

Docker Assignment 6

Multi-stage Dockerfile:

- A Multi-stage Dockerfile is a technique for creating optimized, production-ready container images by using multiple FROM statements in a single file. This approach separates the build environment (which requires compilers and development dependencies) from the runtime environment (which only needs the compiled application and minimal runtime dependencies), resulting in a smaller, more secure final image.
- A multi-stage Dockerfile uses multiple FROM statements to separate the build environment from the runtime environment, resulting in smaller, more secure production images that only contain essential components.
- In a multistage Dockerfile, developers define multiple build stages, each encapsulating a specific set of instructions and dependencies. These stages can be named and referenced within the Dockerfile, enabling seamless communication between them.

Advantages of Multi-Stage Dockerfile:

- **Multiple FROM Statements:** Each FROM keyword starts a new, independent stage of the build process.
- **Separation of Concerns:** The first stage(s) are typically used for development/building, while the final stage is for runtime/production.
- **COPY --from=STAGE_NAME:** This crucial command allows you to pull specific artifacts, such as compiled binaries or static files, from a previous stage without bringing over intermediate files, build tools, or source code.
- **Lightweight Production Images:** The final image often uses minimal base images like Alpine or Distroless, which do not include compilers or development tools, minimizing the attack surface.
- **Named Stages:** Stages can be named (e.g., FROM golang:1.23 AS builder) to make the COPY --from=builder command more readable.
- **Enhanced Security:** Fewer tools and libraries in the final image mean a smaller attack surface

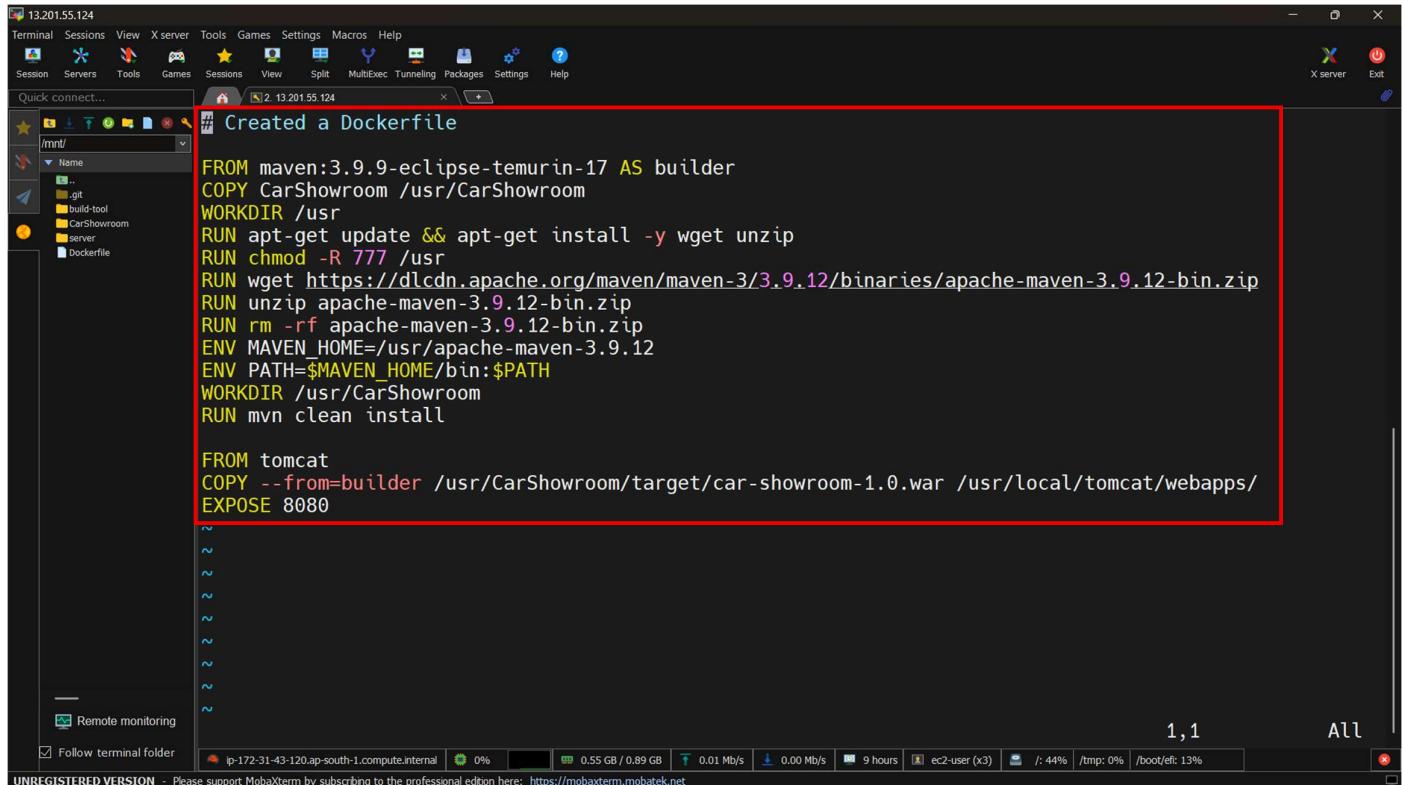
Step 1: Launched an instance for our Docker Host:

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with options like Dashboard, AWS Global View, Events, Instances (selected), Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, and Credits. The main area displays a table titled 'Instances (1/10)'. The table has columns for Name, Instance ID, Instance state, Instance type, and Status check. There are three instances listed: 'D3 Linux Jenkins Master' (Stopped, t3.micro), 'D5 Linux Instance' (Stopped, t3.micro), and 'D6 Linux Instance' (Running, t3.micro). The 'D6 Linux Instance' row is highlighted with a red box. Below the table, a detailed view for 'i-0a2674ddfbbebd053d (D6 Linux Instance)' is shown, with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The 'Details' tab is selected. Under 'Instance summary', it shows the Instance ID and Public IPv4 address (13.201.55.124). The status bar at the bottom indicates the instance is running on a t3.micro instance type.

Step 2: Installed Docker on the Docker Host and created a Car-Showroom Application:

The screenshot shows a terminal session in MobaXterm. The title bar says '13.201.55.124'. The terminal window shows a root shell on an Amazon Linux host. The user runs 'docker --version' and gets the output 'Docker version 25.0.14, build 0bab007'. Then they run 'll' to list the contents of the current directory, which includes 'CarShowroom' (a directory) and 'Dockerfile' (a file). The 'CarShowroom' directory is highlighted with a red box. The status bar at the bottom shows system information like CPU usage, memory, and network activity.

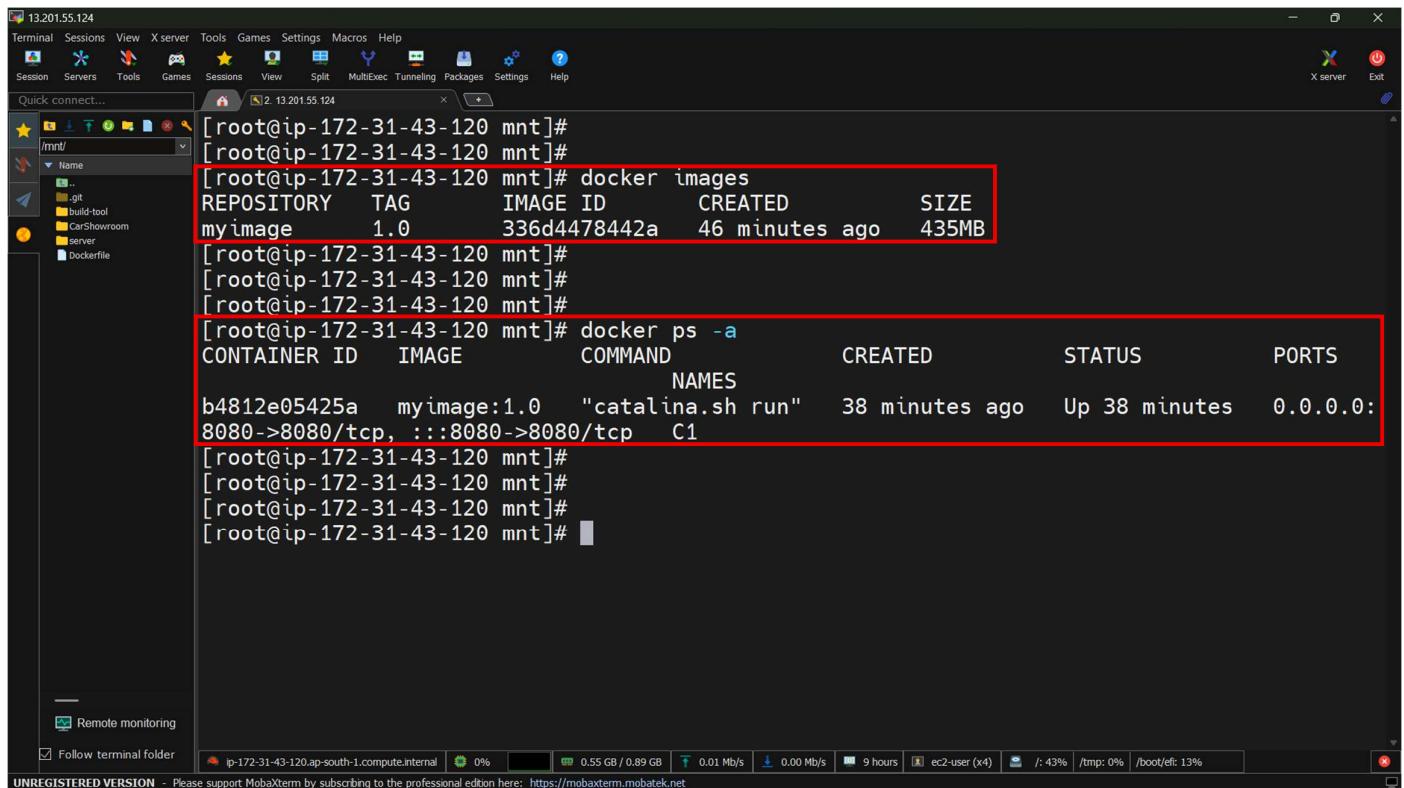
Step 3: Created a ‘Multi-Stage Dockerfile’ on the Docker Host /mnt folder:



```
FROM maven:3.9.9-eclipse-temurin-17 AS builder
COPY CarShowroom /usr/CarShowroom
WORKDIR /usr
RUN apt-get update && apt-get install -y wget unzip
RUN chmod -R 777 /usr
RUN wget https://dlcdn.apache.org/maven/maven-3/3.9.12/binaries/apache-maven-3.9.12-bin.zip
RUN unzip apache-maven-3.9.12-bin.zip
RUN rm -rf apache-maven-3.9.12-bin.zip
ENV MAVEN_HOME=/usr/apache-maven-3.9.12
ENV PATH=$MAVEN_HOME/bin:$PATH
WORKDIR /usr/CarShowroom
RUN mvn clean install

FROM tomcat
COPY --from=builder /usr/CarShowroom/target/car-showroom-1.0.war /usr/local/tomcat/webapps/
EXPOSE 8080
```

Step 4: Created a new image named ‘myimage:1.0’ and a new Container ‘C1’ by using the new image:



```
[root@ip-172-31-43-120 mnt]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
myimage 1.0 336d4478442a 46 minutes ago 435MB

[root@ip-172-31-43-120 mnt]# docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
b4812e05425a myimage:1.0 "catalina.sh run" 38 minutes ago Up 38 minutes 0.0.0.0:
8080->8080/tcp, :::8080->8080/tcp C1
```

Results:

1. First, we have created an image ‘myimage:1.0’ by using the ‘Multi-stage Dockerfile’ and further, we have created a container ‘C1’. Then, we have run our ‘car-showroom-1.0’ application using Tomcat-10 Server on Port No. 8080 from the container:

