Docker Simplified 1/5

Docker is an open-source platform that allows you to build, package, and run applications inside lightweight, portable containers. Containers ensure consistency across environments, making deployment faster, scalable, and more reliable compared to traditional virtual machines.

```
→ apt-get update                          # Update Linux repositories
→ apt install docker.io                   # Install Docker
→ docker images                           # List Docker images
→ docker pull httpd                       # Download an image
→ docker ps                               # List running containers
→ docker ps -a                            # List all containers (running + stopped)
→ docker run -itd -p "8080:80" httpd
                                          # Run Apache HTTPD on port 8080
→ docker run -itd --name "webserver" -p "3212:80" nginx
                                          # Run Nginx with a custom name and port
                                            mapping
→ docker run -itd --name "myjenkins" -p "8888:8080" jenkins/jenkins
                                          # Run Jenkins on port 8888
→ docker stop 45df45656sdf88              # Stop a container
→ docker rm 45df45656sdf88                # Remove a container
→ docker rm -f 45df45656sdf88             # Force remove a container
→ docker rmi httpd                        # Remove an image
→ docker exec -it 45df45656sdf88 /bin/bash
                                          # Access a running container's shell
→ cat /etc/os-release                     # Check OS details
→ docker top 45df45656sdf88               # Check processes running inside container
→ docker stats 45df45656sdf88             # Live stats of container (CPU, RAM,
                                            Network)
→ docker inspect 45df45656sdf88           # Inspect container details (config, IP,
                                            etc.)
→ docker commit 45df45656sdf88 myimage:v1
                                          # Commit container as new image
→ docker build -t mypythonimage:v1 /root
                                          # Build image from Dockerfile(loc - root)
```

Docker Simplified 2/5

1.Sharing of docker image with internet:

```
→ docker tag myimage bkbala3710/newimage:v1
            # Tag a local image with your Docker Hub repo name
            * Now you'll see two copies (same image ID, different tags)
→ docker login                          # Login to Docker Hub
→ docker push bkbala3710/newimage:v1 # Push the tagged image to Docker Hub
→ docker pull bkbala3710/newimage:v1 # Pull the image from Docker Hub
```

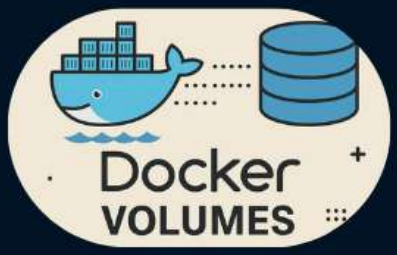
2.Sharing of docker image without internet:

```
→ docker save -o . /mydockerimage.tar myimage:v1
            # Save a Docker image as a tar file
→ scp -i ~/downloads/my-privatekey.pem ./mydockerimage.tar
   ec2-user@3.110.12.45:/home/ec2-user/
            # Copy the image tar file to another server using SCP
→ docker load -i mydockerimage.tar
            # Load the image back into Docker on the target server
```

Docker Storage Types:

1.Bind Mount

```
docker run -itd -p "9000:80" -v /opt/www:/usr/local/apache2/htdocs
bkbala3710/newimage:v1
          # /opt/www (host) is mounted into
            usr/local/apache2/htdocs (container).
```



2.Dedicated volume

```
docker volume ls                # List available volumes
docker volume create my-vol     # Create a volume
docker run -itd -p 9090:80 --mount source=my-vol,
target=/usr/local/apache2/htdocs bkbala3710/newimage:v1
                                # Run container with dedicated volume
```

3.temp mount - Temporary / Anonymous Volume

```
docker run -itd -p 8085:80 -v /usr/local/apache2/htdocs
bkbala3710/newimage:v1          # Run with a temporary anonymous volume
                                # Docker assigns a random volume name,
                                  useful for testing
```

Single-Stage Dockerfile:

```
# Use Maven with JDK for build + Tomcat to run (not recommended for prod)
FROM maven:3.8.5-openjdk-11
WORKDIR /opt/app
# Copy source code
COPY ./javacode .
# Build the WAR file
RUN mvn clean install
# Download and extract Tomcat
RUN apt-get update && apt-get install -y wget
    && wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.82/bin/apache-
    tomcat-9.0.82.tar.gz && tar -xvzf apache-tomcat-9.0.82.tar.gz
    && mv apache-tomcat-9.0.82 tomcat
# Copy WAR into Tomcat
RUN cp target/*.war tomcat/webapps/
WORKDIR /opt/app/tomcat
EXPOSE 8080
ENTRYPOINT ["sh", "bin/startup.sh"]
```

Multi-Stage Dockerfile:

```
# Stage 1: Build with Maven
FROM maven:3.8.5-openjdk-11 AS builder
WORKDIR /opt/app
COPY ./javacode .
RUN mvn clean install   # produces target/app.war
# Stage 2: Lightweight Tomcat runtime
FROM tomcat:9.0.82-jdk11-openjdk-slim
WORKDIR /usr/local/tomcat/webapps
COPY --from=builder /opt/app/target/*.war ./app.war
EXPOSE 8080
ENTRYPOINT ["catalina.sh", "run"]
```

- Single-Stage → ❌ large (~600-800MB) because it includes Maven + JDK + Tomcat + build cache.
- Multi-Stage → ✅ much smaller (~200MB) because final image only has Tomcat + WAR + JDK runtime.

Docker Simplified 4/5

- ENTRYPOINT → Defines the main command that always runs when the container starts.
- CMD → Provides default arguments (can be overridden at runtime).

1. Using CMD alone:

```
FROM ubuntu
CMD ["echo", "Hello World"]
docker run myimage                # Hello world
docker run myimage echo Hi        # Hi

FROM ubuntu
CMD ["echo", "Hello World"]
CMD ["echo", "Welcome Earth"]
docker run myimage                # Welcome Earth
docker run myimage echo Hi        # Hi
```

2. Using ENTRYPOINT alone:

```
FROM ubuntu
ENTRYPOINT ["echo"]
docker run myimage Hello          # Hello
docker run myimage Hi There       # Hi There

FROM ubuntu
ENTRYPOINT ["echo","”check”]
docker run myimage Hello          # check Hello
docker run myimage Hi There       # check Hi There
```

3. ENTRYPOINT + CMD together:

```
FROM ubuntu
ENTRYPOINT ["echo"]
CMD ["Hello World"]
docker run myimage                # Hello World
docker run myimage Hi             # Hi
```

Docker Simplified 5/5

Cleanup Commands:

```
docker container prune      # Removes all stopped containers
docker image prune          # Removes dangling (unused) images
docker system prune         # Removes unused containers, networks, images,
                              cache
```

Docker Networks:

```
docker network create --driver bridge mynet
                            # Create custom bridge network
docker network ls           # List all networks
```

1. bridge → Containers talk via Docker's virtual network, can expose ports.

```
docker run -itd --name cont1 --network mynet -p 8080:80 httpd
```

2. host → Container shares host network (better performance, less isolation).

```
docker run -itd --name cont2 --network host httpd
```

3. none → No network access (used for secure/test cases).

```
docker run -itd --name cont3 --network none httpd
```