

Docker Assignment 6

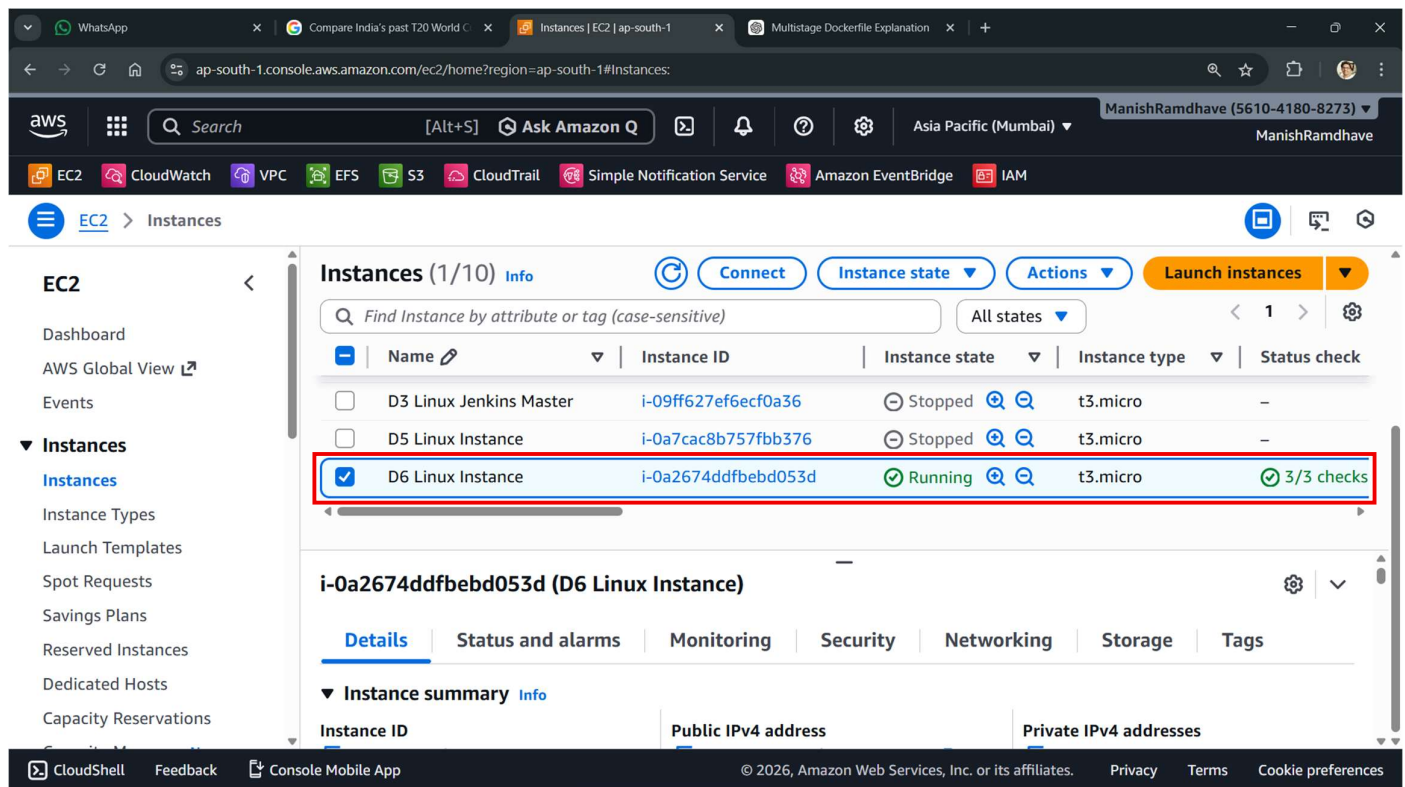
Multi-stage Dockerfile:

- 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 Distrolless, 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.

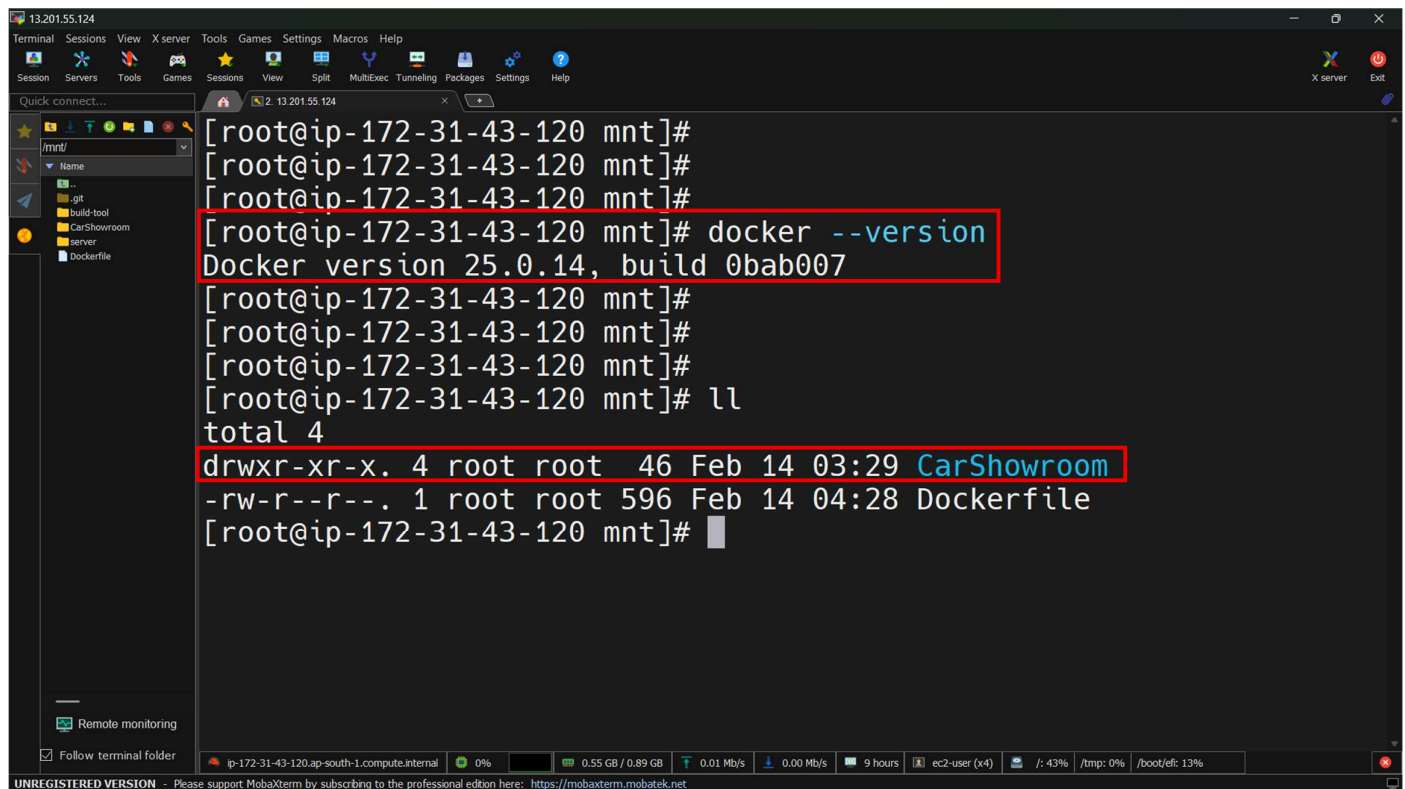
Step 1: Launched an instance for our **Docker Host**:



The screenshot shows the AWS Management Console for the 'ap-south-1' region. The 'Instances' page displays a list of EC2 instances. The 'D6 Linux Instance' (ID: i-0a2674ddfbabd053d) is highlighted with a red box, showing it is in the 'Running' state with '3/3 checks' passed. Below the table, the details for this instance are shown, including tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags.

Name	Instance ID	Instance state	Instance type	Status check
D3 Linux Jenkins Master	i-09ff627ef6ecf0a36	Stopped	t3.micro	-
D5 Linux Instance	i-0a7cac8b757fbb376	Stopped	t3.micro	-
D6 Linux Instance	i-0a2674ddfbabd053d	Running	t3.micro	3/3 checks

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



```
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]# docker --version  
Docker version 25.0.14, build 0bab007  
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]#  
[root@ip-172-31-43-120 mnt]# ll  
total 4  
drwxr-xr-x. 4 root root 46 Feb 14 03:29 CarShowroom  
-rw-r--r--. 1 root root 596 Feb 14 04:28 Dockerfile  
[root@ip-172-31-43-120 mnt]#
```

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

```
Created a Dockerfile

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

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:

