Docker 101: A fresher's Ultimate Guide in one PDF

What is Docker?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

- Docker release March 2013 by Solomon Hykes and Sebastian
- Docker is a platform as a service that uses OS-level virtualization
- Docker is an open-source centralized platform designed to create deploy and run applications
- Docker uses a container on the Host OS to run applications. It allows applications to use the same host computer rather than creating a whole Virtual OS
- We can install Docker on any OS but Docker Engine runs natively on Linux distribution.
- Docker is written in the Go language.
- Docker is a tool that performs OS-level Virtualization, also known as centralization.

Advantage of Docker

- No pre-allocation of RAM
- Continues Integration (CI) Efficiency -> Docker enables you to build a container image and use that same image across every step of the deployment process.
- Less cost
- It is light in weight
- It can re-use the image
- It can run on physical H/W, Virtual H/W, or on cloud.
- It took very little time to create a container.

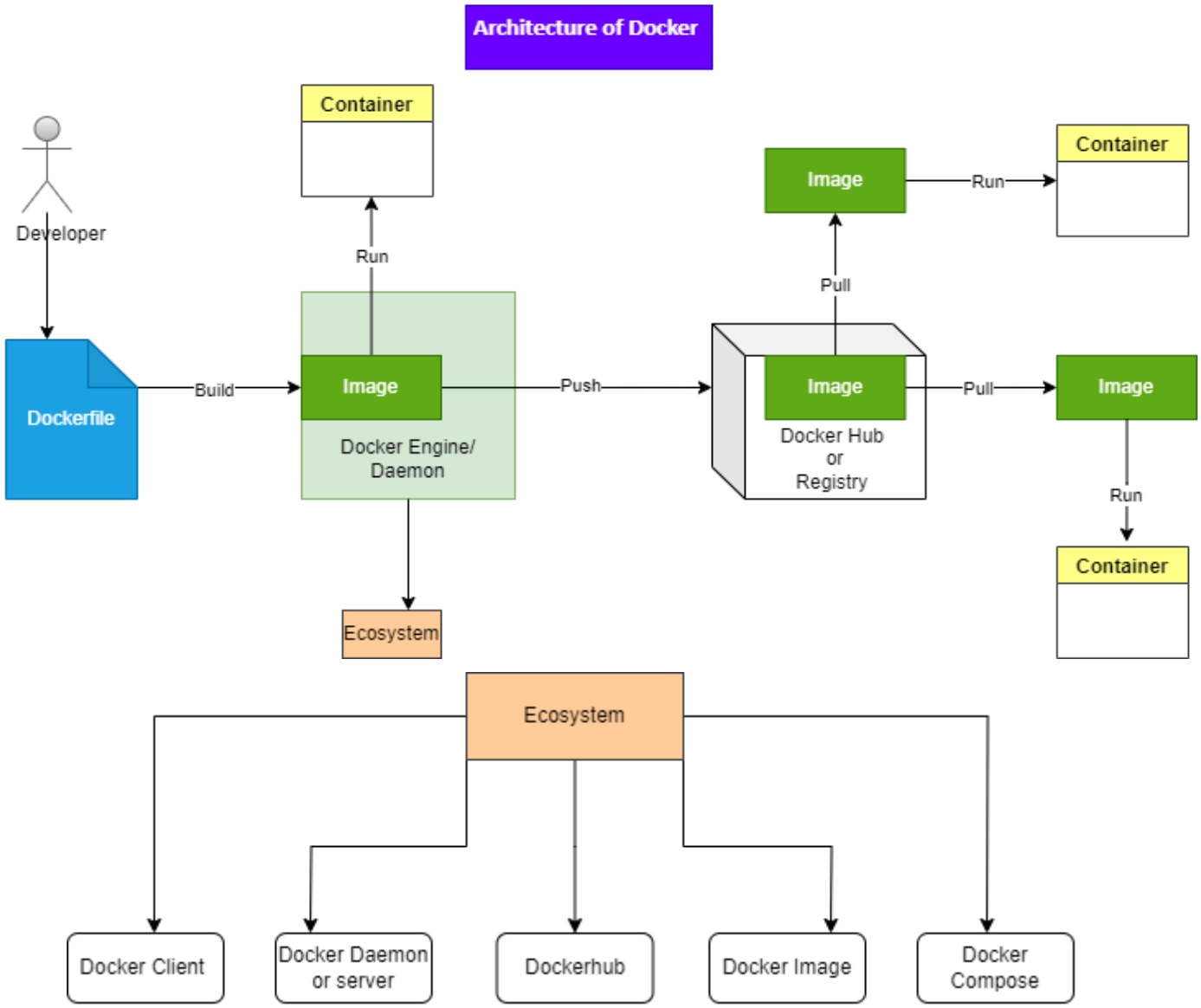
Disadvantages of Docker

- Docker is not a good solution for applications that require rich GUI
- Difficult to manage large amounts of containers
- Docker does not provide cross-platform compatibility means if an application is designed to run in a docker container on Windows, then it can`t run on Linux or vice-versa
- No solutions for Data recovery and backup
- Docker is suitable when the development OS and testing OS are the same.

Why use Docker?

- **Consistency:** Ensure applications run the same way in various environments.
- **Portability:** Easily move applications between different machines.
- **Efficiency:** Use resources more effectively with lightweight containers.
- **Isolation:** Run applications independently, avoiding conflicts.
- **Ease of Deployment:** Streamline deployment with standardized container images.
- **Scalability:** Scale applications easily using orchestration tools.
- **DevOps Integration:** Support for DevOps practices and CI/CD pipelines.
- **Community Support:** Large community and repository for sharing containerized applications.
- **Microservices Ready:** Ideal for microservices architecture, enabling modular development.
- **Version Control:** Versioned images for easy tracking and rollback.

Architecture of Docker



Docker Client

- Docker users can interact with the docker daemon through a client (CLI)
- The Docker client uses CLI and Rest API to communicate with the Docker daemon
- When a client runs any server command on the docker client terminal, the client terminal sends these docker commands to the docker daemon
- The client can communicate with more than one daemon.

Docker Daemon

- Docker daemon runs on the Host OS
- It is responsible for running container to manage docker services
- Docker daemon can communicate with other daemon

Docker Hub/ Registry

- The Docker registry manages and stores the docker images
- There are two types of registries in the docker
  Public registry: is also called docker hub
  Private registry: it is used to share images within the enterprise

Docker Images

Docker image is a lightweight, standalone package that contains all the necessary components (code, runtime, libraries, and system tools) to run a software application. It serves as a blueprint for creating Docker containers, providing consistency and portability across different environments. Images are created using Dockerfiles, stored in registries like Docker Hub, and can be easily pulled, pushed, and shared.

Or

Docker images are the read-only binary templates used to create a docker container

Or

Single file with all dependencies and configuration required to run a program

Ways to create a Images (Three ways)

1. Take images from Docker Hub
2. Create image form Docker file
3. Create image from existing docker container

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the application's services, networks, and volumes, allowing for easy management of complex applications.

Docker File

- A Dockerfile is indeed a text file containing a set of instructions. These instructions define how to build a Docker image step by step.
- Dockerfiles automate the process of creating Docker images, ensuring consistency and reproducibility in the deployment of applications.

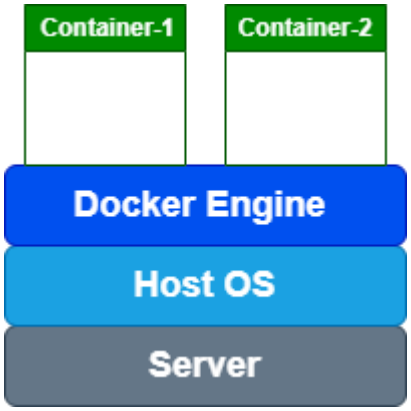
Steps for Dockerfile Usage:

1. Create a File Named Dockerfile
2. Add Instructions in the Dockerfile
3. Build a Dockerfile to Create an Image
4. Run the Image to Create a Container

Docker Container

- Docker containers are like portable, self-contained boxes for software applications.
- They pack code, runtime, libraries, and tools in a single container.
- Containers are created from Docker images, acting as pre-made blueprints.
- Containers provide a reliable and consistent environment for software to run.
- A vital tool for ensuring software consistency in modern development and deployment.

The container itself doesn't have its own operating system.
It operates under the assumption that it has no separate operating system. However, yes, it does contain OS-related files inside, but they are so minimal and lightweight that they are considered negligible.



Diving into Docker Practical: A Hands-On Guide

Going forward, all the practical exercises will be conducted on an AWS EC2 Linux instance. Additionally, it's essential to have some basic knowledge of Linux before diving into these activities
And if you don't have an AWS account, you can create one easily. Watch a quick tutorial on YouTube to guide you through the simple process of setting up an AWS account. It only takes a few minutes.

Connect to your EC2 instance
Step 1: Update Packages: Before installing Docker, it's a good practice to update the package index.

```
$ yum update
```

Step 2. Installing Docker on Amazon Linux
Install Docker using the package manager for your distribution. For Amazon Linux, you can use yum.

```
$ yum install docker -y

      Or for amazon linux 2

$ sudo amazon-linux-extras install docker
```

Step 3: Start and Enable its Service: Docker service will not be started and enabled by default after you complete its installation. We have to do that manually.

```
$ service docker start
```

Now, if you also want Docker to start automatically with system boot then use this command:

```
$ sudo systemctl enable docker
```

To check and confirm the service is running absolutely fine, use:

```
$ service docker status
```

To check the docker version

```
$ docker -v
     Or
$ docker --version
```

Pulling your first Docker image

```
$ docker pull <image_name>

Eg: docker pull ubuntu
```

To find out images in docker hub (Find more about Docker Hub at the end of the document.**)**

```
$ docker search <image_name>

Eg: docker search ubuntu
```

To see all images present in your local machine

```
$ docker images
```

To create a container from image

```
$ docker run -it <image_name> /bin/bash

Eg: docker run -it ubuntu /bin/bash
```

(Run means create and start the container) (-it = interactive mode terminal)
(*Every time new container is created)

To give name to container

```
$ docker run -it --name <container_name> <image_name> /bin/bash

Eg: docker run -it --name linkedin ubuntu /bin/bash
```

(run means create and start container) (-it = interactive mode terminal)(--name = give name to your container)

Upon executing above command, you'll seamlessly enter the container environment.
Post-Command Task
I have a task for you: explore the command that allows you to create a container exclusively without entering it.

You are in the container and want to see container info

```
$ cat /etc/os-release
```

To exit container

```
$ exit
```

*Container shuts down as soon as it comes out of the container.

To see lists of all running containers.

.

```
$ docker ps
```

(ps = process status)

To see lists of all containers, including stopped ones

```
$ docker ps -a
```

(ps = process status , -a = showing running and stop both containers)

Now create image of this container (linkedin)

```
$ docker commit <container_name> <new_image_name>

Eg: docker commit linkedin newimage

$ docker images : show all images
```

To start container

```
$ docker start <container_name>

Eg: docker start linkedin
```

You can find stoped container list by using **`docker ps -a`**

To enter a running container, first use the given command to start the container, and then use a separate command to access its interactive shell.

```
$ docker attach <container_name>

Eg: docker attach linkedin
```

To stop Container

```
$ docker stop <container_name>

Eg: docker stop linkedin
```

To delete container

```
$ docker rm <container_name>

Eg: docker rm linkedin
```

(rm = remove) but running container will not be deleted)

If you want to see the difference between the base image and changes on it then

```
$ docker diff <container_name>
```

Dockerfile

- A Dockerfile is indeed a text file containing a set of instructions. These instructions define how to build a Docker image step by step.
- Dockerfiles automate the process of creating Docker images, ensuring consistency and reproducibility in the deployment of applications.

Steps for Dockerfile Usage:

1. Create a File Named **Dockerfile**
2. Add Instructions in the Dockerfile
3. Build a Dockerfile to Create an Image
4. Run the Image to Create a Container

The Dockerfile supports the following instructions:

Instruction	Description
FROM	Create a new build stage from a base image.
RUN	Execute build commands.
MAINTAINER	Specify the author of an image.
COPY	Copy files and directories.
ADD	Add local or remote files and directories.
EXPOSE	Describe which ports your application is listening on.
WORKDIR	Change working directory.
CMD	Specify default commands.
ENTRYPOINTS	Specify default executable.
ENV	Set environment variables.
VOLUME	Create volume mounts.
USER	Set user and group ID.
LABEL	Add metadata to an image.
ONBUILD	Specify instructions for when the image is used in a build.
SHELL	Set the default shell of an image.
ARG	Use build-time variables.
HEALTHCHECK	Check a container's health on startup.
STOPSIGNAL	Specify the system call signal for exiting a container.

Our First Dockerfile

```
$ vi Dockerfile
……
FROM Ubuntu
RUN echo “Learning Docker”> /tmp/testfile

:wq
……

(echo “Learning Docker”> /tmp/testfile it means write “Learning Docker” in file name testfile in tmp folder)
```

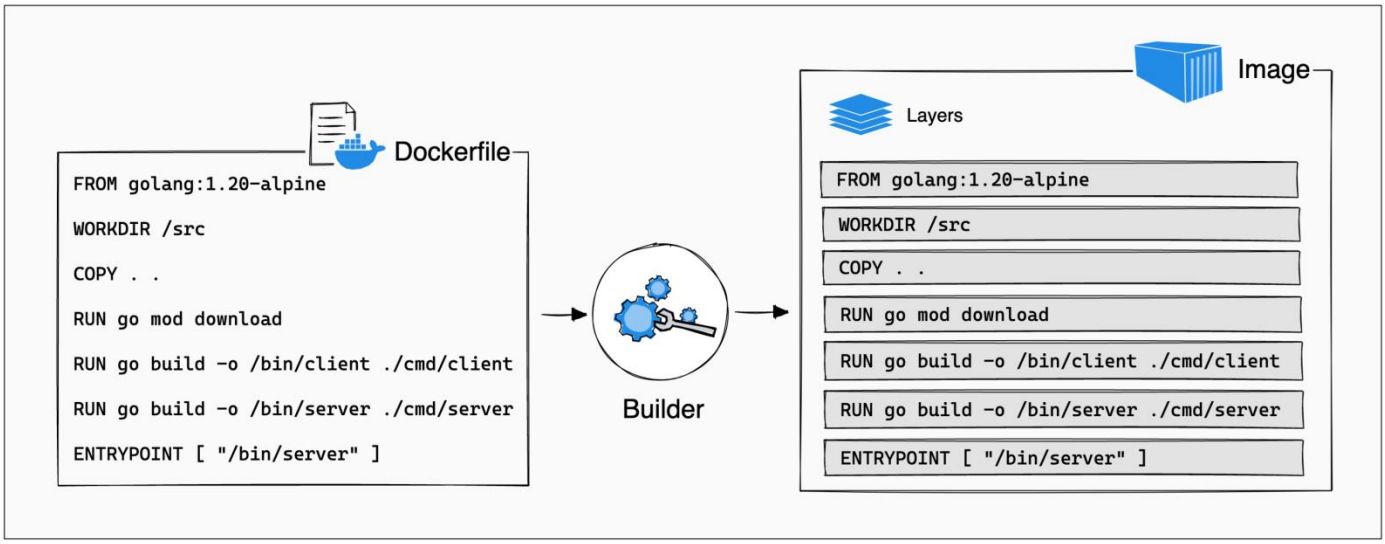
To create image out of Dockerfile

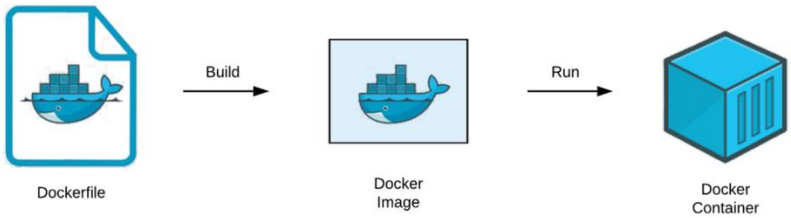
```
$ docker build -t <image_name> .

Eg: docker build -t learningDocker .
```

(-t = use for tag and .(dot) Use the current directory to build a Docker image using the Dockerfile located in the current directory)

Now you can see image by using **$ docker images**
Also you can try to create a container from this image ☺

Now you can play with different Dockerfile and try to do experiment by yourself ☺





Docker Volume

Docker volume is like a special folder that Docker containers can use to save and share data. Imagine it as a shared space where containers can put their files, and even if the containers stop or disappear, the data in that space stays around.

So, if you have different containers that need to work together or need a place to store important information, you can create a Docker volume. It's a way for these containers to talk to each other and share data, making sure nothing important gets lost when the containers are turned off or removed.

- Volume is simply a directory inside our container
- Firstly we have to declare the directory as a volume and then share volume
- Even if we stop container, still we can access volumes.
- Volume will be created in one container.
- You can declare a directory as a volume only while creating container.
- You can't create volume from existing container.
- You can share one volume across any number of container
- Volume will not be included when you update an image.
  (If an image is created from a container and a new container is created from that image, the volume directory in the new container won't be shared or connected with the original container.)
- You can mapped volume in Two ways:
  - Host to Container (A host is simply a device, like a computer or a server, that connects to the internet or a network.)
  - Container to Container

Benefits of Volume

- Decupling container from storage
- Share volume among different containers.
- Attach volume to containers
- On delete containers, volumes does not delete

Creating Volume from Dockerfile (Lab)

Create a Dockerfile

```
$ vi Dockerfile
……..
FROM Ubuntu
VOLUME [“/myvolume1”]

:wq
……..
```

To create an image from this Dockerfile

```
$ docker build –t <image_name> .

Eg:docker build –t volumeimage .
```

(-t = use for tag and .(dot) use for current directory)

Now create a container from this image and run

```
$ docker run –it --name <give_container_name> <image_name> /bin/bash

Eg: docker run –it --name volcontainer1 volumeimage /bin/bash
```

Now you are in container, do

```
$ ls
$ cd myvolume1

Create some files in myvolume1

$ touch file1 file2 file3

$ exit
```

Now, we share myvolume1 with another container

Container1 to container2

```
$ docker run –it <new-container-name> --privileged=true --volumes-from
<container-name> <image-name> /bin/bash

Eg: docker run –it volcontainer2 --privileged=true --volumes-from volcontainer1
ubuntu /bin/bash
```

After creating **volcontainer2, myvolume1** is visible whatever you do in one volume, you can see from other volume let's see

```
Now you are in container, do

$ ls

$ cd myvolume1

You can see files in myvolume1 which we created in volcontainer1

file1 file2 file3

$ exit
```

Creating volume by using Command

Create volume by using command

```
$ docker run -it --name volcontainer3 –v /myvolume2 ubuntu /bin/bash

Now you are in container, do

$ ls

Go to myvolume2 and create some files then exit

$ cd myvolume2/

$ touch filex filey filez

$ exit
```

Now Create another Container and share myvolume2 with new container

```
$ docker run –it <new-container-name> --privileged=true --volumes-from
<container-name> <image-name> /bin/bash

Eg: docker run –it volcontainer4 --privileged=true --volumes-from volcontainer3
ubuntu /bin/bash

Now you are in container, do

$ ls

Goto myvolume2 you can see files then exit

$ cd myvolume2/

filex filey filez

$ exit
```

You can also create files in volcontainer4 volume and see these files in volcontainer3 volume
Try it ☺

To remove the volume.

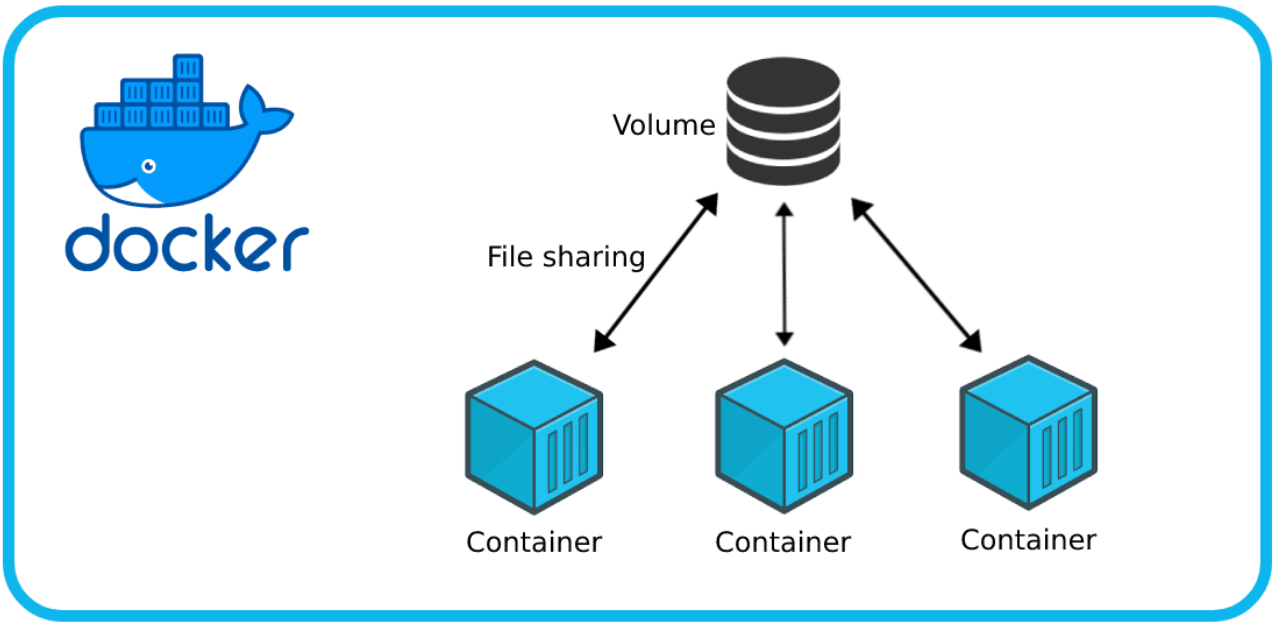
Stop the container and remove the volume.
Note that volume removal is a separate step.

```
$ docker container stop <container-name>
$ docker container rm <container-name>
$ docker volume rm <volume-name>
```

Now, share volume with Host

Host to container

A host is simply a device, like a computer or a server, that connects to the internet or a network.



First Verify files /home/ec2-user

```
$ docker run –it --name<host-container-name>-v /home/ec2-user:/container --
privileged=true /bin/bash

Eg: docker run -it --name hostcontaine -v /home/ec2-user:/container --
privileged=true /bin/bash
```

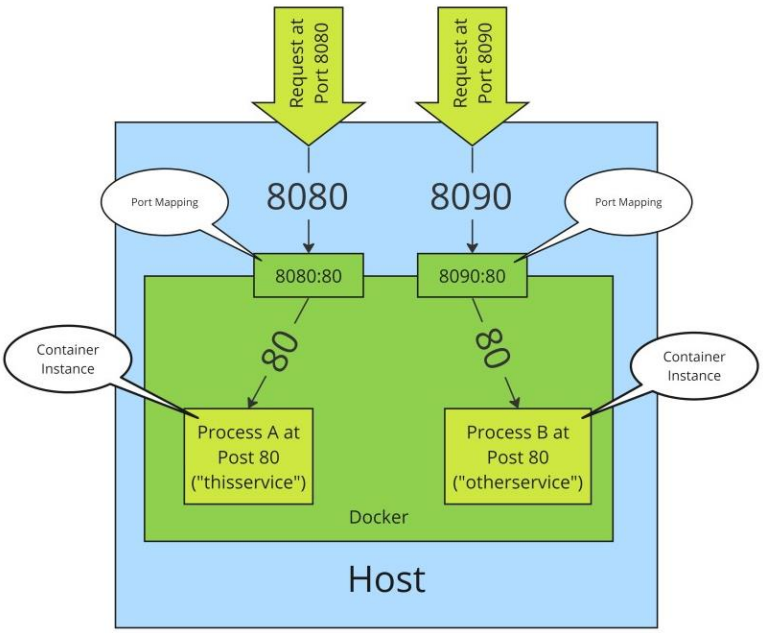
(/home/ec2-user = path in host , /container = container volume path , : = means map both together)

Some other commands

```
$ docker volume ls

$ docker volume create <volume-name> (create volume locally)

$ docker volume rm <volume-name> (To delete volume)

$ docker volume prune (remove all unused docker volumes)

$ docker volume inspect <volume-name> (To check details of volumes)

$ docker container inspect <container-name> (To check details of container)
```

Docker Port Expose

Port mapping is **used to access the services r inside a Docker container**. We open a host p give us access to a corresponding open port in the Docker container. Then all the requests tha made to the host port can be redirected into th Docker container



Creating a container and map port

```
$ docker run –td --name <conatinr-name> -p 80:80 ubuntu

( d use for daemon, p use for port or publish, -td use for create and run
container directly without entering it.)
```

To see which ports are mapped on container

```
$ docker port <container-name>
```

To enter in the container

```
$ docker exec –it <container-name> /bin/bash

(exec = execute) you are in the container
Try this

Because I create container using Ubuntu image in the container I hav to use
Ubuntu commands

$ apt-get update
$ apt-get install apache2
$ exit
```

Also you can try this

```
$ docker run –td --name jenkinsServer –p 8080:8080 jenkins

You can access Jenkins on port 8080
```

Difference between docker attach and docker exec

Docker exec creates a new process in the container's environment while docker attach just connect The standard Input/output of the main process inside the container to corresponding standard Input/output error of current terminal

Or

Docker exec is specifically for running new things in an already started container, be it a shell or some Other process

Difference between expose and publish

Basically, you have three (four) options:

1. Neither specify EXPOSE nor -p
2. Only specify EXPOSE
3. Specify EXPOSE and -p
4. Only specify -p which implicitly does EXPOSE

1. If you specify neither EXPOSE nor -p, the service in the container will only be accessible from inside the container itself.

2. If you EXPOSE a port, the service in the container is not accessible from outside Docker, but from inside other Docker containers. So this is good for inter-container communication.

3. If you EXPOSE and -p a port, the service in the container is accessible from anywhere, even outside Docker.

4. If you do -p, but do not EXPOSE, Docker does an implicit EXPOSE. This is because if a port is open to the public, it is automatically also open to other Docker containers. Hence -p includes EXPOSE. This is effectively same as.

What is Dockerhub?

Docker Hub is a container registry built for developers and open source contributors to find, use, and share their container images. With Hub, developers can host public repos that can be used for free, or private repos for teams and enterprises.

How To Push Docker Image In Dockerhub?
Lets Create a container first

```
$ docker run –it <image-name> /bin/bash
```

Create some files inside this container

```
$ touch file1 file2
$ exit
```

Now we create image from this container

```
$ docker commit <container-name> <image-name>
```

Now create account in hub.docker.com
Come to ec2 instance

```
$ docker login
Enter username and password
Done you are now connected with docker hub
```

Now we have to give tag to our image

```
$ docker tag <image-name> <dockerID/new-image-name>

Now Push Image
$ docker push <dockerID/new-image-name>
Wooohu you can  see this image in your docker hub account
```

To pull image from dockerhub

```
$ docker pull <dockerID/new-image-name>

To create container
$ docker run –it --name <image-name> <dokerID/new-image-name> /bin/bash
```

Some other important commands

```
$ docker stop $(docker ps –a -q)  Stop all running containers

$ docker rm $(docker ps –a -q)  Delete all stopped containers

$ docker rmi $(docker images -q)  Delete all images
```