanding.

so, in a nutshell, container base images are typically smaller compared to VM images because they are designed to be minimalist and only contain the necessary components for running a specific application or service. VMs, on the other hand, emulate an entire operating system, including all its libraries, utilities, and system files, resulting in a much larger size.

I hope it is now very clear why containers are light weight in nature.

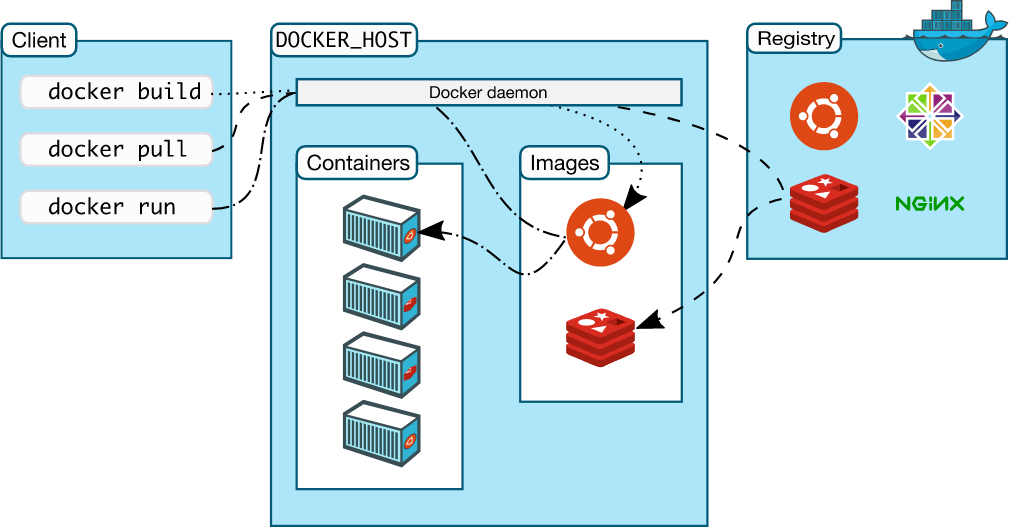
**Docker**

**What is Docker ?**

Docker is a containerization platform that provides easy way to containerize your applications, which means, using Docker you can build container images, run the images to create containers and also push these containers to container regestries such as DockerHub, Quay.io and so on.

In simple words, you can understand as containerization is a concept or technology and Docker Implements Containerization.

**Docker Architecture ?**

[](https://user-images.githubusercontent.com/43399466/217507877-212d3a60-143a-4a1d-ab79-4bb615cb4622.png)

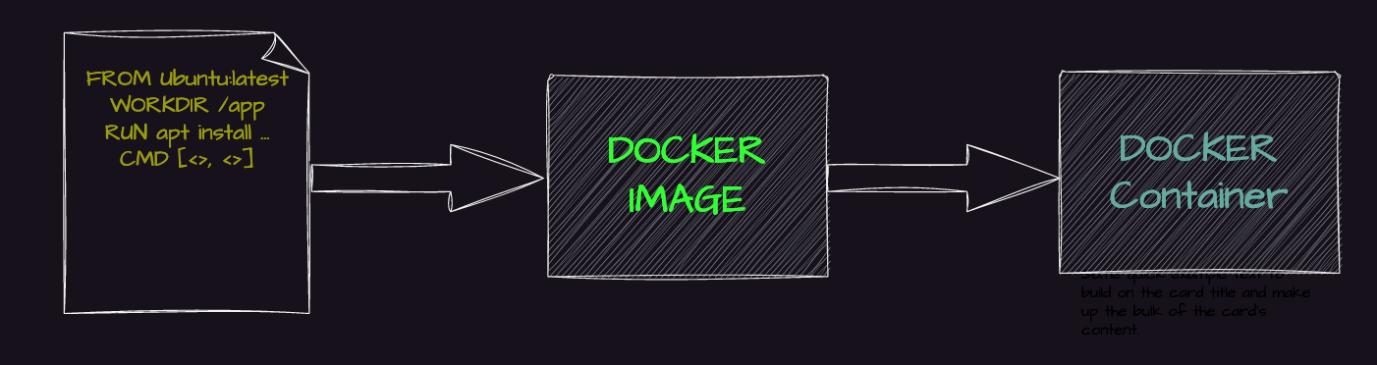
The above picture, clearly indicates that Docker Deamon is brain of Docker. If Docker Deamon is killed, stops working for some reasons, Docker is brain dead :p (sarcasm intended).

**Docker LifeCycle**

We can use the above Image as reference to understand the lifecycle of Docker.

There are three important things,

1. docker build -> builds docker images from Dockerfile
2. docker run -> runs container from docker images
3. docker push -> push the container image to public/private regestries to share the docker images.

[](https://user-images.githubusercontent.com/43399466/217511949-81f897b2-70ee-41d1-b229-38d0572c54c7.png)

**Understanding the terminology (Inspired from Docker Docs)**

**Docker daemon**

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

**Docker client**

The Docker client (docker) 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 dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

**Docker Desktop**

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

**Docker registries**

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry. Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

**Dockerfile**

Dockerfile is a file where you provide the steps to build your Docker Image.

**Images**

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

**INSTALL DOCKER**

A very detailed instructions to install Docker are provide in the below link

<https://docs.docker.com/get-docker/>

For Demo,

You can create an Ubuntu EC2 Instance on AWS and run the below commands to install docker.

sudo apt update

sudo apt install docker.io -y

**Start Docker and Grant Access**

A very common mistake that many beginners do is, After they install docker using the sudo access, they miss the step to Start the Docker daemon and grant acess to the user they want to use to interact with docker and run docker commands.

Always ensure the docker daemon is up and running.

A easy way to verify your Docker installation is by running the below command

docker run hello-world

If the output says:

docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create": dial unix /var/run/docker.sock: connect: permission denied.

See 'docker run --help'.

This can mean two things,

1. Docker deamon is not running.
2. Your user does not have access to run docker commands.

**Start Docker daemon**

You use the below command to verify if the docker daemon is actually started and Active

sudo systemctl status docker

If you notice that the docker daemon is not running, you can start the daemon using the below command

sudo systemctl start docker

**Grant Access to your user to run docker commands**

To grant access to your user to run the docker command, you should add the user to the Docker Linux group. Docker group is create by default when docker is installed.

sudo usermod -aG docker ubuntu

In the above command ubuntu is the name of the user, you can change the username appropriately.

**NOTE:** : You need to logout and login back for the changes to be reflected.

**Docker is Installed, up and running 🥳🥳**

Use the same command again, to verify that docker is up and running.

docker run hello-world

Output should look like:

....

....

Hello from Docker!

This message shows that your installation appears to be working correctly.

...

...

**Great Job, Now start with the examples folder to write your first Dockerfile and move to the next examples. Happy Learning :)**

**Clone this repository and move to example folder**

git clone https://github.com/iam-veeramalla/Docker-Zero-to-Hero

cd examples

**Login to Docker [Create an account with**[**https://hub.docker.com/**](https://hub.docker.com/)**]**

docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.

Username: abhishekf5

Password:

WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.

Configure a credential helper to remove this warning. See

https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

**Build your first Docker Image**

You need to change the username accoringly in the below command

docker build -t abhishekf5/my-first-docker-image:latest .

Output of the above command

Sending build context to Docker daemon 992.8kB

Step 1/6 : FROM ubuntu:latest

latest: Pulling from library/ubuntu

677076032cca: Pull complete

Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f

Status: Downloaded newer image for ubuntu:latest

---> 58db3edaf2be

Step 2/6 : WORKDIR /app

---> Running in 630f5e4db7d3

Removing intermediate container 630f5e4db7d3

---> 6b1d9f654263

Step 3/6 : COPY . /app

---> 984edffabc23

Step 4/6 : RUN apt-get update && apt-get install -y python3 python3-pip

---> Running in a558acdc9b03

Step 5/6 : ENV NAME World

---> Running in 733207001f2e

Removing intermediate container 733207001f2e

---> 94128cf6be21

Step 6/6 : CMD ["python3", "app.py"]

---> Running in 5d60ad3a59ff

Removing intermediate container 5d60ad3a59ff

---> 960d37536dcd

Successfully built 960d37536dcd

Successfully tagged abhishekf5/my-first-docker-image:latest

**Verify Docker Image is created**

docker images

Output

REPOSITORY TAG IMAGE ID CREATED SIZE

abhishekf5/my-first-docker-image latest 960d37536dcd 26 seconds ago 467MB

ubuntu latest 58db3edaf2be 13 days ago 77.8MB

hello-world latest feb5d9fea6a5 16 months ago 13.3kB

**Run your First Docker Container**

docker run -it abhishekf5/my-first-docker-image

Output

Hello World

**Push the Image to DockerHub and share it with the world**

docker push abhishekf5/my-first-docker-image

Output

Using default tag: latest

The push refers to repository [docker.io/abhishekf5/my-first-docker-image]

896818320e80: Pushed

b8088c305a52: Pushed

69dd4ccec1a0: Pushed

c5ff2d88f679: Mounted from library/ubuntu

latest: digest: sha256:6e49841ad9e720a7baedcd41f9b666fcd7b583151d0763fe78101bb8221b1d88 size: 1157

What is docker in AWS?

Answer: Docker is a software platform which will allow to build, test, and deploy applications quickly. Running Docker on AWS will provide developers and admins a highly reliable, low-cost way for building, ship, and run distributed applications at any scale

What is the method for creating a Docker container?

Answer:We will use any of the specific Docker images to create a Docker container using the below command.

docker run -t -i command name

This command not will create the container but also start it.

Difference between Docker Image and container?

Answer: Docker container is the runtime instance of docker image.

Docker Image can not have a state and its state never changes because it will be just set of files whereas docker container will have its execution state.

What is a Docker Hub?

Answer: Docker Hub can be considered as a cloud registry which lets us link the code repositories, create the images, and test them. We will also store our pushed images, or we will link to the Docker Cloud, therefore that the images will be deployed to the host. We have a centralized container image discovery resource which will be used for the collaboration of our teams, automating the workflow and distribution, and changing management by creating the development pipeline.

Docker Vs VM (Virtual Machine)

Virtual Machines Vs Docker Containers

Virtual Machines Docker Containers

Need more resources Less resources are used

Process isolation is done at hardware level Process Isolation is done at Operating System level

Separate Operating System for each VM Operating System resources can be shared within Docker

VMs can be customized. Custom container setup is easy

Takes time to create Virtual Machine Creation of docker is very quick

Booting takes minutes CBooting is done within seconds

What platforms does Docker run on?

Answer: Docker will currently available on the following platforms and also on the following Vendors or Linux:

1. Ubuntu 12.04, 13.04

2. Fedora 19/20+

3. RHEL 6.5+

4 CentOS 6+

5. Gentoo

6. ArchLinux

7. openSUSE 12.3+

8. CRUX 3.0+

Docker will currently available and run on the following Cloud environment setups as following:

1. Amazon EC2

2. Google Compute Engine

3. Microsoft Azure

4. Rackspace

How to stop and restart the Docker container?

Answer: The following command will be used to stop a certain Docker container with the container id as

CONTAINER\_ID:

docker stop CONTAINER\_ID

The following command will be used to restart a certain Docker container with the container id as

CONTAINER\_ID:

docker restart CONTAINER\_ID

How is Docker different from Hypervisor?

Answer: Hypervisor will require to have extensive hardware to function properly, while Docker will run on the actual operating system. This will allow Docker to be exceptionally fast and perform tasks in a fluid manner – something which Hypervisor tends to lack.

What is a Dockerfile?

Answer: A Dockerfile is a set of instructions. Developers provided Docker with such instructions therefore the program could do the job correctly, with those specific parameters in mind.

What is the command to run the image as a container?

Answer: $ sudo docker run -i -t alpine /bin/bash

What is Docker Engine?

Answer: Docker daemon or Docker engine will represent the server. The docker daemon and the clients will run on the same or remote host, which will communicate through command-line client binary and

What is a Docker Container and its advantages?

Answer: Docker containers will include the application and all of its dependencies. It will share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers will not required any specific infrastructure, they will run on any infrastructure, and in any cloud. Docker containers will be basically runtime instances of Docker images.

Following are major advantage of using Docker Container:-

1. It will offer an efficient and easy initial set up.

2. It will allow to describe application lifecycle in detail.

3. Simple configuration and interacts with Docker Compose.

4. Documentation will provide every bit of information.

Difference between virtualization and containerization?

Answer: Containers will provide an isolated environment to run the application. The entire user space will be explicitly dedicated to the application. Any changes which are made inside the container will never reflected on the host or even other containers running on the same host. Containers will be an abstraction of the application layer. Each container has a different application.

In virtualization, hypervisors will provide an entire virtual machine to the guest including Kernal. Virtual machines will be an abstraction of the hardware layer. Each VM is a physical machine.

* Explain how you can clone a Git repository via Jenkins?

Answer: To clone a Git repository via Jenkins, we have to enter the e-mail and user name for Jenkins system. To do that we have to switch into job directory and execute the “git config” command.

* What is a Docker Container and its advantages?

Answer: Docker containers will include the application and all of its dependencies. It will share the kernel with other containers, running as isolated processes in user space on the host operating system. Docker containers will not required any specific infrastructure, they will run on any infrastructure, and in any cloud. Docker containers will be basically runtime instances of Docker images.

Following are major advantage of using Docker Container:-

1. It will offer an efficient and easy initial set up.

2. It will allow to describe application lifecycle in detail.

3. Simple configuration and interacts with Docker Compose.

4. Documentation will provide every bit of information.

. Explain Docker Architecture?

Answer: Docker Architecture will consist of a Docker Engine that is a client-server application:-

1. A server which is a type of long-running program which is called a daemon process ( the docker command ).

2. A REST API which will specify interfaces that programs will use to talk the daemon and instruct it what to do.

3. A command-line interface (CLI) client (the docker command) The CLI will use the Docker REST API to control or interact with Docker daemon applications use the underlying API and CLI.

Is Kubernetes better than Docker?

Answer: Kubernetes and Dicker isn’t an alternative of each other. Quite the contrary; Kubernetes will run without Docker and Docker will function without Kubernetes. Kubernetes will benefit greatly from Docker and vice versa. Docker is a standalone software which will be installed on any computer to run containerized applications.

What is the Lifecycle of Docker container?

Answer: The Lifecycle of Docker Container with CLI is as following:

1. Create a Container.

2. Run the created Container.

3. Pause the processes running inside the Container.

4. Unpause the processes running inside the Container.

5. Start the Container, if exists in a stopped state.

6. Stop the Container as well as the running processes.

7. Restart the Container as well as the running processes.

8. Kill the running Container.

9. Destroy the Container, only if it exists in a stopped state.

Why use Docker?

Answer: 1. A user can quickly build, ship, and run its applications.

2. A single operating system kernel will run all containers.

3. Docker container is more light-weight than the virtual machines.

4. A user will deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.

3. List the most used commands of Docker.

Answer: 1. ps lists the running containers.

2. dockerd launches Docker Daemon.

3. build is used to build an image from a DockerFile.

4. create is used to create a new image form container’s changes.

5. pull is used to download a specific image or a repository.

6. run is used to run a container.

7. logs display the logs of a container.

8. rm removes one or more containers.

9.rmi removes one or more images.

10. stop is used to stop one or more container.

11. kill is used to kill all running containers.

**Docker Commands**

**Installation**

* sudo apt update
* sudo apt install docker.io
* sudo service docker start
* sudo systemctl enable docker
* sudo systemctl status docker

**Docker Hub**

* **docker pull img\_name**: download an image from Docker Hub

**Running Containers**

* **docker run image-name/image-id**: create a container from an image
* **docker run img\_name**: create a container
* **docker run --detach/-d img\_name**: pull, start, and create a container
* **docker run -d --name cont\_name container\_id**: create a container with a name

**Listing Containers and Images**

* **docker ps**: show running containers
* **docker ps -a/--all**: show all containers (running and stopped)
* **docker container ls**: show all containers (running and stopped)
* **docker images**: show a list of images

**Deleting Containers and Images**

* **docker rmi img\_name**: delete an image
* **docker rmi $(docker images -a) / docker image prune / docker rmi $(docker image ls)**: remove all images
* **docker stop container\_id**: stop a container
* **docker rm container\_id**: delete a stopped container
* **docker rm $(docker ps -a / docker container ls -a) / docker container prune**: delete all stopped containers

**Working with Containers**

* **docker exec -it container\_id /bin/bash**: start working in a container
* **docker inspect container\_id**: show all information about a container
* **docker run -d --name name nginx**: create a container with a name
* **docker history img\_id**: show all layers of an image

**Port Mapping**

* **docker run -d -p80(ec2-port):80(container-port) img\_id**: map a port for an existing image
* **docker run -d --name name -p80(ec2-port):(container-port) nginx**: create a container with port mapping

**Creating Images**

* **docker commit container\_id name**: create an image from a container
* **docker save > img\_name.tar**: create a tar file of an image
* **docker export container\_id > file\_name.tar**: create a tar file of a container

**Volumes**

* **docker run -d --name name -p80(ec2-port):(container-port) -v path of directory:project directory path in container img\_name**: create a container with a volume
* **docker volume create vol\_name**: create a named volume

**Environment Variables**

* **docker run -d --name merabaladb1 -e MYSQL\_ROOT\_PASSWORD='Pass@123' MySQL**: create a MySQL container with environment variables
* **docker run -d --name sqlvol -v /home/ubuntu/sqlvol:/var/lib/mysql -e MYSQL\_ROOT\_PASSWORD='pass@123' -e MYSQL\_DATABASE="facebook" MySQL**: create a MySQL container with a database name and mount with a bind volume

**Linking Containers**

* **docker run -d --name wordpress -p80:80 -e WORDPRESS\_DB\_HOST=mydb -e WORDPRESS\_DB\_USER=root -e WORDPRESS\_DB\_PASSWORD=pass123 -e WORDPRESS\_DB\_NAME=db1 --link mydb:mysql wordpress**: link a MySQL container to a WordPress container

**Dockerfile**

**Components of Dockerfile**

* **FROM**: use for base images or to pull an image (can be used multiple times)
* RUN: for installing software or running Linux commands (can be used multiple times)
* **EXPOSE**: to open a port number
* **COPY**: to copy files and directories from the host to the image
* **ENV**: to set environment variables
* **CMD**: specifies the command to run when a container is run from the image (can only be used once)
* **ENTRYPOINT**: specifies the command to run when a container is run from the image, but allows additional arguments to be passed in (can only be used once)
* **ADD**: copies files from the host to the image, downloads zip or tar files from a given link, and extracts them automatically
* **ARG**: defines a variable that is passed to the container while building the image
* **VOLUME**: creates a volume, sets a volume
* **WORKDIR**: sets the working directory
* **MAINTAINER**: sets the name and email of the author/user
* **LABEL**: adds metadata (data about data)
* **USER**: sets the user (root, ec2-user, docker, etc.)
* **HEALTHCHECK**: specifies the path for a health check or checks the health of a mentioned URL
* **SHELL**: specifies the shell to be used to run commands
* **STOPSIGNAL**: specifies the signal to be sent to the container to stop it gracefully
* **ONBUILD**: specifies the instruction to be used when the

**Docker Build**

* **docker build -f file\_name .**: run a file (if file name is different)
* docker build -t tag\_name -f file\_name . --> run file with tag
* docker system df --> for check container space

**Docker Network**

* **docker network ls**: show a list of networks
* **docker network create name**: create a network
* **docker network inspect name**: show all information about a network
* **docker network rm name**: delete a network
* **docker run -d --name cont\_name --network network\_name image\_id**: create a container in a network
* **docker network connect network\_name cont\_name**: add a container to a network
* **docker network disconnect network\_name cont\_name**: remove a container from a network

**Docker Compose**

* **docker-compose up -d**: run a compose file (if the file name is **docker-compose.yml**)
* **docker-compose -f file\_name up -d**: run a compose file with a different name
* **docker-compose down**: delete containers

**Docker System**

* **docker system df**: check container spac

Create ubuntu container in docker

* docker run -d --name myubuntu ubuntu

to create dockerfile

* we can customise image
* to use base image modification
* vi Dockerfile

Dockerfile

FROM ubuntu  
RUN apt update -y  
RUN apt install nginx -y  
EXPOSE 80                                                                                                                            WORKDIR /var/www/html  
RUN touch index.html  
RUN echo "This is my container using dockerfile" > index.html  
CMD ["nginx", "-g", "daemon off;" ]

* to run this file - docker build .
* docker images
* to change docker image name - docker tag imageid newname
* to create container - docker run -d --name mycont -p80:80 myubuntu(image name)
* hit ip address to response dockerfile
* example 2

vi Dockerfile

* FROM nginx
* EXPOSE 80
* WORKDIR /usr/share/nginx/html
* RUN touch index.html
* RUN echo "This is my container using nginx base images dockerfile" > index.html
* CMD ["nginx", "-g", "daemon off;" ]

--- docker build .

* docker run -d --name nginxcont -p80:80 imageid

----

FROM - use for base imges or to pull image (multiple time use)  
RUN - for install any software or for all Linux command (multiple time use)  
EXPOSE - to open port no  
COPY - to copy file & directory from host to images  
ENV - to set environment variables  
CMD - specifier the command to RUN when a container is run from images (use only one times)  
ENTRYPOINT- specifies the command to run when a container is run from images but allow additional argument to be passed in (use only one times)  
ADD - Copies files from host to images download zip or tar files from given link & extract it auto.  
ARG - Define variable that passed to container while building images  
VOLUME - create volume, to set volume

WORKDIR - to set working directory  
MAINTAINER - to set name & email of author/ user   
LABEL - To add metadata (data about data)  
USER - To set user (root, ec2-user, docker etc)  
HEALTHCHECK - to specifies path for healthcheck or check health of mentioned url  
SHELL-specifies shell to be used to run command  
STOPSIGNAL - Specifies the signed to sent to container want to stop container gracefully  
ONBUILD - Specifies the instruction to be used when we uses this images as base images for another images

Node js

Mkdir nodeapp

cd nodeapp/

vi dockerfile

* FROM node
* WORKDIR /usr/src/app
* COPY package\*.json ./
* RUN npm install
* EXPOSE 3000
* CMD ["node", "app.js"]
* Or

FROM node

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install --production

copy . .

EXPOSE 3000

CMD ["node", "index.js"]

* index.js

// Import the http module  
const http = require('http');  
  
// Create a server object  
const server = http.createServer((req, res) => {  
  // Write a response header  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  // Write a response body  
  res.end('Hello from Docker!\n');  
});  
  
// Listen on port 3000  
server.listen(3000, () => {  
  // Log a message when the server starts  
  console.log('Server running at http://localhost:3000/');  
});

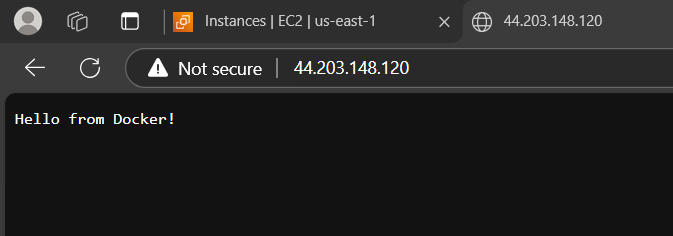
* vi index.js

// Import the http module  
const http = require('http');  
  
// Create a server object  
const server = http.createServer((req, res) => {  
  // Write a response header  
  res.writeHead(200, {'Content-Type': 'text/plain'});  
  // Write a response body  
  res.end('Hello from Docker!\n');  
});  
  
// Listen on port 3000  
server.listen(3000, () => {  
  // Log a message when the server starts  
  console.log('Server running at http://localhost:3000/');  
});

* vi package.json

{  
  "name": "my-node-app",  
  "version": "1.0.0",  
  "description": "A simple Node.js application with Express",  
  "main": "app.js",  
  "scripts": {  
    "start": "node index.js"  
  },  
  "dependencies": {  
    "express": "^4.18.4"  
  },  
  "author": "Your Name",  
  "license": "MIT"  
}

* docker build -f dockerfile -t myimg .
* docker run -d -p80:3000 --name mycont myimg
* output of node js project



Python using flask and dockerfile create website

Dockerfile for python

FROM python  
WORKDIR /app  
COPY requirements.txt .  
RUN pip install -r requirements.txt  
EXPOSE 80  
COPY . ./  
CMD ["python", "server.py"]

* server.py

#!/usr/bin/python

import time  
from flask import Flask  
app = Flask(\_\_name\_\_)

START = time.time()

def elapsed():  
    running = time.time() - START  
    minutes, seconds = divmod(running, 60)  
    hours, minutes = divmod(minutes, 60)  
    return "%d:%02d:%02d" % (hours, minutes, seconds)

@app.route('/')  
def root():  
    return "Hello World (Python)! (up %s)\n" % elapsed()

if \_\_name\_\_ == "\_\_main\_\_":  
    app.run(debug=True, host="0.0.0.0", port=80)

* requirements.txt

flask

* docker build -t mypyimg .
* docker run -d -p80:80 --name pycont mypyimg

example 3

mysql dockerfile

FROM mysql  
MAINTAINER amrita  
ENV MYSQL\_ROOT\_PASSWORD Pass@123  
ENV MYSQL\_DATABASE wordpress  
EXPOSE 3306  
CMD ["mysqld"]

* docker build -f mysqldockefile -t mysqlimg .
* docker run -d --name sanketmysqlcont mysqlimg
* sudo mysql -p -u root
* Pass@123

Wordpresss

* Mkdir wp
* vi wpdockerfile

FROM wordpress  
ENV WORDPRESS\_DB\_HOST mysqlcont  
ENV WORDPRESS\_DB\_USER root  
ENV WORDPRESS\_DB\_PASSWORD Pass@123  
ENV WORDPRESS\_DB\_NAME wordpress  
EXPOSE 80  
CMD ["apache2-foreground"]

or

FROM wordpress  
ENV WORDPRESS\_DB\_HOST mysqlcont  
ENV WORDPRESS\_DB\_USER root  
ENV WORDPRESS\_DB\_PASSWORD Pass@123  
ENV WORDPRESS\_DB\_NAME wordpress  
EXPOSE 80

* New task

Mkdir wp

Vi mysqldockerfile

FROM mysql

MAINTAINER amrita

ENV MYSQL\_ROOT\_PASSWORD Pass@123

ENV MYSQL\_DATABASE wordpress

EXPOSE 3306

CMD ["mysqld"]

-Create dockerfile to image

docker build -f mysqldockerfile -t mysqlimg .

* Convert image to container

docker run -d --name mysqlcont mysqlimg

* docker ps

now wordpress container

ls

vi wpdockerfile

FROM wordpress

ENV WORDPRES\_DB\_HOST mysqlcont

ENV WORDPRESS\_DB\_USER root

ENV WORDPRESS\_DB\_PASSWORD Pass@123

ENV WORDPRESS\_DB\_NAME wordpress

EXPOSE 80

CMD ["apache2-foreground"]

Wpdockerfile convert into image

docker build -f wpdockerfile -t wpimg .

* docker images

docker run -d -p80:80 root@ip-172-31-85-41:/home/ubuntu/wp# docker run -d -p80:80 --name wp --link mysqlcont:mysqlimg wpimg

* new task
* - Shell scripting

Vi shellscipting.sh

#! /bin/bash

mysql -u root -pPass@123 << EOF

create database newdb; use newdb; create table posts(id int, status varchar (200),posturl varchar (30)); insert into posts (id,status,posturl) values (1, "I am not good", "https:test.com");

>>EOF

using shell script file in mysql Dockerfile (when your shell script file in same as Dockerfile location)

FROM mysql  
MAINTAINER amrita  
ENV MYSQL\_ROOT\_PASSWORD Pass@123  
EXPOSE 3306  
COPY mysh.sh /docker-entrypoint-initdb.d  
RUN chmod 777 /docker-entrypoint-initdb.d/mysh.sh  
CMD ["mysqld"]

-\*\*\* docker day 11

New task

Docker network

* Commands

Docker network ls – to show networks

* To create network

Docker network create name

Ex - docker network create frontend

Bydefault it is bridge network

* To show the range firstly inspect them

docker network inspect frontend(name)

* To create nginx container in frontend
* 1st container

Docker run -d --name nginxcont -p80:80 –network frontend nginx

* To create nginx container in backend(name)
* 2nd container

Docker run -d -p9000 --name phpcont php:7.4 -fpm

* To add this container php in frontend

docker network connect frontend phpcont