



DOCKER

Step by step guide with examples



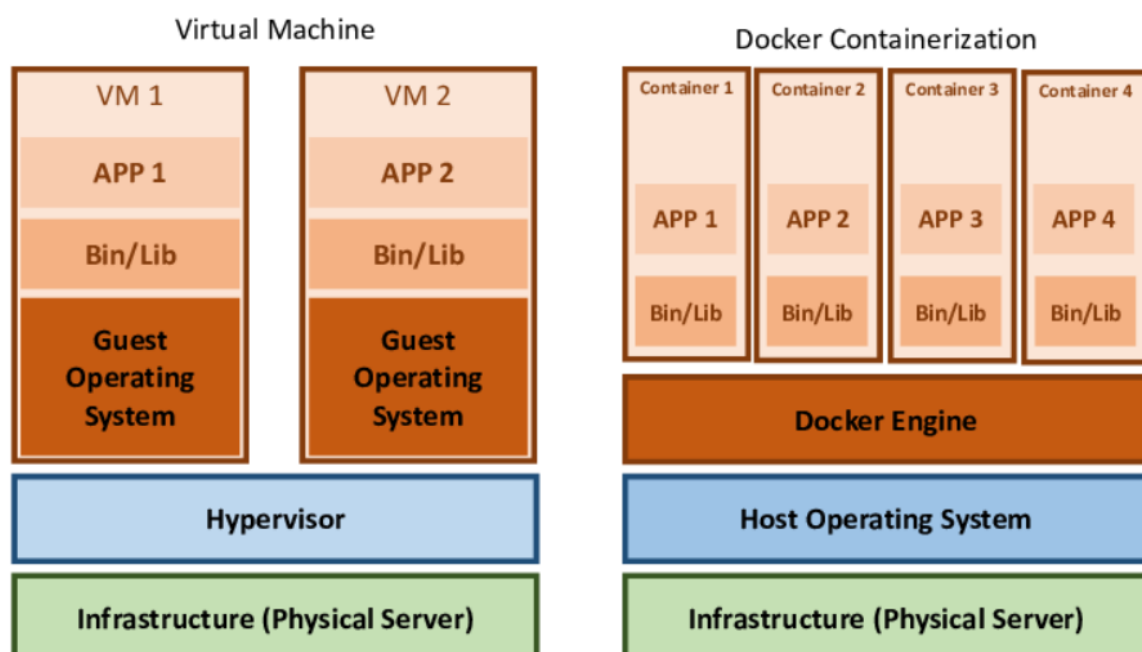
MUHAMMAD TANVEER ASHRAF

Contents

Docker	2
Advantages of Docker	3
Disadvantages of Docker	3
Components of Docker – Ecosystem	3
Docker Daemon (Docker Engine)	3
Docker Client	3
Docker Host	4
Docker Hub/Registry	4
Docker images	4
Docker Container	4
Basic Commands in Docker	4
Container Management	4
Image Management	5
Other Useful commands	7
Docker File	8
Create Image	9
Create Docker file	9
Docker Volume	11
Benefits of volume	12
Creating volume – step by step	12
Share volume between host and container	14
Docker port expose	14
Docker attach vs Docker exec	17
Expose and Publish	17

Docker

- Docker is an open-source centralized platform design to create, deploy and run applications
- Docker uses container on the **host operating system** to run application. It allows applications to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual operating system
- We can install Docker on any operating system, **but Docker engine runs natively on Linux distribution**
- [Docker is written in Go Language](#)
- Docker is a tool that performs **OS level virtualization, also known as containerization**
Because it does not take resources from host hardware, instead operating system.
 - For example, we have a hardware of 16GB RAM and we have 2 containers running on Docker. Let say container 1 needs 10GB RAM to process something, then it will request the RHEL which runs on OS to provide 10GB RMA and once its processing completed, then it releases the RAM and the host again have full capacity available.
- Before docker, many users face the problem that a particular code is running in developer's machine, but not in server. This sometimes happens because of some package's versions mismatch. Docker bundle everything together in a container, and ship it as a whole running application
- Docker is a set of platforms as a service that uses OS level virtualization whereas VMware uses hardware level virtualization



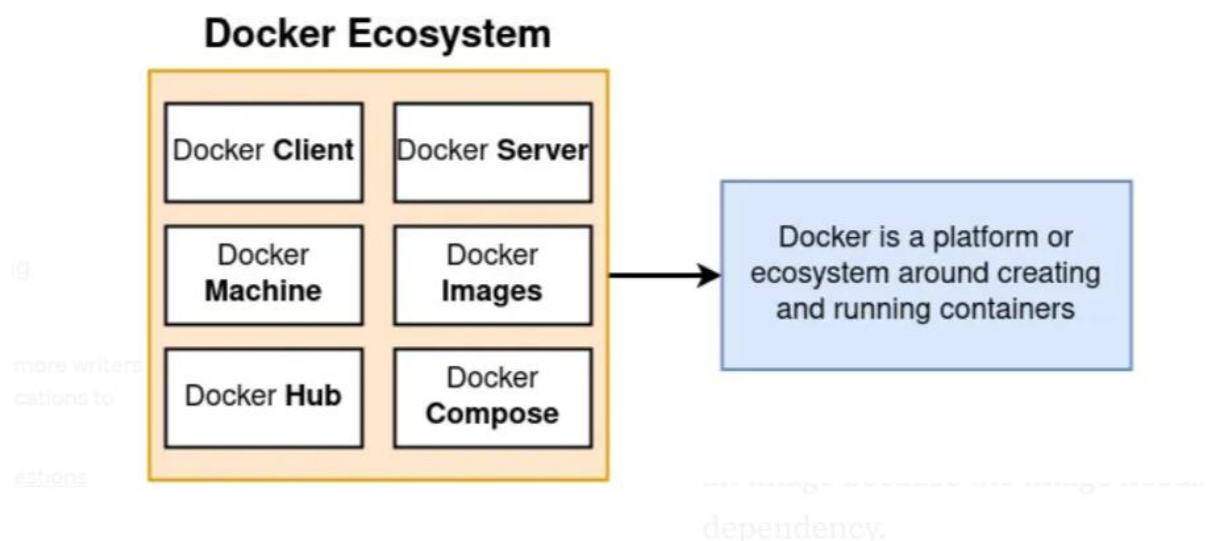
Advantages of Docker

- No pre-allocation of RAM
- Continuous 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 (use less resources to run applications)
- It can run on physical hardware virtual/hardware or on cloud
- You can re-use the image
- It took very less time to create image

Disadvantages of Docker

- Docker does not support cross-platform. If an application is designed to run in a docker container on windows, can't run on Linux or vice-versa
- Docker is not a good solution for application that requires rich GUI
- Difficult to manage large amount of containers
- Docker is suitable when the development OS and Testing OS are same

Components of Docker – Ecosystem



Docker Daemon (Docker Engine)

- Runs on the host operating system
- It is responsible for running containers to manage docker services
- It can also communicate with other daemons

Docker Client

- Client is responsible to take files to server
- Docker users interact with docker daemon through docker client (CLI)

- Docker client uses commands and Rest API to communicate with the docker daemon
- When a client runs any server command on the docker client terminal, the client terminal send these docker commands to the docker daemon
- It is possible for docker client to communicate with more than one daemon

Docker Host

- The physical hardware on which docker engine is running
- Provides resources to the containers
- Docker host is used to provide an environment to execute and run applications. It contains the docker daemon, images, containers, networks and storages

Docker Hub/Registry

- Docker registry manages and stores the docker images
- There are two types of registries
 - Public Registry -> also called as docker hub
 - Private Registry -> used to share images within the enterprise (paid version)

Docker images

- Docker images are the read only binary templates used to create docker containers
- Image can be created from
 - Pull from Docker hub
 - Docker file
 - Existing docker containers

Docker Container

- Container holds the entire package that is needed to run the application
- When we run images on docker engine, they become container

Basic Commands in Docker

Container Management

- `docker run <image>`.
 - Creates and runs a new container from a specified image.
 - For example, you need to create and run a container with specified name, then you have to use “`docker run -it -- name TestName Jenkins/bin/bash`”. In this example, you are pulling the Jenkins image with TestName in your local machine. This will open terminal where the command will run in container
 - In below example, **1d6c34784efd** is the image Id

```

root@1d6c34784efd: /
[root@ip-172-31-19-53 ec2-user]# docker run -it ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
b71466b94f26: Pull complete
Digest: sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfeld2c0322aaf20f9
Status: Downloaded newer image for ubuntu:latest
root@1d6c34784efd: /#

```

- `docker ps -a`
 - To see all containers

```

root@ip-172-31-19-53/home/ec2-user
[root@ip-172-31-19-53 ec2-user]# docker ps -a

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
94f5b6f37cda	ubuntu	"/bin/bash"	About a minute ago	Up 45 seconds		tanveerashraf
fba55a523ef4	eclipse/centos	"/bin/bash"	4 minutes ago	Exited (127) 4 minutes ago		heuristic_shockley
5d02647b9074	eclipse/centos	"/bin/bash"	5 minutes ago	Created		quirky_lederberg
92039427ebee	eclipse/centos	"/bin/bash"	8 minutes ago	Exited (127) 5 minutes ago		eager_gagarin
ef45de7e95f2	jenkins/jenkins	"/usr/bin/tini -- /u..."	12 minutes ago	Exited (127) 12 minutes ago		naughty_antonelli
170b2dd622c4	ubuntu	"/bin/bash"	39 minutes ago	Exited (127) 39 minutes ago		adoring_napier
1d6c34784efd	ubuntu	"/bin/bash"	58 minutes ago	Exited (127) 57 minutes ago		determined_sutherland

```

[root@ip-172-31-19-53 ec2-user]#

```

- `docker ps`
 - To see only running containers. PS means, process status
- `docker start <container>`
 - To start container
- `docker attach <container>`
 - To go inside container
- `docker stop <container>`
 - To stop a container
- `docker rm <container>`
 - To delete a container
- `docker exec -it <container> <command>`
 - Executes a command inside a running container. The -it flags provide an interactive terminal.
- `docker logs <container>`
 - Fetches the logs of a container.

Image Management

- `docker images`
 - Lists all locally available Docker images.

```
root@ip-172-31-19-53:/home/ec2-user

[root@ip-172-31-19-53 ec2-user]# docker images
REPOSITORY          TAG             IMAGE ID         CREATED          SIZE
jenkins/jenkins     latest         ac09998055cc    4 days ago      480MB
ubuntu              latest         e0f16e6366fe    4 weeks ago     78.1MB
[root@ip-172-31-19-53 ec2-user]#
```

- docker search <image>
 - To find out an image in docker hub
 - For example, “docker search Jenkins” to search all Jenkins images

```
root@ip-172-31-19-53:/home/ec2-user

[root@ip-172-31-19-53 ec2-user]# docker search jenkins
NAME                                DESCRIPTION                                STARS    OFFICIAL
jenkins/jenkins                    The leading open source automation server  4175
jenkins                            DEPRECATED; use "jenkins/jenkins:lts" instead 5706    [OK]
jenkins/inbound-agent              This is an image for Jenkins agents using TC... 146
jenkins/ssh-agent                  Docker image for Jenkins agents connected ov... 66
jenkins/jnlp-agent-maven            A JNLP-based agent with Maven 3 built in      10
jenkins/slave                      base image for a Jenkins Agent, which includ... 53
jenkins/agent                      This is a base image, which provides the Jen... 78
jenkins/jnlp-agent-ruby             1
jenkins/jnlp-agent-docker           12
jenkins/ath                        Jenkins Acceptance Test Harness              1
jenkins/jnlp-agent-node             1
jenkins/jnlp-agent-python           4
jenkins/jenkins-experimental        Experimental images of Jenkins. These images... 3
jenkins/pct                         Plugin Compat Tester - no longer published a... 5
jenkins/core-pr-tester              Docker image for testing pull-requests sent ... 2
jenkins/jnlp-agent-alpine           2
jenkins/jnlp-agent-coresdk          2
jenkins/jenkinsfile-runner          Jenkinsfile Runner packages                 3
jenkins/evergreen                  An automatically self-updating Jenkins distr... 5
jenkins/jnlp-agent-jdk11            2
jenkins/jnlp-agent-terraform        7
jenkins/remoting-kafka-agent        Remoting Kafka Agent                       1
jenkins/jnlp-agent-python3          3
jenkins/core-changelog-generator    Tool for generating Jenkins core changelogs  1
bitnami/jenkins                    Bitnami container image for Jenkins         78
[root@ip-172-31-19-53 ec2-user]#
```

- docker pull <image>
 - Pulls an image from a Docker registry (e.g., Docker Hub) to local machine
 - For example, “docker pull Jenkins” to pull Jenkins image to your local machine
- docker rmi <image>
 - Removes one or more images.
- docker push <image>
 - Pushes an image to a Docker registry.

Other Useful commands

- service docker status
 - To check service is running or not
- docker system prune
 - Removes unused Docker data (containers, images, networks, volumes).'
- service docker start
 - To Start docker service
- docker commit <newContainer> <image>
 - To create an image from existing container

```
root@2c0801f6bde7: /tmp
[root@ip-172-31-19-53 ec2-user]# docker commit TanveerContainer updateimage
sha256:5272ed184116c4a1c71c148a2e6fe1e567a46de12f2cf5c795e5955602bf21cd
[root@ip-172-31-19-53 ec2-user]# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
updateimage         latest         5272ed184116   7 seconds ago  78.1MB
jenkins/jenkins     latest        ac09998055cc   5 days ago     480MB
ubuntu              latest        e0f16e6366fe   4 weeks ago    78.1MB
eclipse/centos      latest        769b57b7d29e   8 years ago    267MB
[root@ip-172-31-19-53 ec2-user]# docker run -it --name NewTanvContainer updateimage /bin/bash
root@2c0801f6bde7:/# ls
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib  media  opt  root  sbin  sys  usr
root@2c0801f6bde7:/# cd tmp/
root@2c0801f6bde7:/tmp# ls
TanveerFile
root@2c0801f6bde7:/tmp#
```

- docker diff <container>
 - The docker diff command is used to inspect changes made to the filesystem of a running Docker container since its creation or last restart. It provides a list of files and directories that have been added, deleted, or modified within the container's filesystem layer on top of its base image.
 - The output of docker diff indicates the type of change for each file or directory:
 - A: Indicates that a file or directory was added.
 - D: Indicates that a file or directory was deleted.
 - C: Indicates that a file or directory was changed (modified).
- docker container inspect <containerName>
 - to inspect details of a container


```
root@ip-172-31-19-53:/home/ec2-user

[root@ip-172-31-19-53 ec2-user]# docker run --name TanveerContainer -it ubuntu /
bin/bash
root@1f5b9d739514:/# ls
bin    dev    home  lib64  mnt    proc   run    srv    tmp    var
boot  etc    lib   media  opt    root   sbin   sys    usr
root@1f5b9d739514:/# cd temp/
bash: cd: temp/: No such file or directory
root@1f5b9d739514:/# cd temp/
bash: cd: temp/: No such file or directory
root@1f5b9d739514:/# cd tmp/
root@1f5b9d739514:/tmp# touch TanveerFile
root@1f5b9d739514:/tmp# ls
TanveerFile
root@1f5b9d739514:/tmp# exit
exit
[root@ip-172-31-19-53 ec2-user]# docker diff TanveerContainer
C /root
A /root/.bash_history
C /tmp
A /tmp/TanveerFile
[root@ip-172-31-19-53 ec2-user]#
```

Docker File

- Docker file is basically a text file contains some set of instructions. With the help of this, we automate the docker image creation. The name of the file must be Dockerfile with capital D.
- The instructions must be provided in capital letter which includes
 - FROM -> For base image. This command must be on top of the docker file
 - RUN -> To execute commands we have in docker file. It will create a layer in image (because container architecture is in layered form)
 - MAINTAINER -> author/owner/description
 - COPY -> Copy files **from local system** (docker VM). We need to provide source, destination. (we can't download file from internet and any remote repo)
 - ADD -> Similar to copy but, it provides a feature to download file from internet. It also extracts the file at docker image side because the downloaded file will be in zip format
 - EXPOSE -> To expose ports such as port 8080 for tomcat, port 80 for nginx etc
 - WORKDIR -> To set working directory for a container
 - CMD -> Execute commands but during container creation
 - ENTRYPOINT -> Similar to CMD, but has the higher priority over CMD. First command will be executed by ENTRYPOINT only
 - ENV -> Environment variables. Docker ENV instruction is used at **Runtime**.
 - ARG -> Defines a **build-time** variable that can be passed to the Docker build process. This default value can also be overridden using a simple option with the Docker build command

- Syntax: ARG <name>[=<default value>]. <name> is the name of the variable. [=<default value>] is optional and provides a default value if no value is provided during the build.
- Example: ARG IMAGE_VERSION=latest
- <https://www.geeksforgeeks.org/devops/docker-arg-instruction/>

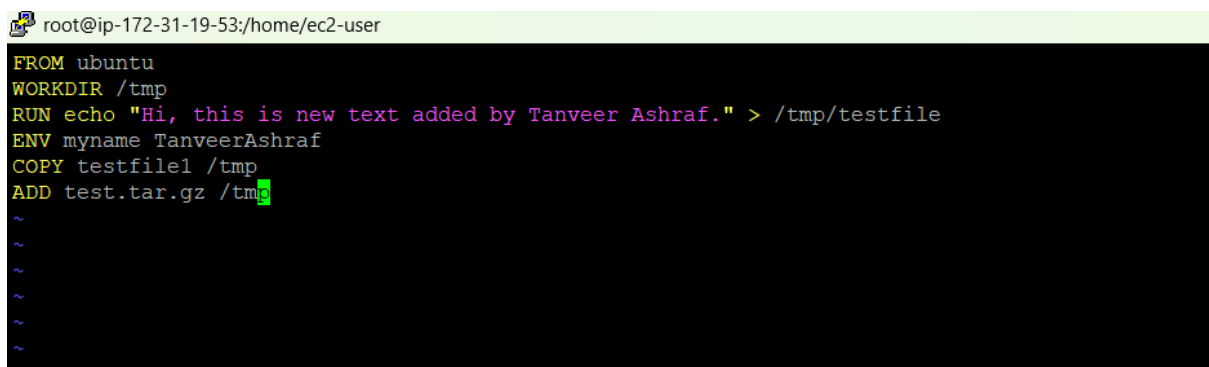
Create Image

There are 3 ways to create image

1. Pull an image from Docker Hub
2. Create an image from a container
 - a. For example, you pull a ubuntu image from docker hub and created a container. You have installed many software required for your application in that container. Now, you want to share the same container to another person so that he does not need to do the same effort again. In this scenario, you create an image from your container and share that to the other person
3. Create an image from Docker file

Create Docker file

1. Create a file named as Dockerfile
 - a. For example, in Linux, the command **vi Dockerfile** used to create a file named as Dockerfile
2. Add instructions in docker file
3. Build docker file to create image
 - a. docker build -t <image> . The "." Means to create image from current docker file
4. Run image to create container
 - a. docker run -it --name <container> <image> /bin/bash



```

root@ip-172-31-19-53:/home/ec2-user
FROM ubuntu
WORKDIR /tmp
RUN echo "Hi, this is new text added by Tanveer Ashraf." > /tmp/testfile
ENV myname TanveerAshraf
COPY testfile1 /tmp
ADD test.tar.gz /tmp
~
~
~
~
~
~

```

In above docker file, we are saying

1. build a new docker image based on ubuntu
2. the working directory will be temp
3. add a string in testfile inside tmp directory

4. the environment variable is myname
5. copy the testfile1 from temp directory
6. download a file and unzip

```

root@ip-172-31-19-53:/home/ec2-user
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile  test.tar.gz  testfile1
[root@ip-172-31-19-53 ec2-user]# vi Dockerfile
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile  test.tar.gz  testfile1
[root@ip-172-31-19-53 ec2-user]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
test          latest    3109ed3a3e0a   11 minutes ago  78.1MB
[root@ip-172-31-19-53 ec2-user]# docker build -t newimage .
[+] Building 0.7s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 262B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/ubuntu:latest@sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfeld2c0322aaf20f9
=> resolve docker.io/library/ubuntu:latest@sha256:7c06e91f61fa88c08cc74f7e1b7c69ae24910d745357e0dfeld2c0322aaf20f9
=> [internal] load build context
=> => transferring context: 296B
=> CACHED [2/5] WORKDIR /tmp
=> [3/5] RUN echo "Hi, this is new text added by Tanveer Ashraf." > /tmp/testfile
=> [4/5] COPY testfile1 /tmp
=> [5/5] ADD test.tar.gz /tmp
=> exporting to image
=> => exporting layers
=> => writing image sha256:bb6a9186a7c8deb6b6b518cc13f793edba8459091ea34097ded17ed0ec2ed094
=> => naming to docker.io/library/newimage
[root@ip-172-31-19-53 ec2-user]#

```

As seen above, a new docker image with name “newimage” is successfully created from the Dockerfile. Now, we can create a container from that image as below. To print the environment variable we can echo with \$ sign

```

root@ip-172-31-19-53:/home/ec2-user
[root@ip-172-31-19-53 ec2-user]# docker run -it --name NewContainer newimage /bin/bash
root@4466de016620:/tmp# ls
test  testfile  testfile1
root@4466de016620:/tmp# cat testfile
Hi, this is new text added by Tanveer Ashraf.
root@4466de016620:/tmp# echo $myname
TanveerAshraf
root@4466de016620:/tmp# exit
exit
[root@ip-172-31-19-53 ec2-user]# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS          NAMES
4466de016620   newimage  "/bin/bash"             3 minutes ago  Exited (0)   4 seconds ago           NewContainer
4b0159a33ba7   test     "/bin/bash"             17 minutes ago  Exited (0)   17 minutes ago          TanveerContainer
[root@ip-172-31-19-53 ec2-user]#

```

Note: To create testfile, the command is touch “filename”

```

root@ip-172-31-19-53:/home/ec2-user
[root@ip-172-31-19-53 ec2-user]# vi Dockerfile
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile
[root@ip-172-31-19-53 ec2-user]# touh testfile
bash: touh: command not found
[root@ip-172-31-19-53 ec2-user]# touch testfile
[root@ip-172-31-19-53 ec2-user]#

```

To convert a file to zip file

```
root@ip-172-31-19-53:/home/ec2-user
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile test testfile1
[root@ip-172-31-19-53 ec2-user]# tar -cvf test.tar test
test
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile test test.tar testfile1
[root@ip-172-31-19-53 ec2-user]# gzip test.tar
[root@ip-172-31-19-53 ec2-user]# ls
Dockerfile test test.tar.gz testfile1
[root@ip-172-31-19-53 ec2-user]#
```

Docker Volume

- Volume is simply a directory inside our container
- First, we have to declare a directory as volume, and then we share the volume
- Even if we stop the container, the volume still can be accessible
- 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 containers
 - When you share the volume with other containers, then whoever makes a change in it, will be visible in all containers using that shared volume
- Volume will not be included when you update an image
 - For example, you have a container "A" with some files and a volume "V". You create an image from this container and from that image, you create another container "B". In container B you can see the volume as directory only, not as volume. This means when you add any file from B in this directory, will not be reflected in container A and vice versa.
- You can map volume in two ways
 - Container to container and vice versa
 - Host to container and vice versa
- Commands used for volume are
 - docker volume ls -> list all volumes
 - docker volume create <volumename> -> create a new volume
 - docker volume rm <volumename> remove volume
 - docker volume prune -> remove all unused docker volumes
 - docker volume inspect <volumeName> -> to inspect details of a volume

Benefits of volume

- Decoupling container from storage
- Share volume among different containers
- Attach volume to containers
- On deleting container, volume does not delete

Creating volume – step by step

- Create a docker file and write some commands for example
 - FROM ubuntu
 - VOLUME ["volumeName"]
- Create image from this docker file
 - docker build -t <image> .
 - This will create image with provided name using instructions given in the docker file. -t means tag and is used to give image name
- Create a container from this image and run
 - docker run -it --name <container> <image> /bin/bash
- Now doing "ls" you can see the volume. In below example, the volume name is "myvolume"

```
root@b5c72a536608: /myvolume
[root@ip-172-31-19-53 ec2-user]# docker run -it --name container1 myimage /bin/b
ash
root@b5c72a536608:/# ls
bin  dev  home  lib64  mnt      opt  root  sbin  sys  usr
boot  etc  lib   media  myvolume  proc  run   srv   tmp  var
root@b5c72a536608:/# cd myvolume
root@b5c72a536608:/myvolume# touch filex filey filez
root@b5c72a536608:/myvolume# ls
filex  filey  filez
root@b5c72a536608:/myvolume#
```

- Share this volume with other containers
 - docker run -it --name <newContainer> --privileged=true --volumes-from <oldContainer> <image> /bin/bash
 - privileged=true means to give rights to the new container to add/update the volume content
- Now after creating new container, the volume will be visible in it. If you add a new file in this volume, it will be visible to the old container as well
 - touch /<volume>/newfile -> to create a newfile in the volume
 - docker start <oldContainer>
 - docker attach <oldContainer> -> to go inside the old container
 - ls <volume> -> will list the newfile created in <newContainer>

```
[root@ip-172-31-19-53 ec2-user]# docker start container2
container2
[root@ip-172-31-19-53 ec2-user]# docker attach container2
root@e06d773b62aa:/# ls
bin  dev  home  lib64  mnt      opt   root  sbin  sys  usr
boot  etc  lib   media  myvolume  proc  run   srv   tmp   var
root@e06d773b62aa:/# cd myvolume
root@e06d773b62aa:/myvolume# ls
filex  filey  filez
root@e06d773b62aa:/myvolume# touch fileTanveer
root@e06d773b62aa:/myvolume# exit
exit
[root@ip-172-31-19-53 ec2-user]# docker start container1
container1
[root@ip-172-31-19-53 ec2-user]# docker attach container1
root@b5c72a536608:/# ls
bin  dev  home  lib64  mnt      opt   root  sbin  sys  usr
boot  etc  lib   media  myvolume  proc  run   srv   tmp   var
root@b5c72a536608:/# cd myvolume
root@b5c72a536608:/myvolume# ls
fileTanveer  filex  filey  filez
root@b5c72a536608:/myvolume#
```

- To create container using command and share the volume, you have to create volume during creation of container. Add command -v /volumename

```
root@ec7fc2154dbc: /
[root@ip-172-31-19-53 ec2-user]# docker run -it --name container3 -v /volume2 ubuntu /bin/bash
root@ec7fc2154dbc:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volume2
root@ec7fc2154dbc:/#
```

- In below example, we have a container 3, in which we created a volume as “volume2”. This volume contains 3 files vol1, vol2 and vol3. Create a new container “container4” with sharing the volume and add a new file in volume2 inside container 4. This will be visible in container3 as well and vice versa.

```
root@ec7fc2154dbc: /volume2
[root@ip-172-31-19-53 ec2-user]# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS   NAMES
ec7fc2154dbc   ubuntu   "/bin/bash"             About an hour ago   Exited (0) 22 seconds ago           container3
e06d773b62aa   ubuntu   "/bin/bash"             2 hours ago       Exited (0) 2 hours ago           container2
d9c03ca5e867   ubuntu   "/bin/bash"             2 hours ago       Exited (0) 2 hours ago           my-ubuntu-container
b5c72a536608   myimage   "/bin/bash"             2 hours ago       Exited (0) About an hour ago           container1
[root@ip-172-31-19-53 ec2-user]# docker run -it --name container4 --privileged=true --volumes-from container3 ubuntu /bin/bash
root@5c492aff88a1:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volume2
root@5c492aff88a1:/# cd volume2
root@5c492aff88a1:/volume2# ls
vol1  vol2  vol3
root@5c492aff88a1:/volume2# touch newFileContainer4
root@5c492aff88a1:/volume2# ls
newFileContainer4  vol1  vol2  vol3
root@5c492aff88a1:/volume2# exit
exit
[root@ip-172-31-19-53 ec2-user]# docker start container3
container3
[root@ip-172-31-19-53 ec2-user]# docker attach container3
root@ec7fc2154dbc:/# cd volume
bash: cd: volume: No such file or directory
root@ec7fc2154dbc:/# cd volume2
root@ec7fc2154dbc:/volume2# ls
newFileContainer4  vol1  vol2  vol3
root@ec7fc2154dbc:/volume2#
```

Share volume between host and container

We can also share directory/volume between host and container. For example, we have an ec2-user and we want to create a container with name “hostContainer”. We want to create a directory with name “newdirectory” which we want to share between the host and hostContainer. In this case, we have to give path for host directory as volume as below

- `docker run -it --name hostContainer -v /home/ec2-user:/newDirectory --privileged=true ubuntu /bin/bash`

```
root@57feb5e767e4: /tanveerDirectory
[ec2-user]# cd ..
[ec2-user]# ls
Dockerfile
[ec2-user]# touch file1 file2
[ec2-user]# ls
Dockerfile file1 file2
[ec2-user]# docker run -it --name hostContainer -v /home/ec2-user:/tanveerDirectory --privileged=true ubuntu /bin/bash
root@57feb5e767e4:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tanveerDirectory tmp usr var
root@57feb5e767e4:/# cd tanveerDirectory/
root@57feb5e767e4:/tanveerDirectory# ls
Dockerfile file1 file2
root@57feb5e767e4:/tanveerDirectory#
```

- we can also verify by creating more files inside the container and check the same files existence from host as below

```
[ec2-user]# docker start hostContainer
hostContainer
[ec2-user]# ls
Dockerfile file1 file2
[ec2-user]# docker attach hostContainer
root@57feb5e767e4:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tanveerDirectory tmp usr var
root@57feb5e767e4:/# cd tanveerDirectory
root@57feb5e767e4:/tanveerDirectory# touch tiru1 tiru2 tiru3
root@57feb5e767e4:/tanveerDirectory# ls
Dockerfile file1 file2 tiru1 tiru2 tiru3
root@57feb5e767e4:/tanveerDirectory# exit
exit
[ec2-user]# ls
Dockerfile file1 file2 tiru1 tiru2 tiru3
[ec2-user]#
```

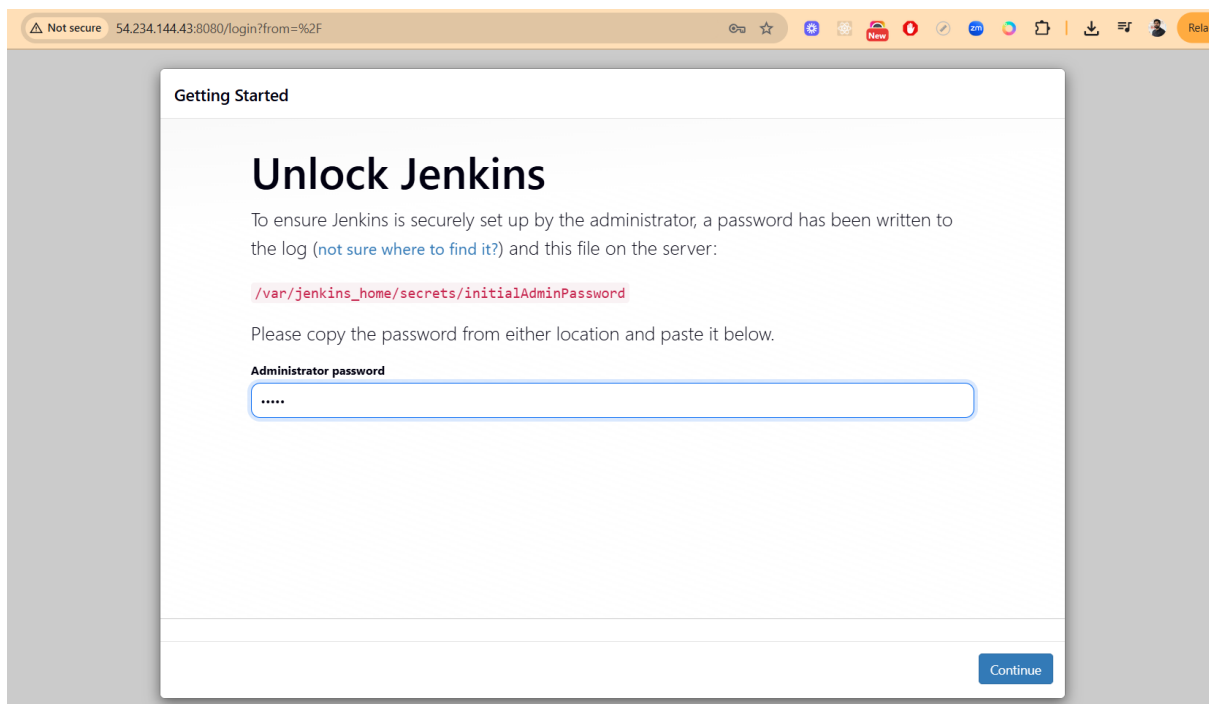
Docker port expose

- Login to Aws account and one Linux instance, and login as ec2-user
- Run command `sudo su` to get admin privileges
- To install docker, `yum install docker -y`
- To start docker, `service docker start`
- Run the command to expose port 80 of the container
 - **`docker run -td --name techserver -p 80:80 ubuntu`**
 - **-i** (interactive)
 - i. Keeps STDIN open even if not attached.
 - ii. Useful if you want to interact with the container (e.g., run a shell).
 - **-t** (tty)
 - i. Allocates a pseudo-TTY (terminal).
 - ii. Gives you a shell-like experience (colors, cursor movement, etc.).

- **-d (detached)**
 - i. Runs the container in the background.
 - ii. You don't stay "attached" to the container's logs/terminal.
 - iii. Example, `docker run -d -p 80:80 nginx`. This means Nginx runs in background; you can access it via port 80. The first 80 is host port number, and the second is container port number
 - iv. Similarly, for example, you want to run Jenkins on port 8080, use below command

```
root@ip-172-31-22-100:/home/ec2-user
[root@ip-172-31-22-100 ec2-user]# docker run -td --name "myJenkins" -p 8080:8080 jenkins/jenkins
Unable to find image 'jenkins/jenkins:latest' locally
latest: Pulling from jenkins/jenkins
f014853ae203: Pull complete
cb50b8a39448: Pull complete
6456b752e285: Pull complete
6bb1fadd55ef: Pull complete
d0dc2fddd89c: Pull complete
603ef113f8d9: Pull complete
3b74159c44a8: Pull complete
d69085ea972e: Pull complete
4c7ba2216ab5: Pull complete
224d9f0f6697: Pull complete
eae8978797bd: Pull complete
333714e93eb7: Pull complete
Digest: sha256:7fc10c3969dd72c81531ed815cbee0c3e7b789603cd7eb6c8de8e609ae572f1e
Status: Downloaded newer image for jenkins/jenkins:latest
96a15a272b33967be2f737e4978a900d09eb31f7d1728afd16a7d8b59d1de924
[root@ip-172-31-22-100 ec2-user]#
```

Now if you browse from the browser, just copy your public IP of the instance and add port 8080 will show you Jenkins, and port 80 will show you the simple html page



My name is Tanveer Ashraf

- docker ps
- docker port <container>
 - To check which ports are exposed. Example, docker port techserver to check ports exposed in techserver container
- docker exec -it techserver /bin/bash
 - Exec is similar to attach and the purpose is same as attach i.e. to get inside the container, but exec starts a new process
- apt-get update
 - As our container is ubuntu, hence why this command is used to update
- apt-get install apache2 -y
 - In Linux, we run the command httpd to install apache2. In ubuntu it will be apt-get install apache2
- Create a web page in default directory
 - cd /var/www/html
 - echo "Hello, my name is Tanveer Ashraf" > index.html
 - i. This will add the above text in index.html file

```
root@ca3365f7b1c6: /var/www/html
root@ca3365f7b1c6:/# cd /var/www/html
root@ca3365f7b1c6:/var/www/html# echo "My name is Tanveer Ashraf" > index.html
root@ca3365f7b1c6:/var/www/html# service apache2 restart
* Restarting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'ServerName' directive globally to suppress this message
[ OK ]
root@ca3365f7b1c6:/var/www/html#
```

- service apache2 start
 - To start the apache2 server
- Now you can copy the host public IP address and check in google, the above index.html page should be visible
- Similarly, if you want to access Jenkins
 - docker run -td --name myJenkins -p 8080:8080 jenkins

- i. with -td, it will run the container but will not take you in the container
like using -it, it runs the container and also takes us inside it.
- o both host and container ports are 8080 and container name is myJenkins
- o This also needs to allow port 8080 in ec2 instance security group

Docker attach vs Docker exec

Docker exec creates a new process in the container's environment while docker attach just Attaches your terminal's standard input, output, and error streams to the primary process (PID 1) of a running container.

Feature	docker exec	docker attach
Action	Runs a new process	Attaches to the primary process (PID 1)
Purpose	Debugging, running commands, interactive shell	Viewing output, direct interaction with main app
Impact on Container	Exiting does not stop container	Exiting primary process usually stops container (unless detached cleanly)

Expose and Publish

Basically, we have three options

1. Neither specify **expose** nor **-p** (for publish, we use -p)
 - a. In this case the service in the container will only be accessible from inside the container
2. Only specify **expose**
 - a. The service in the container is not accessible from **outside docker containers**, so this is good for inter-container communication
3. Specify both **expose** and **-p**
 - a. The service in the container is accessible from anywhere, even outside docker
 - b. If you do **-p** and do not expose, docker does an **implicit expose**. This is because if a port is open to the public, it is automatically open to other docker containers. Hence -p includes expose