Deployment of Java Web Application on Apache Tomcat Server

Introduction

In the modern software development environment, deploying applications in a reliable and efficient manner is very important. Cloud computing provides scalable infrastructure that helps organizations host applications with high availability. DevOps practices simplify the deployment process by automating build and deployment tasks. Java is widely used for developing enterprise-level web applications, and Apache Tomcat is a popular server for hosting Java applications. Docker is used to containerize applications, ensuring consistency across different environments. In this project, a Java web application is deployed on an Apache Tomcat server using Docker. The application source code is maintained in GitHub and built into a WAR file using Maven. The Dockerized Tomcat application is hosted on an AWS EC2 instance. Users can access the application through a web browser using the EC2 public IP address. This project provides hands-on experience with cloud computing, containerization, and DevOps deployment workflows.

This project demonstrates the deployment of a Java web application on an Apache Tomcat server using Docker, hosted on an AWS EC2 instance. The application source code is managed in GitHub and built into a WAR file using Maven. Docker is used to ensure consistent and portable deployment, while AWS EC2 provides scalable cloud infrastructure. The project highlights practical DevOps concepts such as containerization, build automation, and cloud-based application deployment.

Problem Statement

Many Java web applications are deployed using manual and traditional methods, which often lead to configuration errors and inconsistent environments. Such deployments are difficult to manage and maintain, especially when applications need to be scaled. Differences between development and production systems can cause unexpected application issues. Manual deployment processes are also time-consuming and inefficient. Therefore, there is a need for a standardized and reliable approach to deploy Java web applications efficiently.

Objective

This project is carried out to understand how Java web applications can be deployed in a structured and reliable manner using modern DevOps tools and cloud platforms. The focus is on reducing manual deployment effort and achieving consistency across environments by using containerization and cloud infrastructure.

The key objectives of this project include:

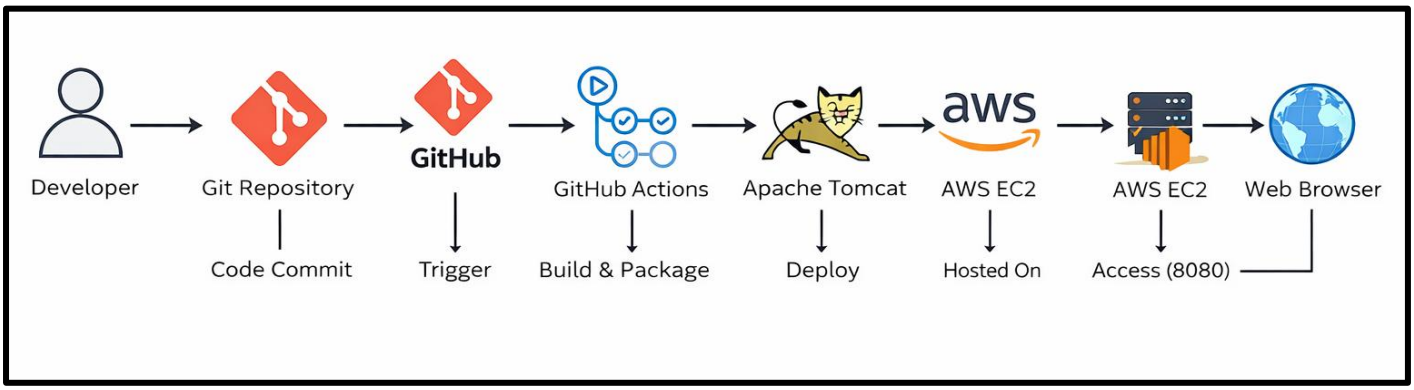
- Deploying a Java-based web application using the Apache Tomcat server
- Automating the build process by converting source code into a WAR file using Maven
- Using Docker to create a portable and consistent deployment environment
- Hosting the application on an AWS EC2 instance for cloud-based access
- Managing source code efficiently using GitHub
- Gaining hands-on experience with real-world DevOps deployment workflows

Architecture Overview

The architecture of this project follows a simple cloud-based deployment model for a Java web application. The source code of the application is maintained in a GitHub repository, which acts as the central version control system. An AWS EC2 instance is used as the compute server to host the application.

Inside the EC2 instance, Docker is installed to provide containerization support. Apache Tomcat runs inside a Docker container and acts as the web server for the Java application. The Java application is built into a WAR file using Maven and deployed inside the Tomcat container. Tomcat listens on port 8080 and handles incoming HTTP requests from users.

When a user accesses the application through a web browser using the EC2 public IP address, the request is routed to the Tomcat server running inside the Docker container. Tomcat processes the request and sends the response back to the user through the same path. This architecture ensures a consistent, portable, and scalable deployment environment by combining cloud infrastructure with containerization.



Components Used

The following components were used in the implementation of this project:

1. **GitHub Repository**

   GitHub acts as a centralized repository to store the Java application source code. It also serves as the platform that triggers automated workflows when new code is pushed.

2. **GitHub Actions**

   GitHub Actions is used to automate the build and deployment process. When code is pushed to the repository, GitHub Actions automatically pulls the code, builds the application, and deploys it to the server.

3. **Maven**

   Maven is used as the build automation tool for the Java application. It compiles the source code, manages dependencies, and generates the WAR file required for deployment.

4. **Apache Tomcat**

   Apache Tomcat is a Java-based web server used to host and run the Java web application. The generated WAR file is deployed inside the Tomcat server to handle client requests.

5. **AWS EC2**

   Amazon EC2 provides the cloud-based virtual server where Apache Tomcat is hosted. It offers scalable compute resources and allows the application to be accessed over the internet.

6. **Web Browser**

   A web browser is used by end users to access the deployed application using the EC2 public IP address and port 8080.

Technologies Used

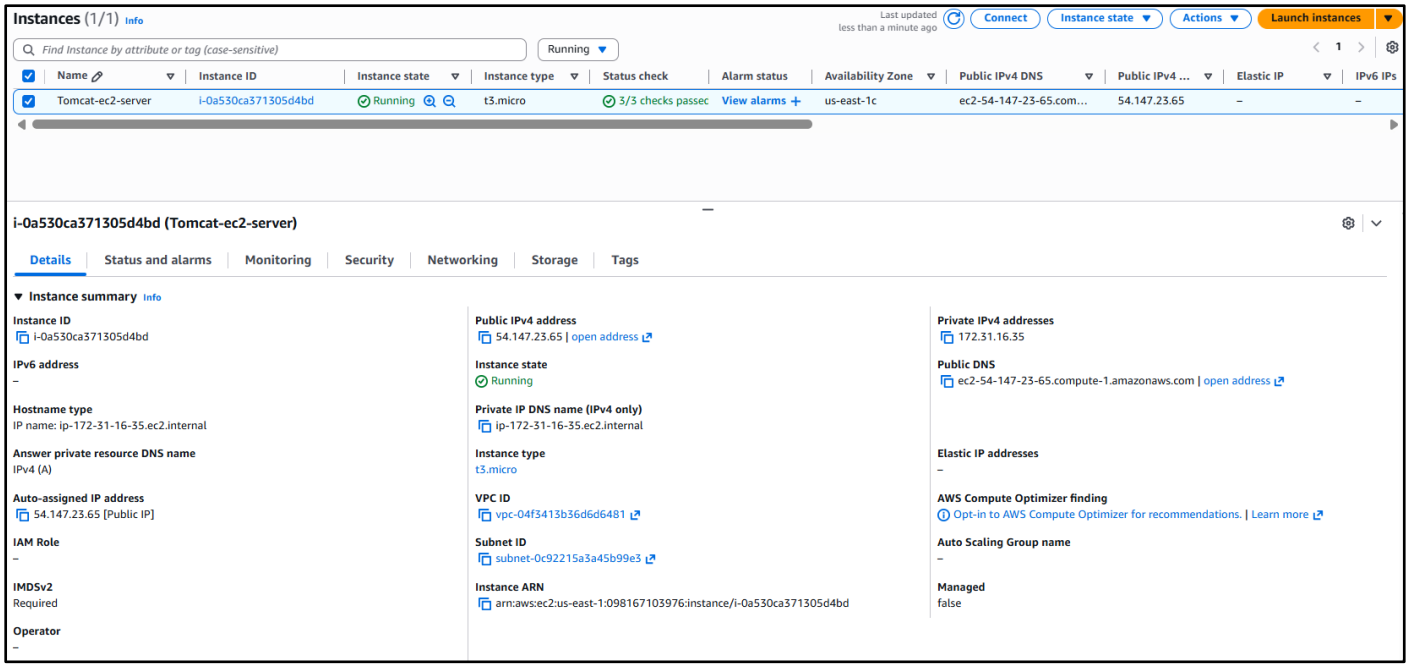
Layer	Technology
Cloud Platform	Amazon Web Services (AWS)
Compute	EC2 (Ubuntu Server 22.04 LTS)
Programming Language	Java
Build Tool	Maven
Web Server	Apache Tomcat 9
Containerization	Docker
CI/CD Tool	GitHub Actions
Version Control	Git & GitHub
Client Tool	MobaXterm
Client Access	Web Browser

Step-by-Step Implementation

Step 1: Launch EC2 Instance

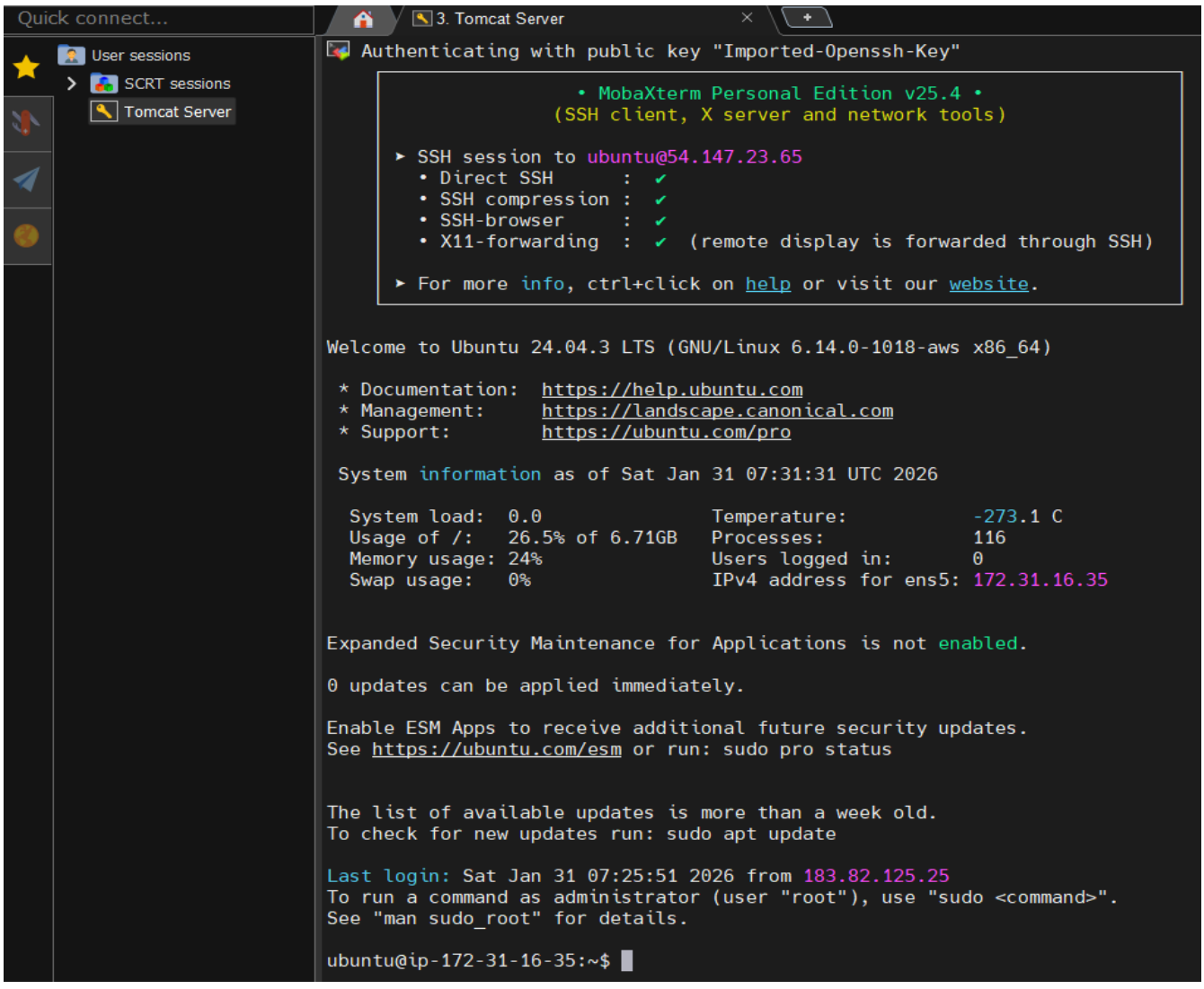
An Amazon EC2 instance was launched to host the Java web application with the following configuration:

- AMI: Ubuntu Server 22.04 LTS
- Instance Type: t3.micro
- Security Group Rules:
  - SSH: Port 22 (for remote access)
  - HTTP: Port 80 (for web traffic)
  - Custom TCP: Port 8080 (for Tomcat application access)



Step 2: Connect to EC2 Instance

- The EC2 instance was accessed securely using MobaXterm with the following details:
  - Public IP of EC2
  - Username: ubuntu
  - Authentication using the EC2 key pair (.pem file)

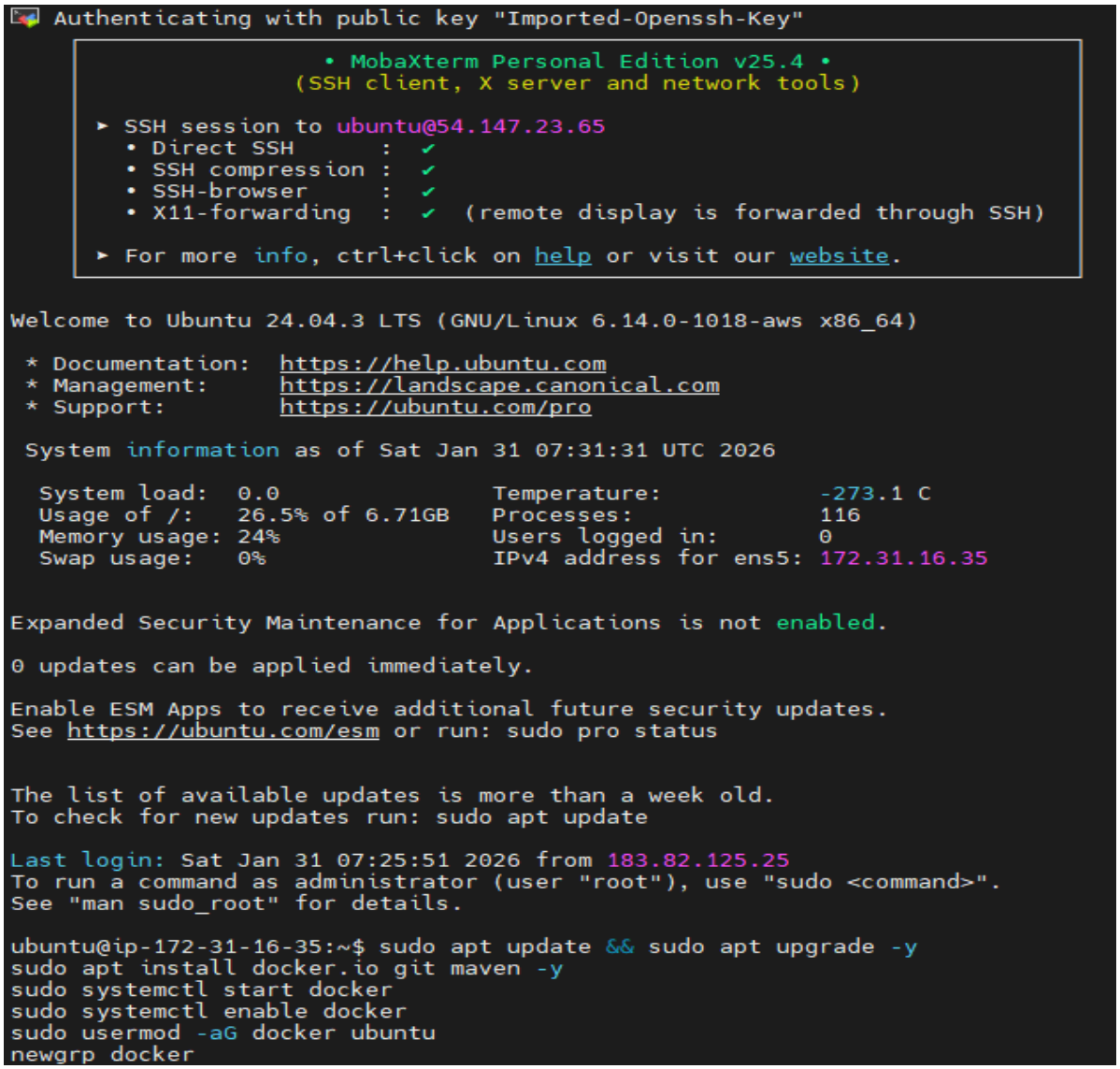


Step 3: Install Required Software

The necessary software dependencies were installed on the EC2 instance to support application deployment:

- Docker
- Git
- Maven

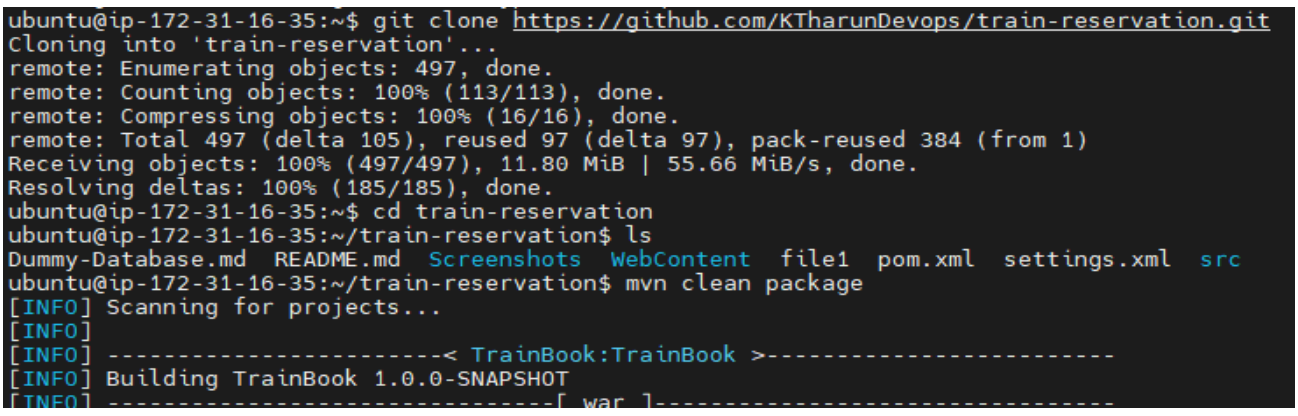
The Docker service was enabled and verified to ensure container operations function correctly.



Step 4: Clone Java Project from GitHub

The Java web application source code was cloned from a GitHub repository using Git.

The project structure was reviewed to confirm all required files were present before building the application.



Step 5: Build WAR File

Maven was used to compile and package the Java application.

The build process completed successfully, generating a **WAR file** in the target directory.

```
r-8.0.30.2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/com/github/jsimone/webapp-runner/8.0.30.2/webapp-runner
-8.0.30.2.jar (9.1 MB at 51 MB/s)
[INFO] Copying webapp-runner-8.0.30.2.jar to /home/ubuntu/train-reservation/target/dependency/webapp-runner.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  21.085 s
[INFO] Finished at: 2026-01-31T07:36:46Z
[INFO] ------------------------------------------------------------------------
ubuntu@ip-172-31-16-35:~/train-reservation$ ls target
TrainBook-1.0.0-SNAPSHOT      classes     generated-sources  maven-status
TrainBook-1.0.0-SNAPSHOT.war  dependency  maven-archiver
ubuntu@ip-172-31-16-35:~/train-reservation$ FROM tomcat:9.0
RUN rm -rf /usr/local/tomcat/webapps/*
COPY target/TrainBook-1.0.0-SNAPSHOT.war /usr/local/tomcat/webapps/
EXPOSE 8080
CMD ["catalina.sh", "run"]
FROM: command not found
RUN: command not found
COPY: command not found
EXPOSE: command not found
CMD: command not found
```

Step 6: Create Dockerfile

A Dockerfile was created to:

- Use Apache Tomcat as the base image
- Copy the generated WAR file into the Tomcat webapps directory
- Configure the container for application deployment

```
ubuntu@ip-172-31-16-35:~/train-reservation$ sudo nano Dockerfile
ubuntu@ip-172-31-16-35:~/train-reservation$ ls
Dockerfile  Dummy-Database.md  README.md  Screenshots  WebContent  file1  pom.xml  settings.xml  src  target
ubuntu@ip-172-31-16-35:~/train-reservation$ docker build -t train-reservation-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  57.29MB
Step 1/5 : FROM tomcat:9.0
9.0: Pulling from library/tomcat
a3629ac5b9f4: Pulling fs layer
bd41d65c3828: Pulling fs layer
```

Step 7: Build Docker Image

Using the Dockerfile, a custom Docker image was built successfully.

This image contained the Tomcat server along with the deployed Java application.

```
root@ip-172-31-34-121:~/train-reservation# ls
Dummy-Database.md  README.md  Screenshots  WebContent  file1  pom.xml  settings.xml  src  target
root@ip-172-31-34-121:~/train-reservation# sudo nano dockerfile
root@ip-172-31-34-121:~/train-reservation# ls
Dummy-Database.md  README.md  Screenshots  WebContent  dockerfile  file1  pom.xml  settings.xml  src  target
root@ip-172-31-34-121:~/train-reservation# docker build -t train-reservation-app .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  57.29MB
Step 1/5 : FROM tomcat:9.0
9.0: Pulling from library/tomcat
a3629ac5b9f4: Pulling fs layer
bd41d65c3828: Pulling fs layer
8be760c2a9a8: Pulling fs layer
4f4fb700ef54: Pulling fs layer
d0702afe109d: Pulling fs layer
c1c5ef39cba9: Pulling fs layer
47177afe6d41: Pulling fs layer
c1c5ef39cba9: Waiting
```

Step 8: Run Docker Container

A Docker container was launched from the built image.

The Tomcat server started successfully and began listening on **port 8080**.

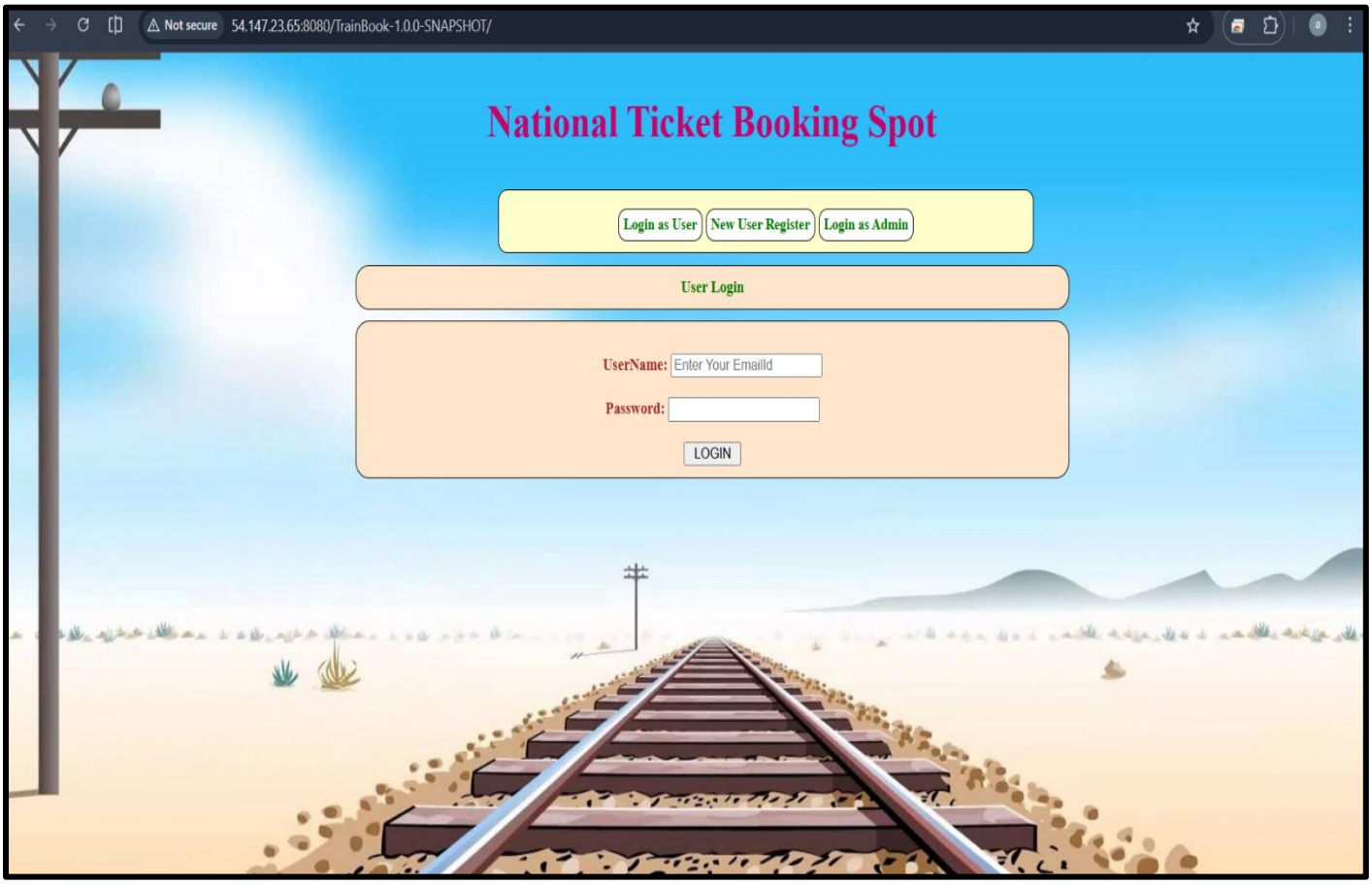
```
d0702afe109d: Pull complete
c1c5ef39cba9: Pull complete
47177afe6d41: Pull complete
Digest: sha256:3d200773e3c53855cd349cfd4785aa3662f496badfc6f8b419a33db6084f6784
Status: Downloaded newer image for tomcat:9.0
 ---> 9bdfa3783498
Step 2/5 : RUN rm -rf /usr/local/tomcat/webapps/*
 ---> Running in 973dbf07a2d8
 ---> Removed intermediate container 973dbf07a2d8
 ---> d8665ae1e069
Step 3/5 : COPY target/TrainBook-1.0.0-SNAPSHOT.war /usr/local/tomcat/webapps/
 ---> a6b1d1d13bf0
Step 4/5 : EXPOSE 8080
 ---> Running in 84c47837e6c2
 ---> Removed intermediate container 84c47837e6c2
 ---> 6f8a3684e458
Step 5/5 : CMD ["catalina.sh", "run"]
 ---> Running in 035d4f96c1d2
 ---> Removed intermediate container 035d4f96c1d2
 ---> 4488520719f3
Successfully built 4488520719f3
Successfully tagged train-reservation-app:latest
root@ip-172-31-34-121:~/train-reservation# docker run -d -p 8080:8080 --name train-reservation-container train-reserva
78b39b47692de3ff77417e8fb0558a9c7289d686e4fbbd228b8e3d3db6c62a16
root@ip-172-31-34-121:~/train-reservation# docker ps
CONTAINER ID   IMAGE                 COMMAND             CREATED          STATUS          PORTS
78b39b47692d   train-reservation-app   "catalina.sh run"   13 seconds ago   Up 12 seconds   0.0.0.0:8080->8080/tcp, [
root@ip-172-31-34-121:~/train-reservation#
```

Step 9: Access Application

The deployed application was accessed through a web browser using the following URL:

http://<EC2_PUBLIC_IP>:8080/TrainBook-1.0.0-SNAPSHOT

The application loaded successfully, confirming proper deployment.



Testing Validation

The deployment was validated using the following checks:

- EC2 instance running without issues
- Docker container in a running state
- Tomcat server responding on port 8080
- WAR file deployed correctly inside the container
- Application accessible via browser

Project Outcome

- Successfully deployed a Java web application on AWS EC2
- Implemented containerized deployment using Docker and Tomcat
- Followed DevOps best practices for application packaging and deployment
- Gained practical hands-on experience with cloud infrastructure and containerization

Conclusion

This project demonstrates the successful end-to-end deployment of a Java web application using AWS EC2, Docker, Maven, and Apache Tomcat. It covers the complete process from cloud infrastructure setup to application deployment and access. Maven was used to build the application, while Docker ensured a consistent and portable deployment environment. Apache Tomcat served as the application server for hosting the WAR file. The project highlights practical DevOps concepts such as containerization, cloud computing, and application lifecycle management. It also reflects real-world deployment practices followed in the industry. Overall, this project is suitable for academic submission, capstone evaluation, and professional portfolio showcasing.

By -
R. Adithi
Aspiring DevOps Engineer
Batch - 06