Docker Hands-On: From Basics to Real-World Use

Introduction

Docker is an open platform for building, shipping, and running containerized applications. It simplifies software delivery by isolating applications in containers — lightweight, executable units that include everything needed to run software: code, runtime, system tools, libraries, and settings.

Virtualization vs Containerization

Virtualization enables running multiple virtual machines (VMs) on a single physical system using a hypervisor.
Containerization, on the other hand, allows multiple containers to run on a single OS kernel, making them more lightweight and faster.

Hypervisors

Type 1 Hypervisor (Bare Metal)
Installed directly on the hardware.
Examples: KVM, VMware ESXi, Xen

Type 2 Hypervisor
Runs on top of an existing OS.
Example: VirtualBox

🔹 **What is Docker?**

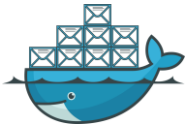
Docker is an open-source platform that allows developers and system administrators to **build, run, and manage applications in lightweight, portable containers**.

Each **container** includes the application, its dependencies, libraries, and configuration—bundled together to run consistently across different environments (development, testing, production).

🔹 **Why Docker?**

Docker solves many common challenges like:

✅ **"It works on my machine"** issues
✅ Dependency conflicts
✅ Slow setup times
✅ Inconsistent environments



With Docker, you can:

- Create **isolated environments** for every application
- **Launch containers in seconds**, not minutes
- **Reuse images** to save time and resources
- **Run the same image on any system** (as long as Docker is installed)

🔹 How Docker Works?

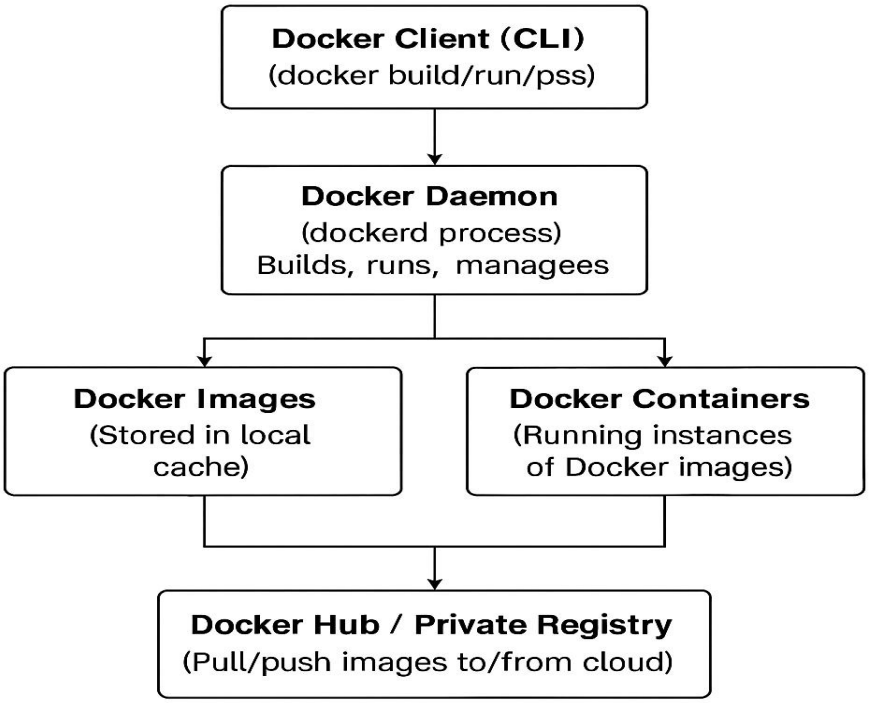
1. A **developer writes instructions** in a **Dockerfile** (like: install Python, copy app files, set up Apache).
2. Docker uses the Dockerfile to **build an image**.
3. From that image, Docker **creates a container**, which is a running instance.
4. The container runs the app with all its dependencies, isolated from the host system and other containers.

Behind the scenes:

- Docker uses the **host OS kernel** (no full OS inside containers).
- **Docker Engine** manages everything — building images, creating containers, networking, and volumes.
- Docker architecture diagram
- Sample Dockerfile
- Running a container with docker run -it ubuntu bash

🔹 Key Concepts

- **Image**: Read-only template used to create containers.
- **Container**: A running instance of an image.
- **Dockerfile**: Script with instructions to build Docker images.
- **Registry**: Storage for Docker images (e.g., Docker Hub).



Screenshot 1: Docker architecture diagram

Docker Installation & Setup

yum install -y docker

systemctl start docker

docker --version

```
[root@ip-172-31-33-40 ~]# systemctl start docker
[root@ip-172-31-33-40 ~]# docker --version
Docker version 25.0.8, build 0bab007
[root@ip-172-31-33-40 ~]#
```

Screenshot 2: Docker installation and version output

Working with Images

Pulling Images

docker pull ubuntu

docker pull ubuntu:22.04

docker images

```
[root@ip-172-31-33-40 ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
ubuntu       latest    65ae7a6f3544   3 weeks ago   78.1MB
ubuntu       22.04     1d3ca894b30c   3 weeks ago   77.9MB
```

Screenshot 3: Output of docker images

Deleting Images

docker rmi -f ubuntu

docker system prune -a -f

Creating and Running Containers

Basic Run

docker run ubuntu

docker run ubuntu:20.04

Interactive Shell

docker run -it ubuntu bash

Background Container

docker run -itd ubuntu

Stop, Start, Attach

docker stop <container-id>

docker start <container-id>

docker attach <container-id>

```
[root@ip-172-31-33-40 ~]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS         PORTS     NAMES
e6e0a335a6c7   ubuntu    "/bin/bash"   2 minutes ago   Up 3 seconds             blissful_nash
[root@ip-172-31-33-40 ~]#
```

Screenshot 4: List of running containers docker ps

Naming & Managing Containers

docker rename old_name new_name

docker run --name mycontainer ubuntu

docker ps -a

```
[root@ip-172-31-33-40 ~]# docker rename blissful_nash container1
[root@ip-172-31-33-40 ~]# docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED         STATUS              PORTS     NAMES
e6e0a335a6c7   ubuntu    "/bin/bash"   4 minutes ago   Up About a minute             container1
[root@ip-172-31-33-40 ~]# docker run --name mycontainer ubuntu
[root@ip-172-31-33-40 ~]# docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS                     PORTS     NAMES
e47506b5f851   ubuntu         "/bin/bash"   5 seconds ago   Exited (0) 4 seconds ago             mycontainer
e6e0a335a6c7   ubuntu         "/bin/bash"   4 minutes ago   Up 2 minutes                         container1
13902bad66cb   ubuntu         "bash"        5 minutes ago   Exited (0) 5 minutes ago             jolly_raman
b14e4a788ab6   ubuntu:20.04   "/bin/bash"   5 minutes ago   Exited (0) 5 minutes ago             bold_babbage
a713d5d98f05   ubuntu         "/bin/bash"   6 minutes ago   Exited (0) 6 minutes ago             friendly_galois
```

Screenshot 5: Renaming and listing containers

Port Mapping and Web Server Setup

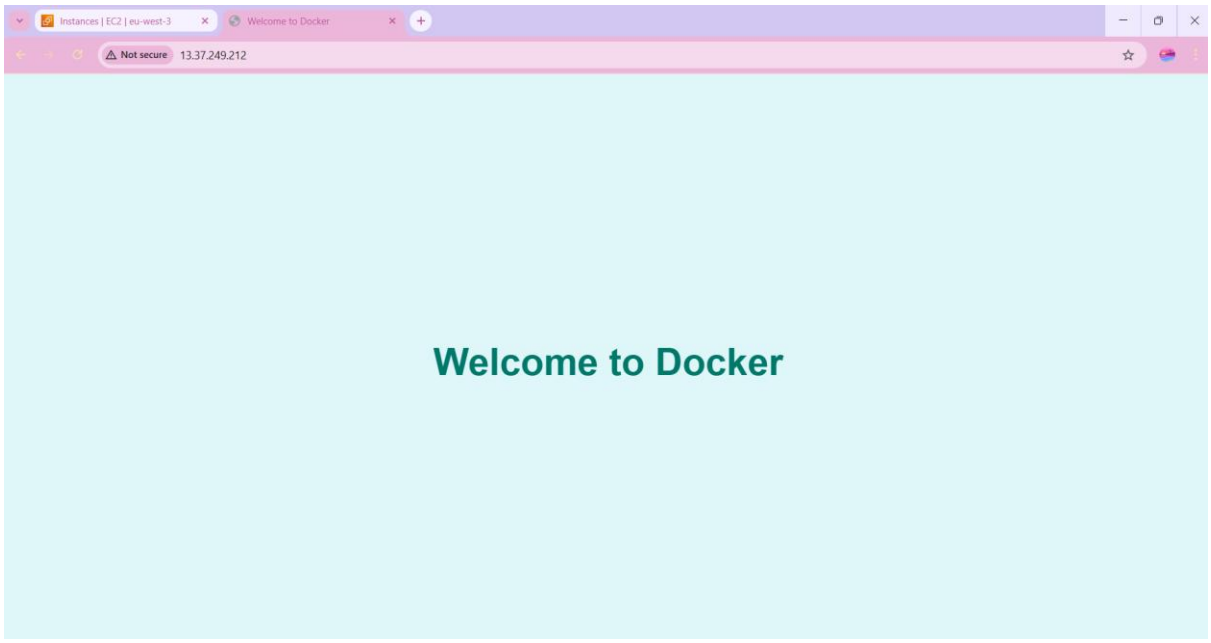
docker run -it -p 80:80 ubuntu

apt-get update -y

apt-get install apache2 -y

service apache2 start

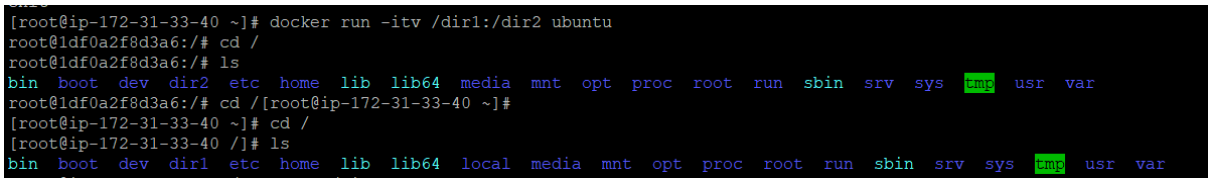
echo "Welcome to Docker" > /var/www/html/index.html



Screenshot 6: Web page displayed in browser (localhost)

Docker Volume (Host ↔ Container Directory Sync)

docker run -it -v /host/dir:/container/dir ubuntu



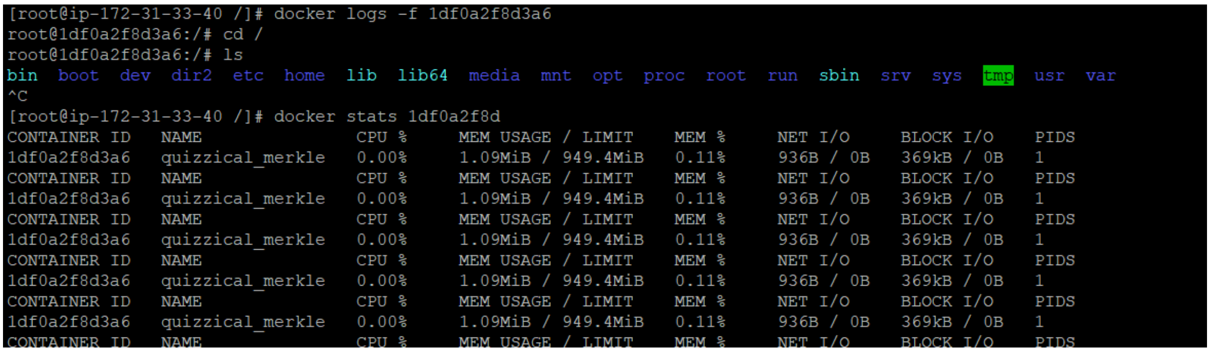
Screenshot 7: Host file visible inside the container

Container Logs & Stats

docker logs -f <container-id>

docker stats <container-id>

docker info



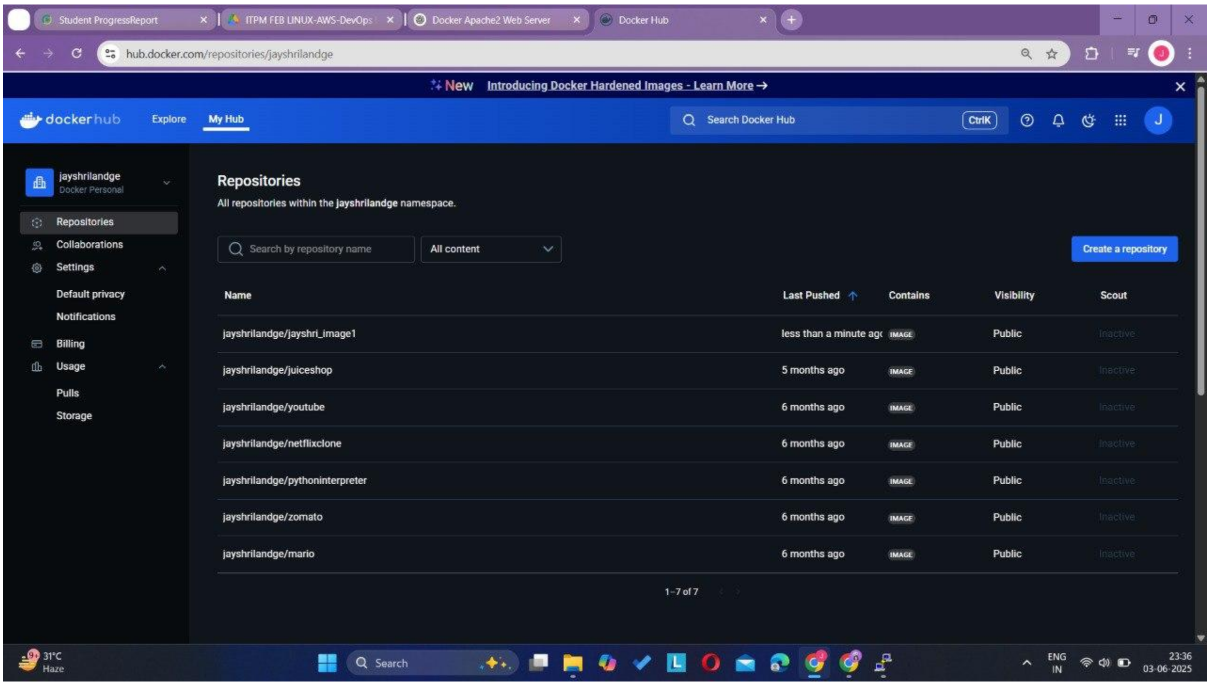
Screenshot: Real-time CPU and memory usage from docker stats

Commit & Push to Docker Hub

docker commit mycontainer jayshri_image1

docker login

docker push username/jayshri_image1



Screenshot: Custom image pushed to Docker Hub

Dockerfile Basics

Dockerfile Example

FROM ubuntu

MAINTAINER Jayshri

WORKDIR /tmp

RUN echo "Hello all" > /file1

Build Image from Dockerfile

docker build -t customimage /path/to/Dockerfile

Run Container from Custom Image

docker run -it customimage bash

```
[root@ip-172-31-33-40 /]# cat Dockerfile
FROM ubuntu
MAINTAINER Jayshri
WORKDIR /tmp
RUN echo "Hello all" > /file1

[root@ip-172-31-33-40 /]# docker build -t jayshri_image1 .
[+] Building 0.1s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 173B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/ubuntu:latest
 => CACHED [2/3] WORKDIR /tmp
 => CACHED [3/3] RUN echo "Hello all" > /file1
 => exporting to image
 => => exporting layers
 => => writing image sha256:95bc8d2f3460389be3761f0391e287f19a0ffc192903ed84c10c5f1cfbb419f2
 => => naming to docker.io/library/jayshri_image1
[root@ip-172-31-33-40 /]# docker images
REPOSITORY        TAG       IMAGE ID       CREATED              SIZE
jayshri_image12   latest    95bc8d2f3460   About a minute ago   78.1MB
jayshri_image1    latest    95bc8d2f3460   About a minute ago   78.1MB
```

Screenshot: Dockerfile and image build output

Optimized Dockerfile (Chained RUN)

FROM ubuntu

RUN apt-get update -y && apt-get install tree git apache2 -y

Screenshot: Optimized image layer output

Docker Compose: Managing Multi-Container Environments

Installation

curl -L --fail https://github.com/docker/compose/releases/download/1.29.2/run.sh -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose

Compose File Example

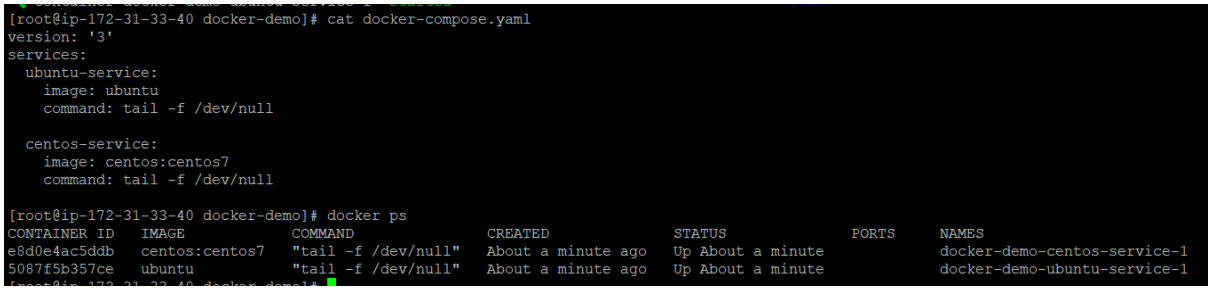
```
version: '3'

services:

 ubuntu-service:

   image: ubuntu

   command: tail -f /dev/null

 centos-service:

   image: centos:centos7

   command: tail -f /dev/null
```

Running Compose

docker-compose up -d

docker ps

docker exec -it ubuntu-service bash



Screenshot: Compose file and docker ps output

Ubuntu Web Server with Compose + Volume + Port Mapping

```
version: '3.8'

services:

 ubuntu-apache:

   image: ubuntu:22.04

   container_name: ubuntu_apache_server

   ports:

    - "80:80"

   volumes:
```

- /dir1:/dir2

command: >

bash -c "apt-get update &&

apt-get install -y apache2 &&

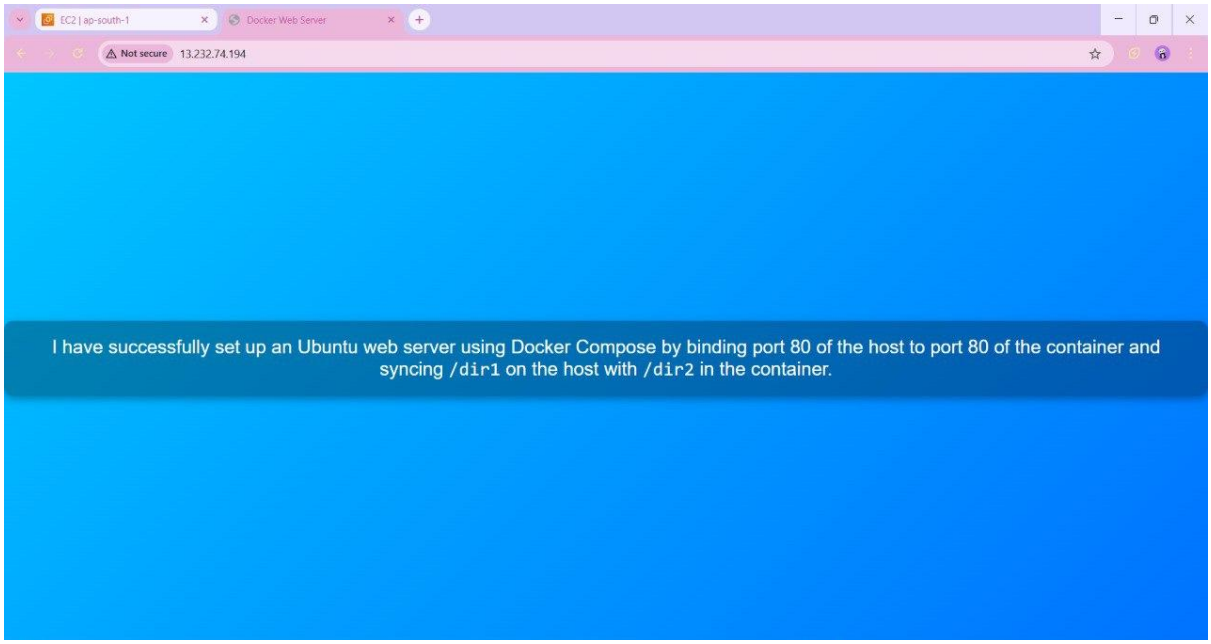
echo ' I have successfully set up an Ubuntu web server using Docker Compose by binding port 80 of the host to port 80 of the container and syncing /dir1 on the host with /dir2 in the container.' > /var/www/html/index.html &&

apachectl -D FOREGROUND"

mkdir /dir1

touch /dir1/file1

docker-compose up -d



Screenshot: Apache homepage output in browser

Formats in Dockerfile

Shell Format

FROM ubuntu

RUN mkdir /dir1

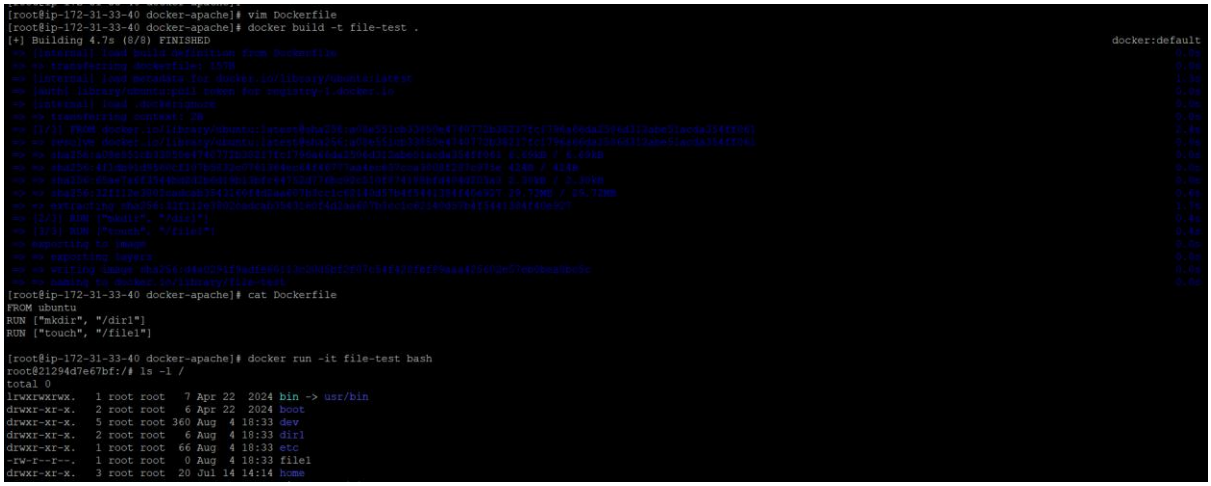
RUN touch /file1

Exec Format

FROM ubuntu

RUN ["mkdir", "/dir1"]

RUN ["touch", "/file1"]



Screenshot: File creation verification in container

Conclusion

This hands-on practice helped me develop a clear understanding of Docker fundamentals, including building custom images using both shell and exec formats in Dockerfiles, managing containers, and using Docker Compose for running multi-container environments. I practiced setting up services, mounting volumes, verifying file operations inside containers, and deploying a basic Apache server. These activities have improved my confidence in working with containerized environments and strengthened my foundational skills in Docker and container management, which are essential for DevOps and modern development practices.