

Docker

Step by step guide with examples



Muhammad Tanveer Ashraf

Contents

[Docker 2](#_Toc207836088)

[Advantages of Docker 3](#_Toc207836089)

[Disadvantages of Docker 3](#_Toc207836090)

[Components of Docker – Ecosystem 3](#_Toc207836091)

[Docker Daemon (Docker Engine) 3](#_Toc207836092)

[Docker Client 3](#_Toc207836093)

[Docker Host 4](#_Toc207836094)

[Docker Hub/Registry 4](#_Toc207836095)

[Docker images 4](#_Toc207836096)

[Docker Container 4](#_Toc207836097)

[Basic Commands in Docker 4](#_Toc207836098)

[Container Management 4](#_Toc207836099)

[Image Management 5](#_Toc207836100)

[Other Useful commands 7](#_Toc207836101)

[Docker File 8](#_Toc207836102)

[Create Image 9](#_Toc207836103)

[Create Docker file 9](#_Toc207836104)

[Docker Volume 11](#_Toc207836105)

[Benefits of volume 12](#_Toc207836106)

[Creating volume – step by step 12](#_Toc207836107)

[Share volume between host and container 14](#_Toc207836108)

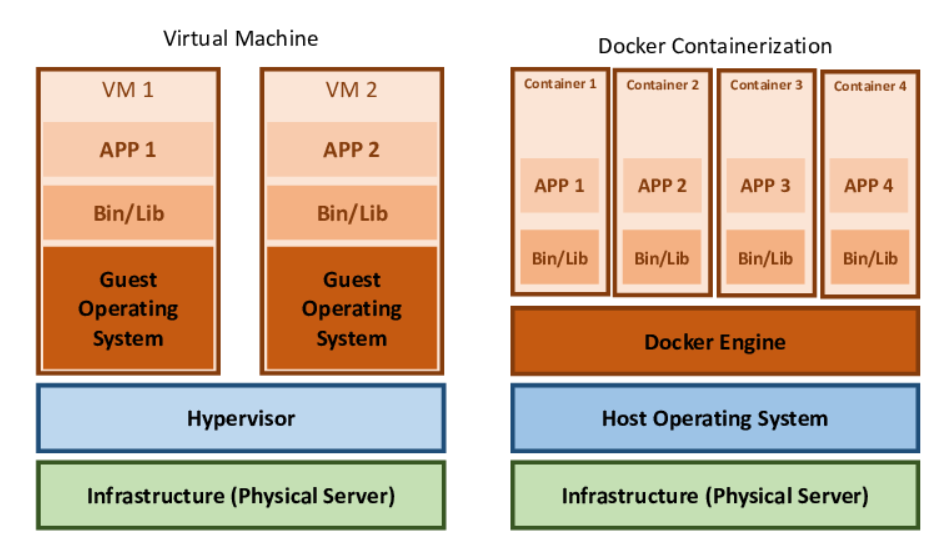
[Docker port expose 14](#_Toc207836109)

[Docker attach vs Docker exec 17](#_Toc207836110)

[Expose and Publish 17](#_Toc207836111)

# 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



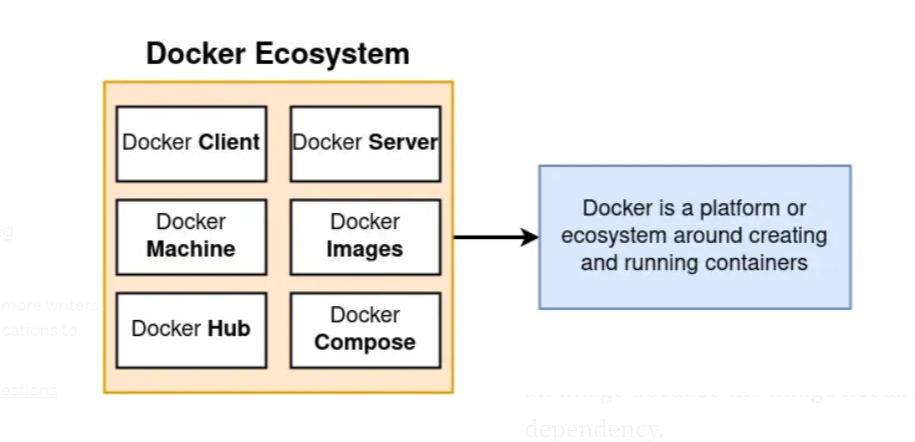
## Advantages of Docker

* 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

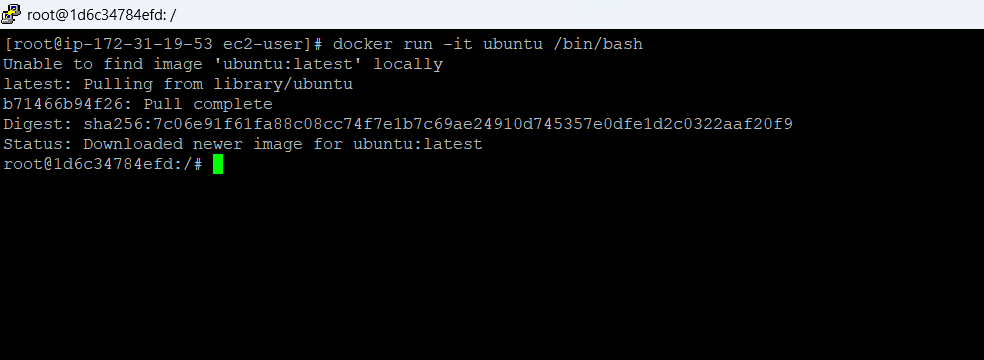
### Docker Container

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

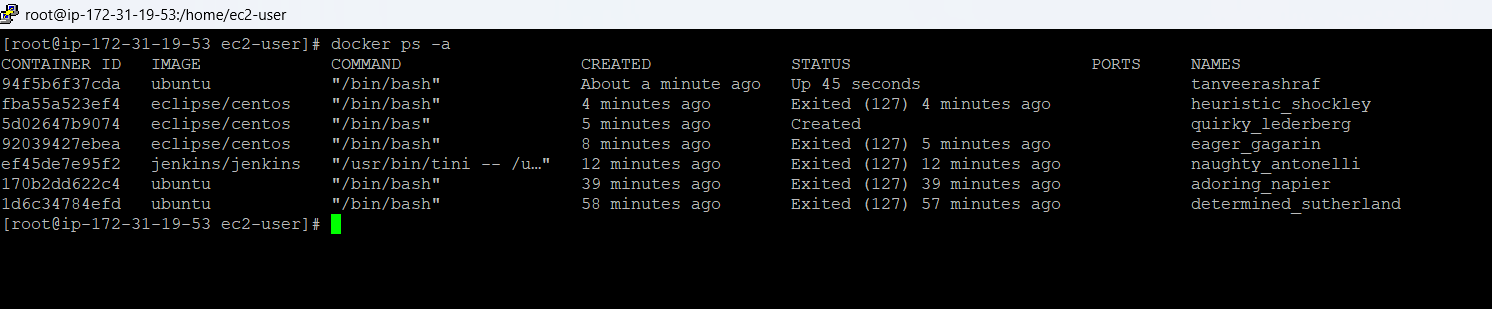
## Basic Commands in Docker

### 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



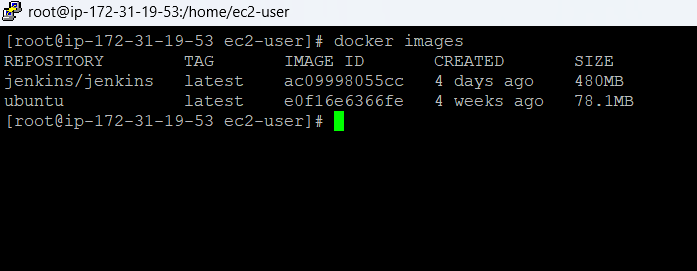
* docker ps -a
  + To see all containers



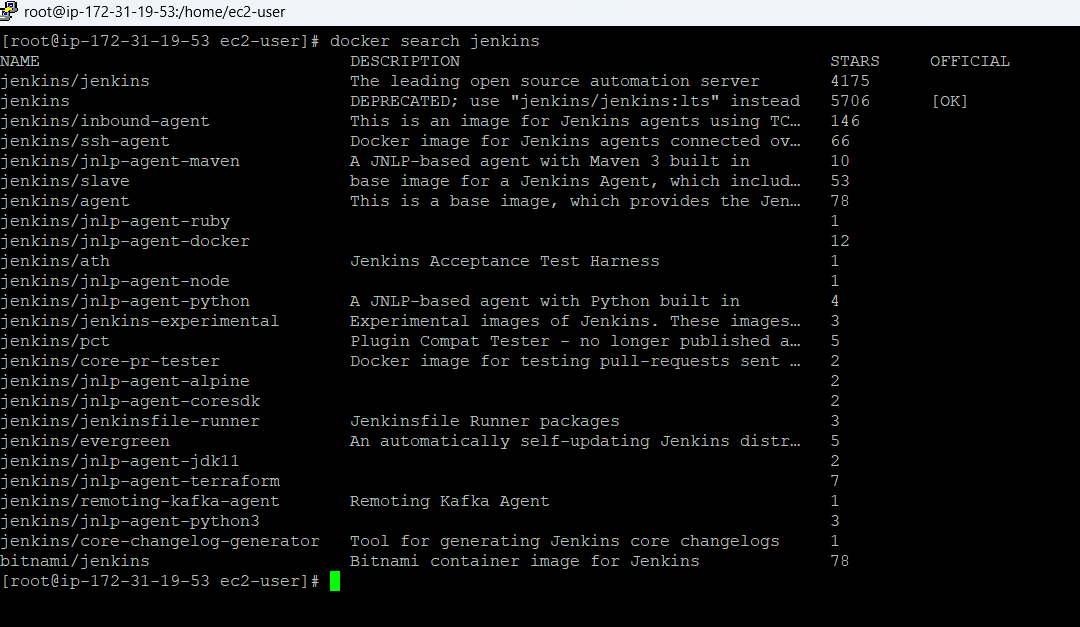
* docker ps
  + To see only running containers. PS means, process status
* docker start <container>
  + To start container
* docker attach <container>
  + To go inside container
* docker stop <container>
  + To stop a container
* docker rm <container>
  + To delete a container
* docker exec -it <container> <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.



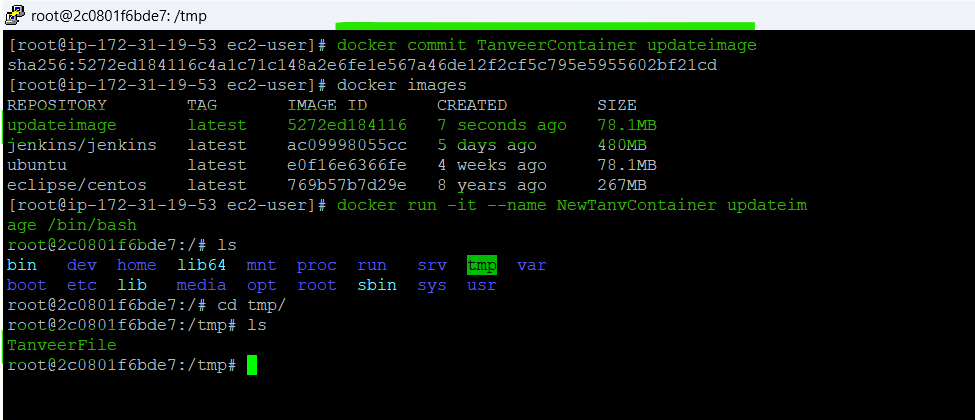
* docker search <image>
  + To find out an image in docker hub
  + For example, “docker search Jenkins” to search all Jenkins images



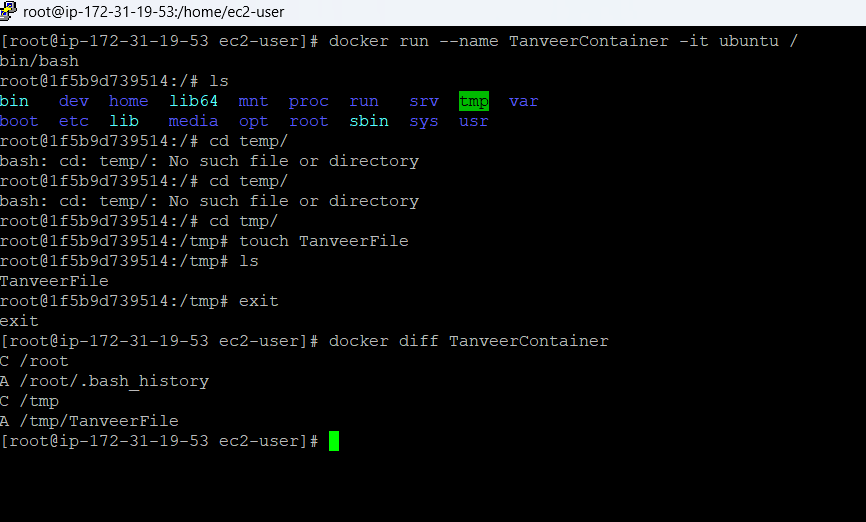
* docker pull <image>
  + Pulls an image from a Docker registry (e.g., Docker Hub) to local machine
  + For example, “docker pull Jenkins” to pull Jenkins image to your local machine
* docker rmi <image>
  + Removes one or more images.
* docker push <image>
  + Pushes an image to a Docker registry.

### 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



* 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



## 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/>

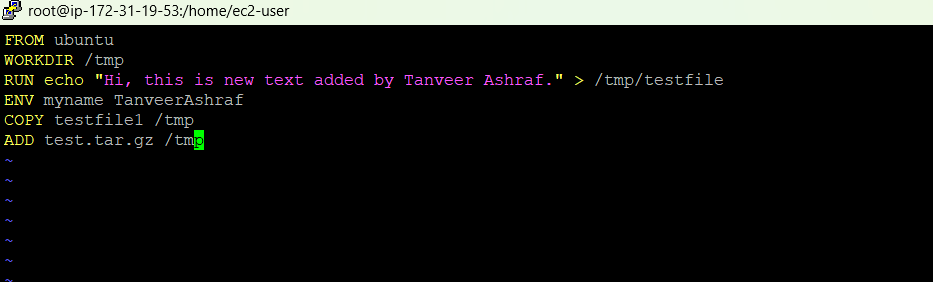
### Create Image

There are 3 ways to create image

1. Pull an image from Docker Hub
2. Create an image from a container
   1. 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

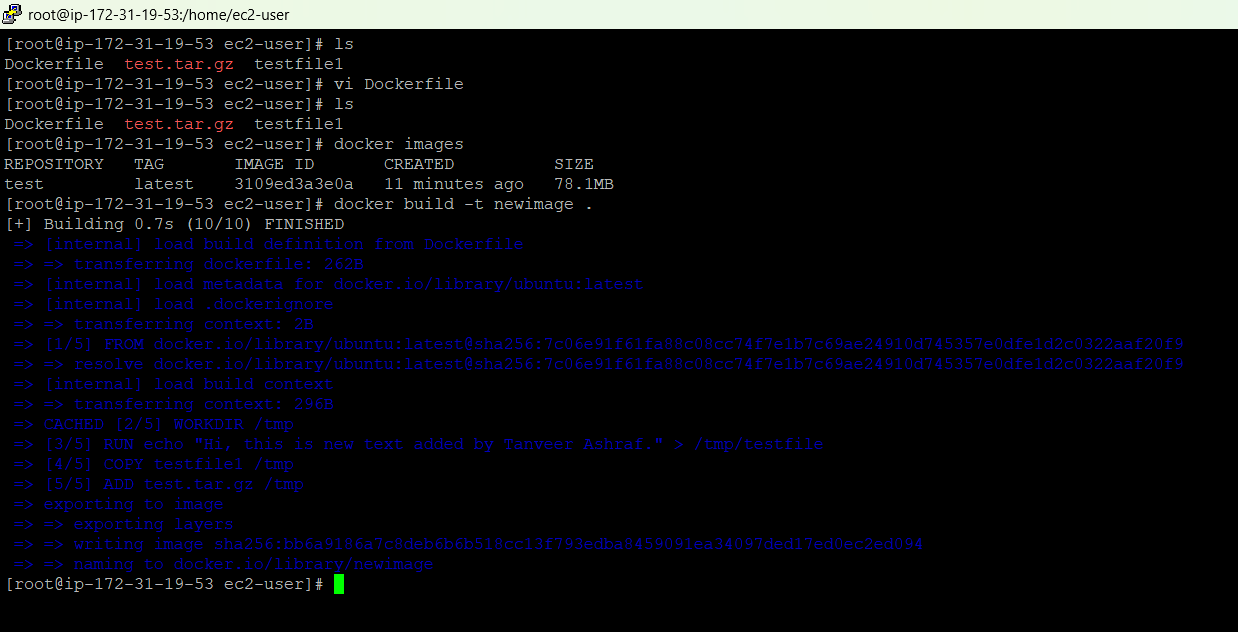
### Create Docker file

1. Create a file named as Dockerfile
   1. 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
   1. docker build -t <image> . The “.” Means to create image from current docker file
4. Run image to create container
   1. docker run -it --name <container> <image> /bin/bash

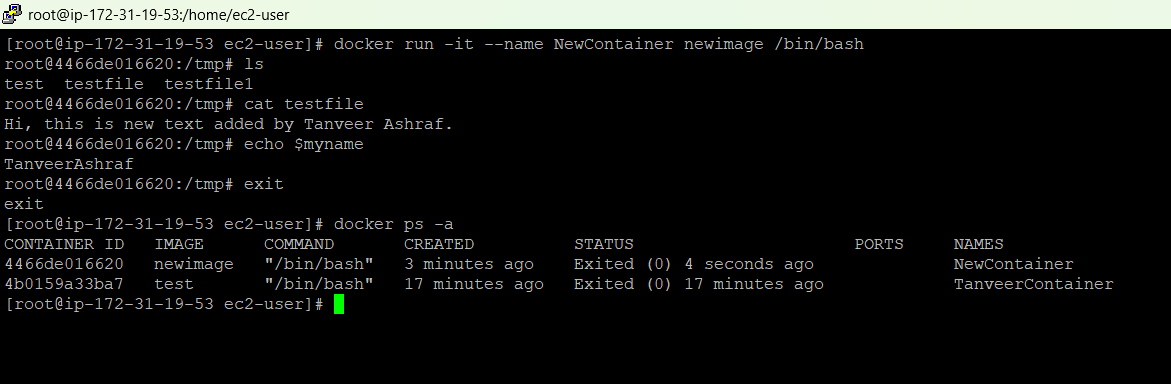


In above docker file, we are saying

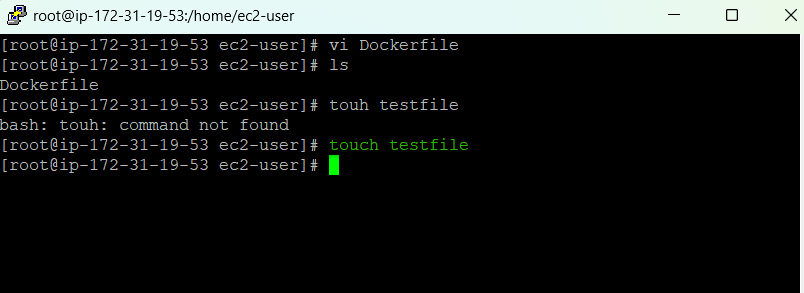
1. build a new docker image based on ubuntu
2. the working directory will be temp
3. add a string in testfile inside tmp directory
4. the environment variable is myname
5. copy the testfile1 from temp directory
6. download a file and unzip



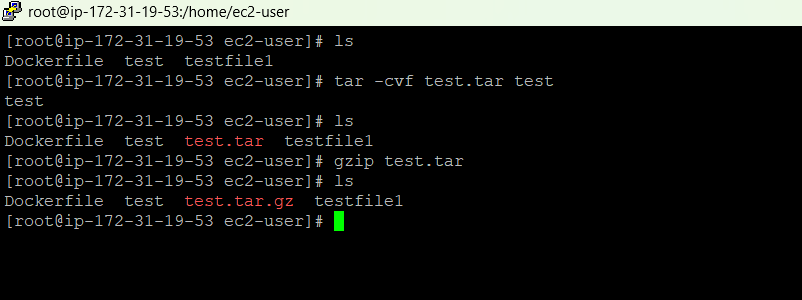
As seen above, a new docker image with name “newimage” is successfully created from the Dockerfile. Now, we can create a container from that image as below. To print the environment variable we can echo with $ sign



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



To convert a file to zip file



## Docker Volume

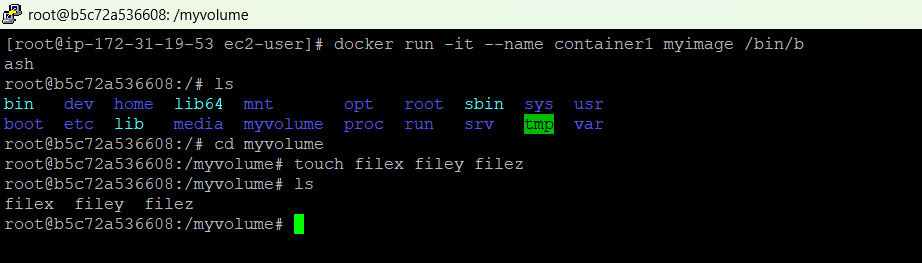
* Volume is simply a directory inside our container
* First, we have to declare a directory as volume, and then we share the volume
* Even if we stop the container, the volume still can be accessible
* Volume will be created in one container
* You can declare a directory as a volume ***only while creating 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

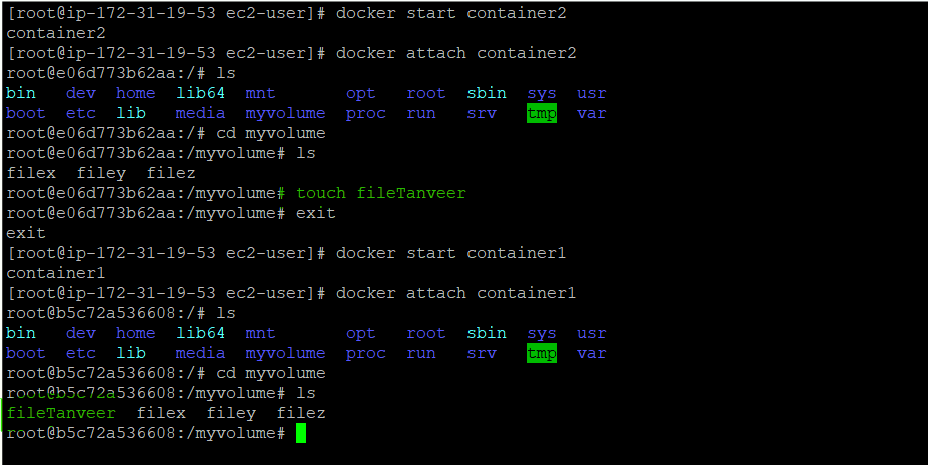
* Decoupling container from storage
* Share volume among different containers
* Attach volume to containers
* On deleting container, volume does not delete

### Creating volume – step by step

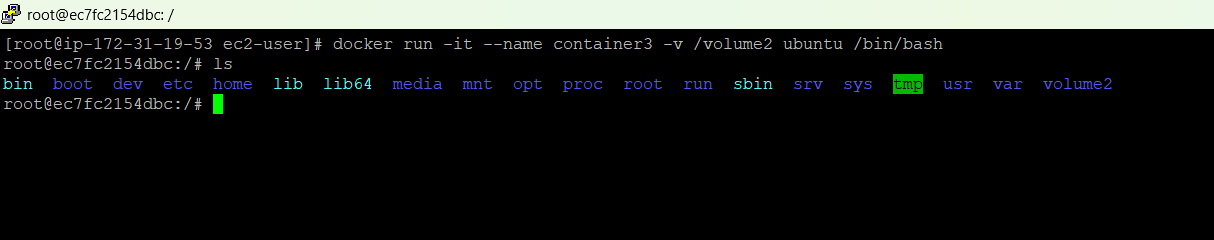
* Create a docker file and write some commands for example
  + FROM ubuntu
  + VOLUME [“volumeName”]
* Create image from this docker file
  + docker build -t <image> .
  + This will create image with provided name using instructions given in the docker file. -t means tag and is used to give image name
* 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”



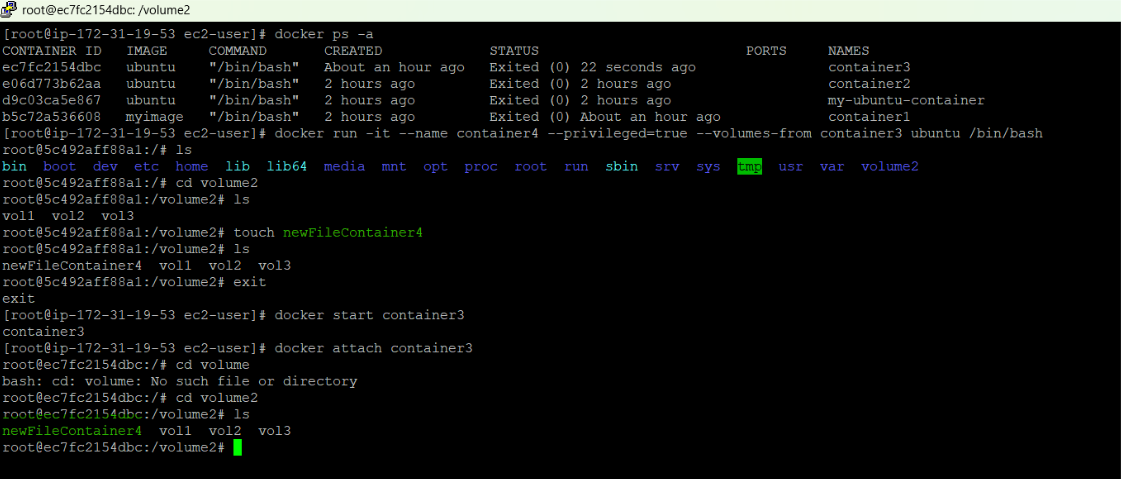
* Share this volume with other containers
  + docker run -it --name <newContainer> --privileged=true –volumes-from <oldContainer> <image> /bin/bash
  + privileged=true means to give rights to the new container to add/update the volume content
* Now after creating new container, the volume will be visible 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>



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



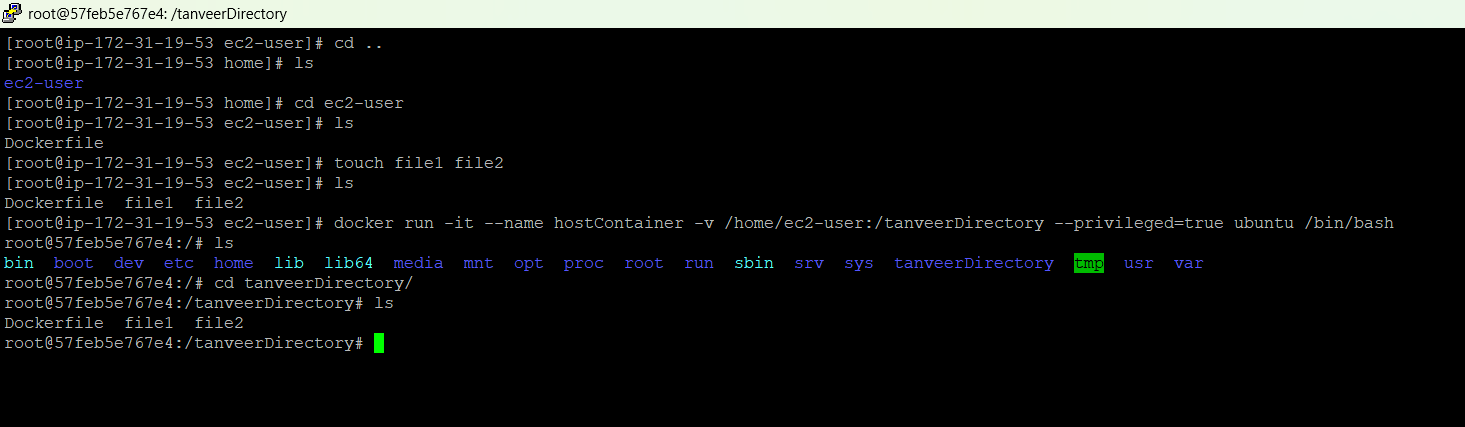
* + In below example, we have a container 3, in which we created a volume as “volume2”. This volume contains 3 files vol1, vol2 and vol3. Create a new container “container4” with sharing the volume and add a new file in volume2 inside container 4. This will be visible in container3 as well and vice versa.



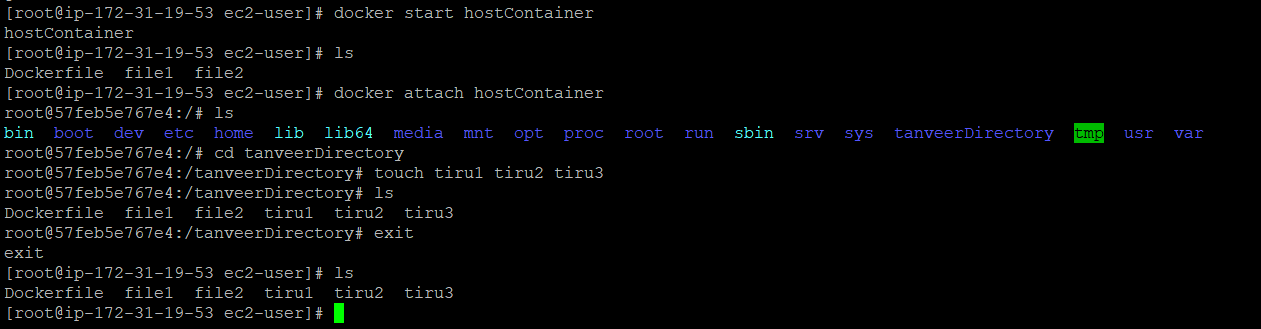
### Share volume between host and container

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 --priviliged=true ubuntu /bin/bash

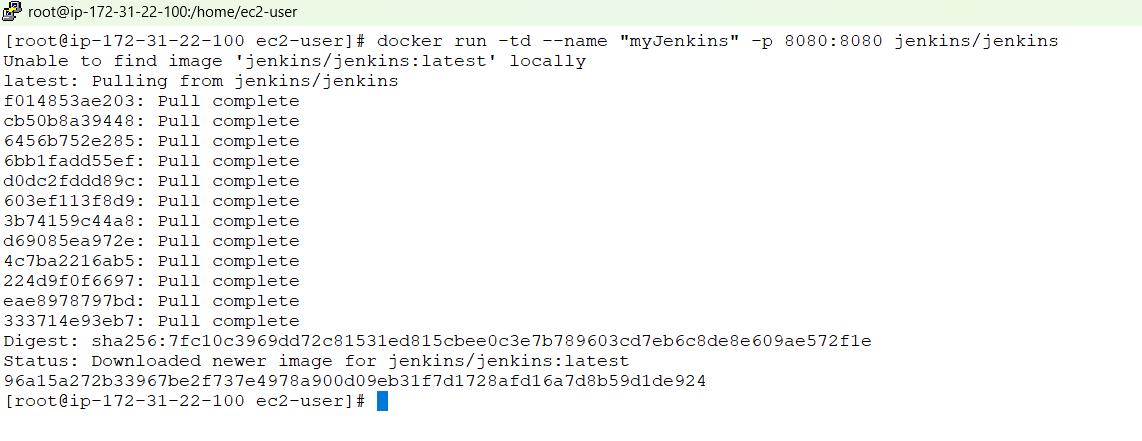


* we can also verify by creating more files inside the container and check the same files existence from host as below

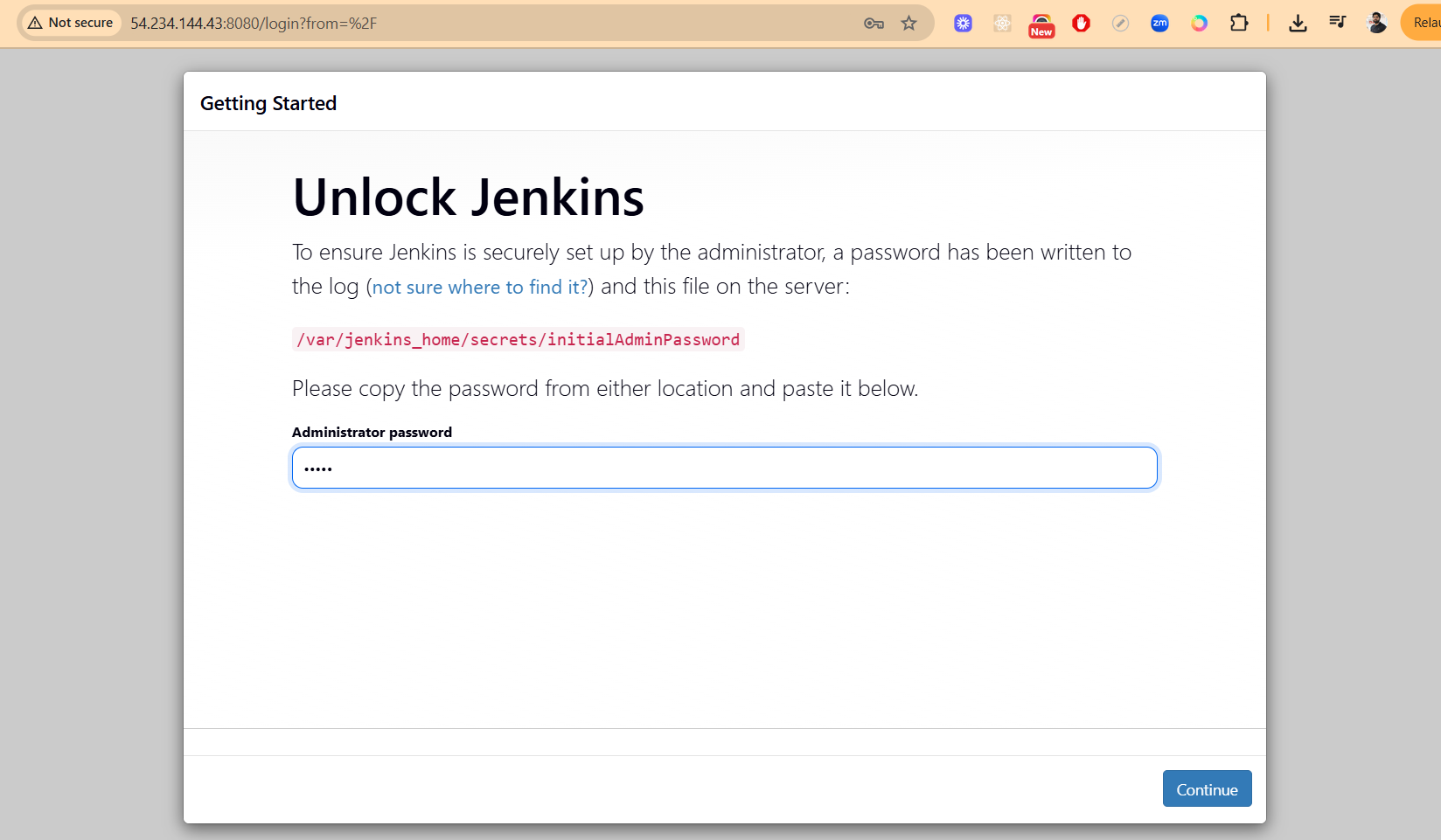


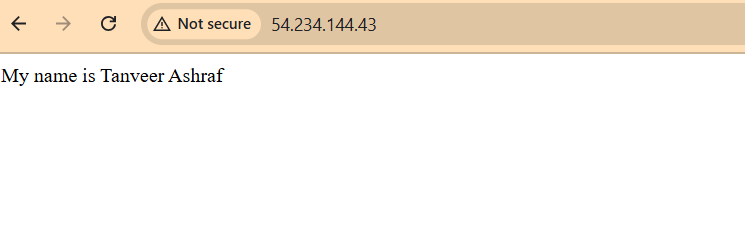
## Docker port expose

* 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)
    1. Keeps STDIN open even if not attached.
    2. Useful if you want to interact with the container (e.g., run a shell).
  + **-t** (tty)
    1. Allocates a pseudo-TTY (terminal).
    2. Gives you a shell-like experience (colors, cursor movement, etc.).
  + **-d** (detached)
    1. Runs the container in the background.
    2. You don’t stay “attached” to the container’s logs/terminal.
    3. 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
    4. Similarly, for example, you want to run Jenkins on port 8080, use below command

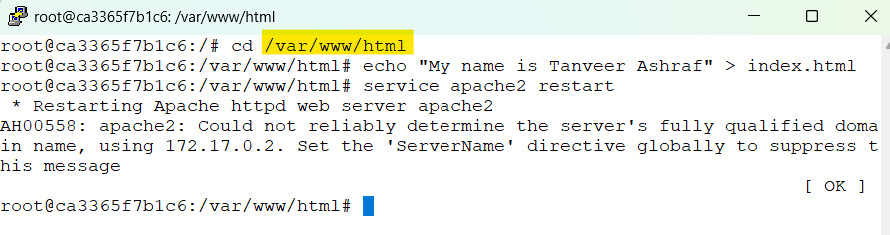


Now if you browse from the browser, just copy your public IP of the instance and add port 8080 will show you Jenkins, and port 80 will show you the simple html page





* 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
    1. This will add the above text in index.html file



* 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
    1. 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.
  + both host and container ports are 8080 and container name is myJenkins
  + 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)
   1. In this case the service in the container will only be accessible from inside the container
2. Only specify **expose**
   1. 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**
   1. The service in the container is accessible from anywhere, even outside docker
   2. 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