



Project-04

Managing Web Application Releases Across Different Environments Using GitHub Actions

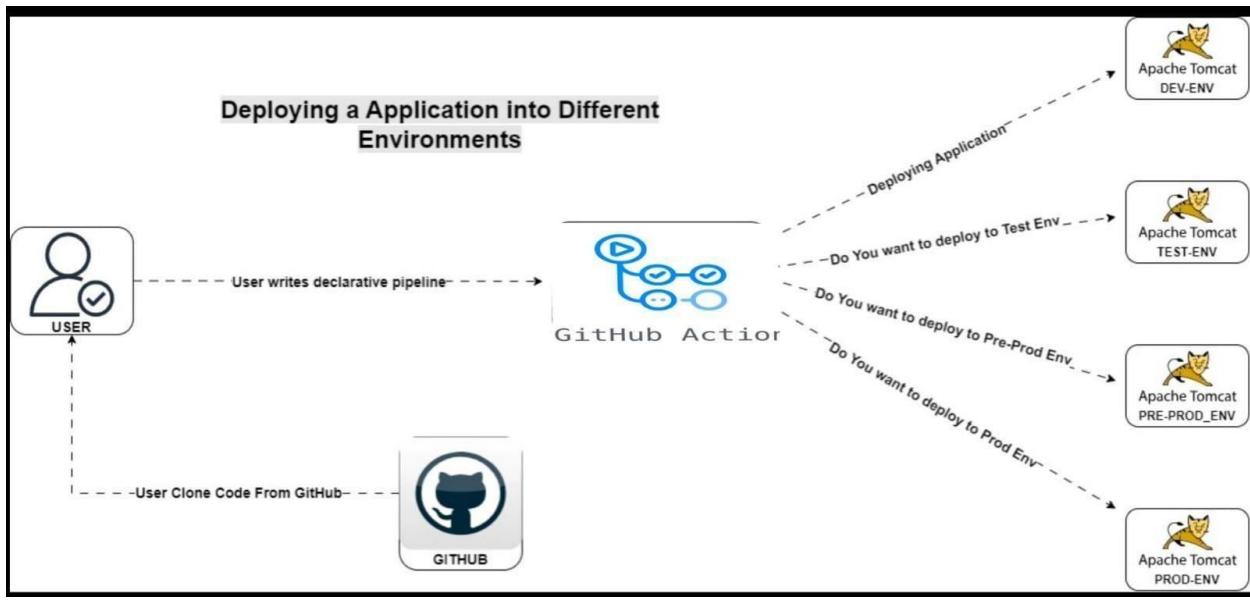


Fig: 1 Hosting Web Application in Apache Tomcat

Introduction to the Project:

This project illustrates the automation of web application deployment across multiple environments—namely development, testing, pre-production, and production—by leveraging **GitHub Actions**. The primary objective is to establish a consistent and dependable CI/CD pipeline seamlessly integrated with **Apache Tomcat servers**.

Project Objectives:

- Implement automated deployment pipelines through GitHub Actions.
- Facilitate deployment across multiple environments (Development, Testing, Pre-Production, Production) driven by user configuration.
- Reduce human intervention errors while improving the speed and efficiency of deployment cycles.

Provisioning AWS EC2 for Deployment

The project uses four Amazon EC2 machines, each running Apache Tomcat, to handle different environments.

- **DEV** – Development server

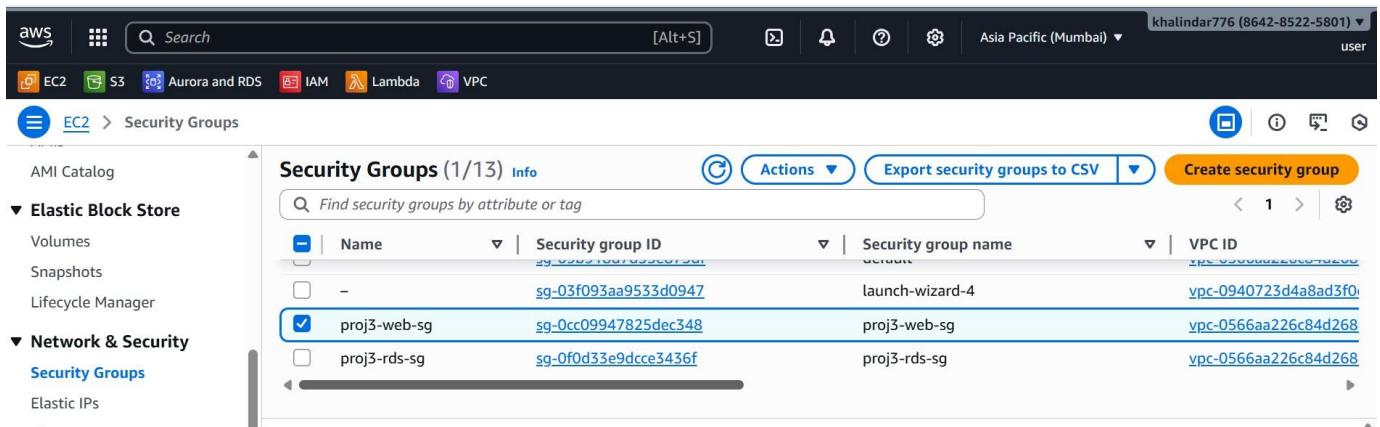
- **TEST** – Testing server
- **PRE-PROD** – Pre-Production server
- **PROD** – Production server

Apache Tomcat was installed on the development server, and an AMI was built from it to quickly set up the test, pre-production, and production servers.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with 'EC2' selected. The main area displays a table of instances. One instance is highlighted: 'dev-server' (i-09c9154c5a5af9814). The instance is 'Running' in a 'c7i-flex.large' type. Below the table, a detailed view for 'i-09c9154c5a5af9814 (dev-server)' is shown, with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. Under 'Instance summary', it shows the Instance ID, Public IPv4 address, and Private IPv4 address.

Fig: 2 Dev-Tomcat Instance

- Go to the AWS EC2 Console and click **Launch Instance**.
- Name the server **Dev-Tomcat** and pick **Ubuntu** as the AMI.
- Use **t2.micro** for demo setups. For real use, choose **t3.small** or **t3.medium**. Create or pick a key pair and save it safely.
- In the network settings, use the default or custom VPC, pick a subnet, and turn on auto public IP. Allow **Port 22** for SSH and **Port 8080** for Tomcat in the security group.



Installation of Java & Apache Tomcat

To configure the Dev-Tomcat instance, connect to it through Git Bash and first become the root user by typing **sudo -i**

Then update the server packages with **apt update -y** and install Java using **apt install openjdk-11-jdk -y**.

You can confirm the installation by running **java -version**. Next, download Apache Tomcat 9 with **wget**

<https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.112/bin/apachetomcat9.0.112.tar.gz>

Check if the file has been downloaded using **ls**.

Extract the archive by executing **tar -xf apache-tomcat-9.0.112.tar.gz** then rename the folder with **mv apache-tomcat-9.0.112 tomcat**.

Move into the Tomcat directory using **cd tomcat/** and then into the bin folder with **cd bin/**.

Start the Tomcat server by running **./startup.sh** and stop it when required using **./shutdown.sh**.

Before updating the tomcat-users.xml file, comment out or edit the access restriction lines in the **manager** and **host-manager** configuration files. Inside the conf directory, add the following roles and users:

```
<role rolename="manager-gui"/> <user username="tomcat" password="s3cret" roles="manager-gui"/> <role rolename="managerscript"/> <user username="deployer" password="deployer" roles="managerscript"/>
```

Here, the rolename defines the type of access (like manager-gui or managerscript), while the user element specifies the username, password, and the roles assigned to that user.

You can customize these values to match your own security requirements, ensuring that each user has the correct level of access to Tomcat's management tools.

You can start the Tomcat server from the bin directory by running `./startup.sh`. Once the server is running, copy the public IP address of the Dev-Tomcat instance and paste it into a browser to confirm that Tomcat has been successfully installed on the Development server.

After completing the setup of the DEV instance, clone the application code from GitHub inside the server by executing `git clone <repository-url>`.

Once the DEV server is fully configured, you need to create an Amazon Machine Image (AMI) from the instance so that it can be reused to provision the other environments.

To do this, go to the EC2 console, select the DEV instance, choose **Actions → Image and templates → Create Image (AMI)**, and provide a name such as *tomcat-base-image*.

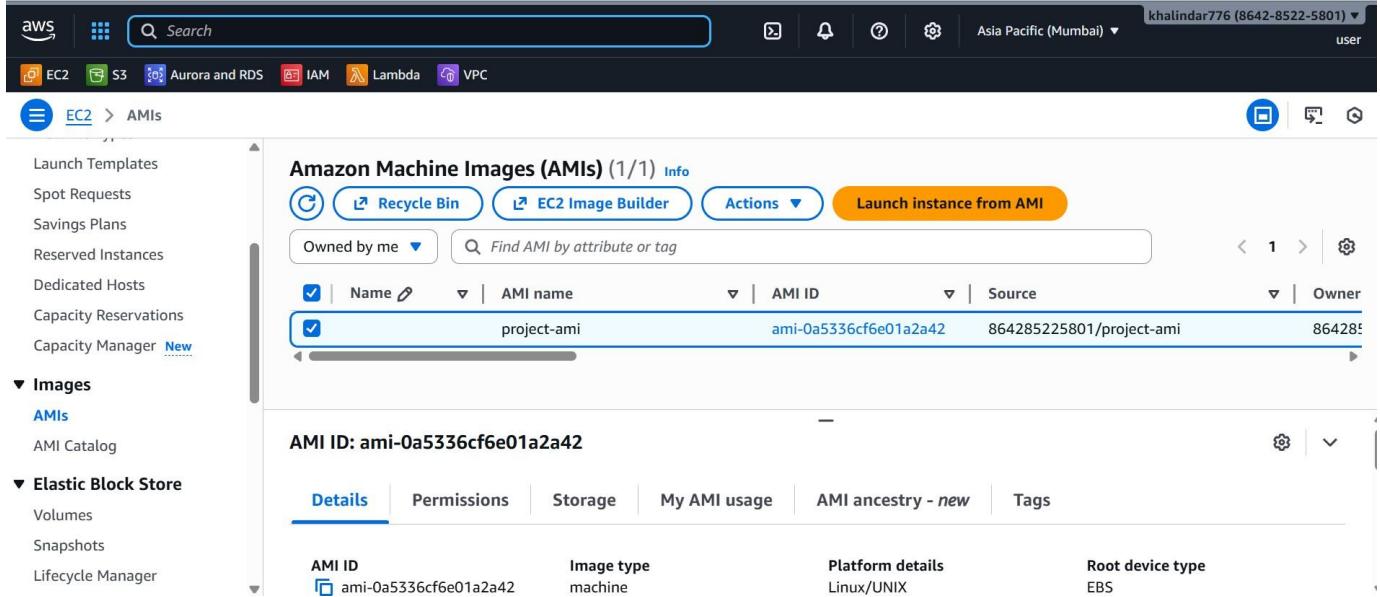
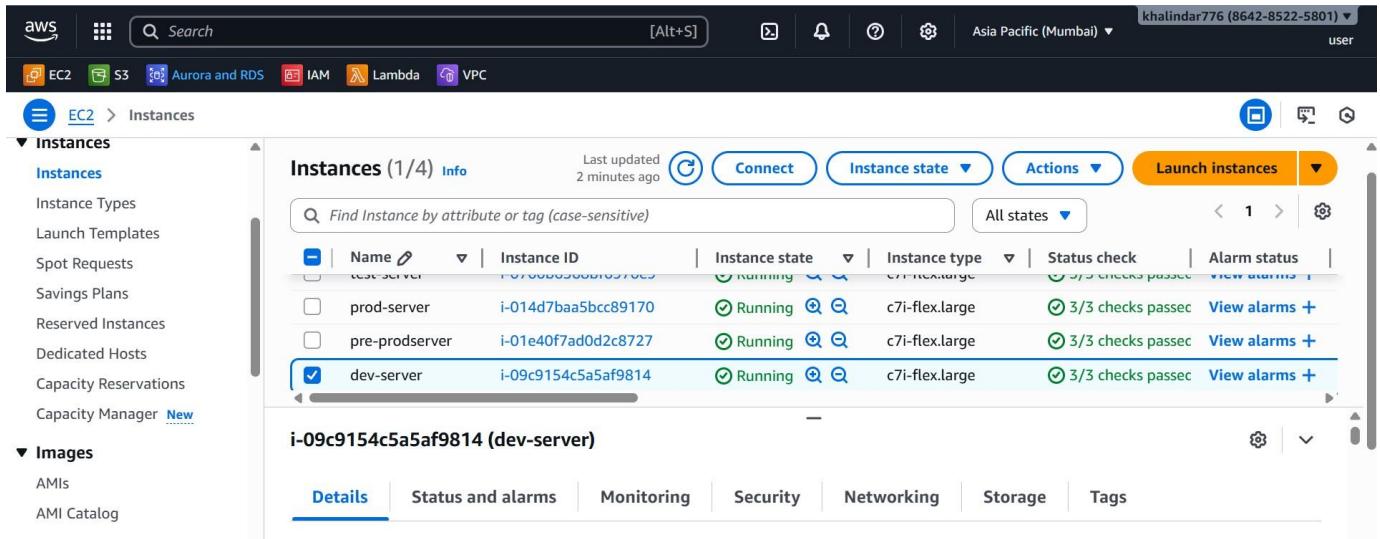


Fig: 3 Amazon Machine Image

After the AMI is created, you can launch three new EC2 instances from it, which will serve as the TEST, PRE-PROD, and PROD servers.

By using the same Amazon Machine Image, each of these servers inherits the identical Java installation, Apache Tomcat configuration, user roles, and folder structure from the DEV instance, ensuring consistency across all stages of deployment and eliminating the need for repetitive setup or reconfiguration.



The screenshot shows the AWS EC2 Instances page. The top navigation bar includes the AWS logo, search bar, and account information (khalindar776 (8642-8522-5801)). Below the navigation are links for EC2, S3, Aurora and RDS, IAM, Lambda, and VPC. The main content area has a sidebar with 'Instances' and 'Images' sections. The 'Instances' section shows a table of four instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
test-server	i-070000000000000000	Running	c7i-flex.large	3/3 checks passed	View alarms
prod-server	i-014d7baa5bcc89170	Running	c7i-flex.large	3/3 checks passed	View alarms
pre-prodserver	i-01e40f7ad0d2c8727	Running	c7i-flex.large	3/3 checks passed	View alarms
dev-server	i-09c9154c5a5af9814	Running	c7i-flex.large	3/3 checks passed	View alarms

The 'dev-server' instance is selected. Below the table, a modal window titled 'i-09c9154c5a5af9814 (dev-server)' is open, showing tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The 'Details' tab is selected.

Fig: 3(i) Launched Instances of Test, Pre-Prod, and Prod

CI/CD Pipeline Setup with GitHub Actions

In GitHub Actions, four deployment environments were defined—DEV, TEST, PRE-PROD, and PROD—and each of these environments was mapped to its corresponding Apache Tomcat server hosted on AWS EC2 by navigating to *Repository → Settings → Environments → New Environment*.

For the deployment workflow, the DEV, TEST, and PRE-PROD environments were configured to run automatically without requiring any manual approval, while the PROD environment was protected with a rule that enforces at least one manual approval before deployment can proceed.

The screenshot shows the GitHub settings interface for a repository named "train-ticket-reservation". The left sidebar has a "General" section selected under "Environments". The main area is titled "Environments" and lists four environments: PROD, PRE_PROD, TEST, and DEV. Each environment has a delete icon to its right. A "New environment" button is located at the top right of the list.

Fig: 4 Configuring Environments

The screenshot shows the GitHub settings interface for the "TEST" environment. The left sidebar has a "General" section selected under "Environments". The main area is titled "Environments / Configure TEST" and shows "Deployment protection rules". It includes sections for "Required reviewers" (checked), "Add up to 5 more reviewers" (with a search bar and a team entry for "DADAKHALINDAR"), "Prevent self-review" (unchecked), and "Wait timer" (unchecked).

Fig: 4(i) Manual Approval for PROD Environment

Environment Credentials and Variables in GitHub

In GitHub Actions, environment secrets and variables are created to securely store the connection details required for each Apache Tomcat server running on AWS EC2 instances.

These secrets are defined within the repository settings under *Environments*, ensuring that sensitive information such as server credentials, IP addresses, and configuration parameters are protected while still being accessible to the deployment workflows.

By setting up environment-specific secrets, the DEV, TEST, PRE-PROD, and PROD servers can be provisioned consistently without exposing critical data directly in the pipeline.

- **DEV Environment** ◦
TOMCAT_HOST_DEV ◦ TOMCAT_USER_DEV ◦
TOMCAT_PASSWORD_DEV
- **TEST Environment** ◦
TOMCAT_HOST_TEST ◦ TOMCAT_USER_TEST ◦
TOMCAT_PASSWORD_TEST
- **PRE_PROD Environment** ◦ TOMCAT_HOST_PRE_PROD ◦
TOMCAT_USER_PRE_PROD
◦
TOMCAT_PASSWORD_PRE_PROD
- **PROD Environment** ◦
TOMCAT_HOST_PROD ◦ TOMCAT_USER_PROD ◦
TOMCAT_PASSWORD_PROD

Secrets and variables		
Actions		
Codespaces		
Dependabot		
<hr/>		
Integrations		
GitHub Apps		
Email notifications		

 TOMCAT_PASS_DEV	1 hour ago	 
 TOMCAT_PASS_POD	41 minutes ago	 
 TOMCAT_PASS_PREPOD	44 minutes ago	 
 TOMCAT_PASS_TEST	46 minutes ago	 
 TOMCAT_URL_DEV	11 minutes ago	 
 TOMCAT_URL_POD	42 minutes ago	 
 TOMCAT_URL_PREPOD	8 minutes ago	 
 TOMCAT_URL_TEST	38 minutes ago	 
 TOMCAT_USER_DEV	1 hour ago	 
 TOMCAT_USER_POD	41 minutes ago	 
 TOMCAT_USER_PREPOD	43 minutes ago	 
 TOMCAT_USER_TEST	47 minutes ago	 

Fig: 5 Secrets & Variables

After you finish configuring the environment and secret variables in GitHub Actions, the next step is to update the workflow definition file located at `.github/workflows/tomcat.yaml` with the appropriate references to those secrets and environment settings.

Once the file has been modified to reflect the deployment requirements, commit the changes to the repository, which will automatically trigger the pipeline execution and initiate the deployment process.

GitHub Actions Pipeline Script

----- name:
Build and Deploy to Tomcat

on: push: branches:

```
- master # DEV deployment workflow_dispatch: # optional manual trigger  
for PROD if needed
```

jobs:

```
# ----- DEV Deployment ----- deploydev:  
runs-on: ubuntu-latest
```

environment: DEV steps:

```
- uses: actions/checkout@v4 - uses: actions/setup-java@v4 with:  
distribution: 'temurin' java-  
version: '17'  
- run: mvn clean package -DskipTests  
- name: Deploy to DEV Tomcat run:  
curl -v -u ${{ secrets.TOMCAT_USER_DEV }}:${{  
secrets.TOMCAT_PASSWORD_DEV }} \\\n-T target/TrainBook-1.0.0-SNAPSHOT.war \\\n"http://${{ secrets.TOMCAT_HOST_DEV }}:\\  
8080/manager/text/deploy?path=/TrainBook&update=true"
```

```
# ----- TEST Deployment -----  
deploytest: runs-on: ubuntu-latest
```

needs: deploy-dev
environment: TEST
steps:

```

- uses: actions/checkout@v4      - uses: actions/setup-java@v4      with:
    distribution: 'temurin'        java-
version: '17'
- run: mvn clean package -DskipTests
- name: Deploy to TEST Tomcat
  run: |
    curl -v -u ${ secrets.TOMCAT_USER_TEST }:${ secrets.TOMCAT_PASSWORD_TEST } \
    -T target/TrainBook-1.0.0-SNAPSHOT.war \
    "http://${ secrets.TOMCAT_HOST_TEST }:8080/manager/text/deploy?path=/TrainBook&update=true"

# ----- PRE-PROD Deployment ----- deploy-
preprod: runs-on: ubuntu-latest needs: deploy-test
environment: PRE-PROD steps:
- uses: actions/checkout@v4      - uses: actions/setup-java@v4      with:
    distribution: 'temurin'        java-
version: '17'
- run: mvn clean package -DskipTests      - name: Deploy to PRE-PROD
  Tomcat      run: |
    curl -v -u ${ secrets.TOMCAT_USER_PRE_PROD }:${ secrets.TOMCAT_PASSWORD_PRE_PROD } \
    -T target/TrainBook-1.0.0-SNAPSHOT.war \
    "http://${ secrets.TOMCAT_HOST_PRE_PROD }:8080/manager/text/deploy?path=/TrainBook&update=true"

# ----- PROD Deployment (Manual Approval) -----
deploy-prod: runs-on: ubuntu-latest needs: deploy-preprod if:
github.ref ==
'refs/heads/master' environment:
name: PROD
  # Make sure this environment has "Required reviewers" enabled
steps:
- uses: actions/checkout@v4      - uses:
  actions/setup-java@v4      with:

```

```

distribution: 'temurin'      java-
version: '17'
- run: mvn clean package -DskipTests
- name: Deploy to PROD Tomcat    run: |
  curl -v -u ${ secrets.TOMCAT_USER_PROD }:${ secrets.TOMCAT_PASSWORD_PROD } \
  -T target/TrainBook-1.0.0-SNAPSHOT.war \
  "http://${ secrets.TOMCAT_HOST_PROD }:8080/manager/text/deploy?path=/TrainBook&update=true"

```

After the code was committed, the CI/CD pipeline ran successfully, and the deployments to the DEV, TEST, and PRE-PROD environments were automatically triggered and completed without any manual intervention.

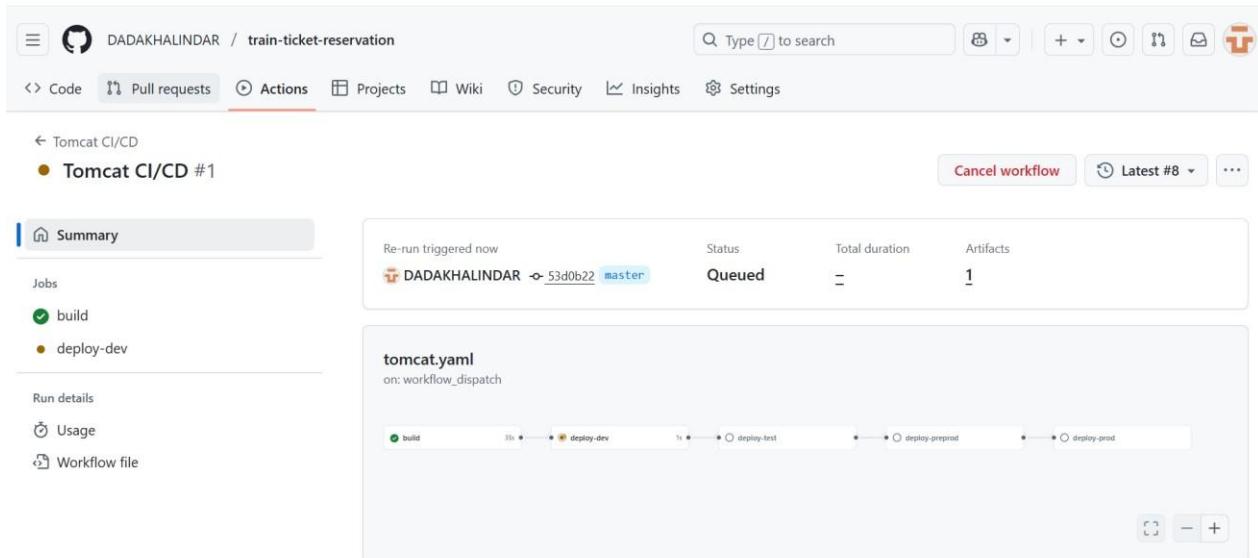


Fig: 6 Deployment Pipeline

The PROD deployment, by design, was set to pause and await manual approval before continuing, thereby enforcing a controlled and secure release management process.

This safeguard ensures that production changes are reviewed and authorized, reducing risk and maintaining stability in the live environment.

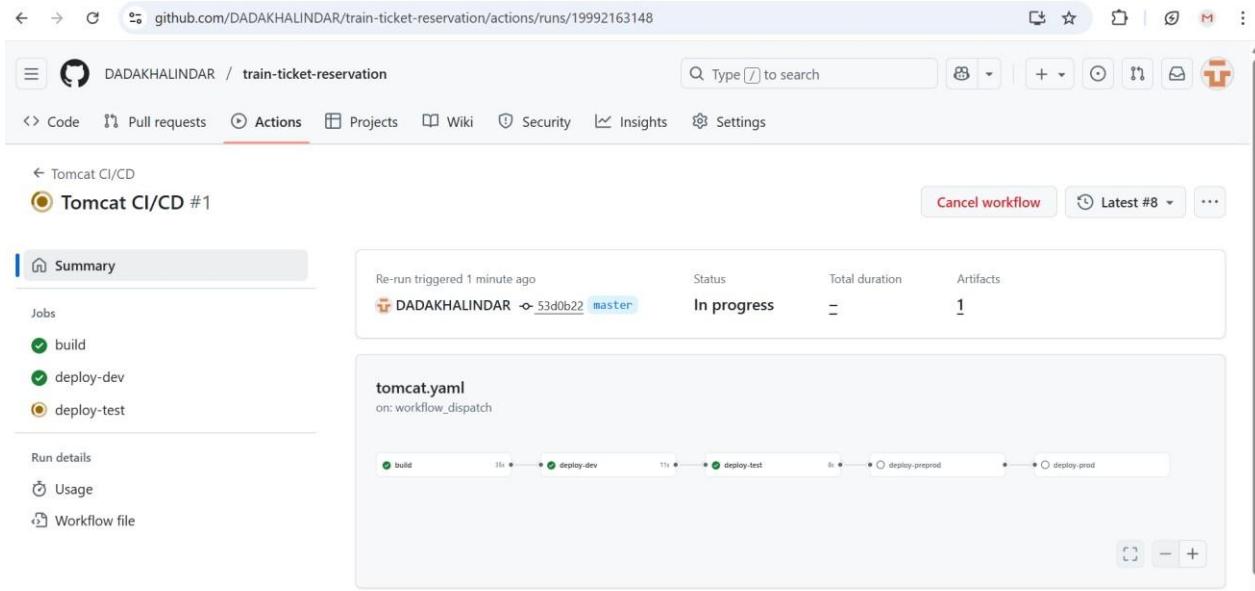


Fig: 6(i) Deployment up to Pre-Prod

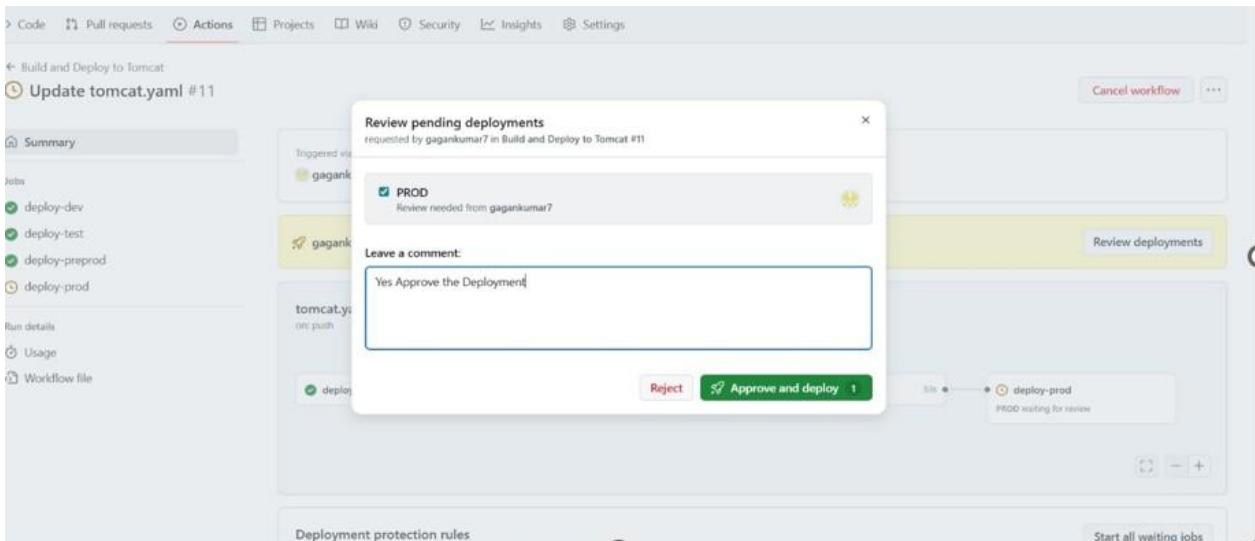


Fig: 6(ii) Deployment Approval to Production Environment

Note: The image above demonstrates how an approval request is presented in GitHub Actions. A comparable approval prompt is displayed in the TEST, PRE-PROD, and PROD environments whenever a manual approval step is enforced, ensuring that deployment does not proceed until the required authorization is granted.

Fig: 6(iii) Web Application Deployed to all Servers

Application Deployed on all the Servers after Approval

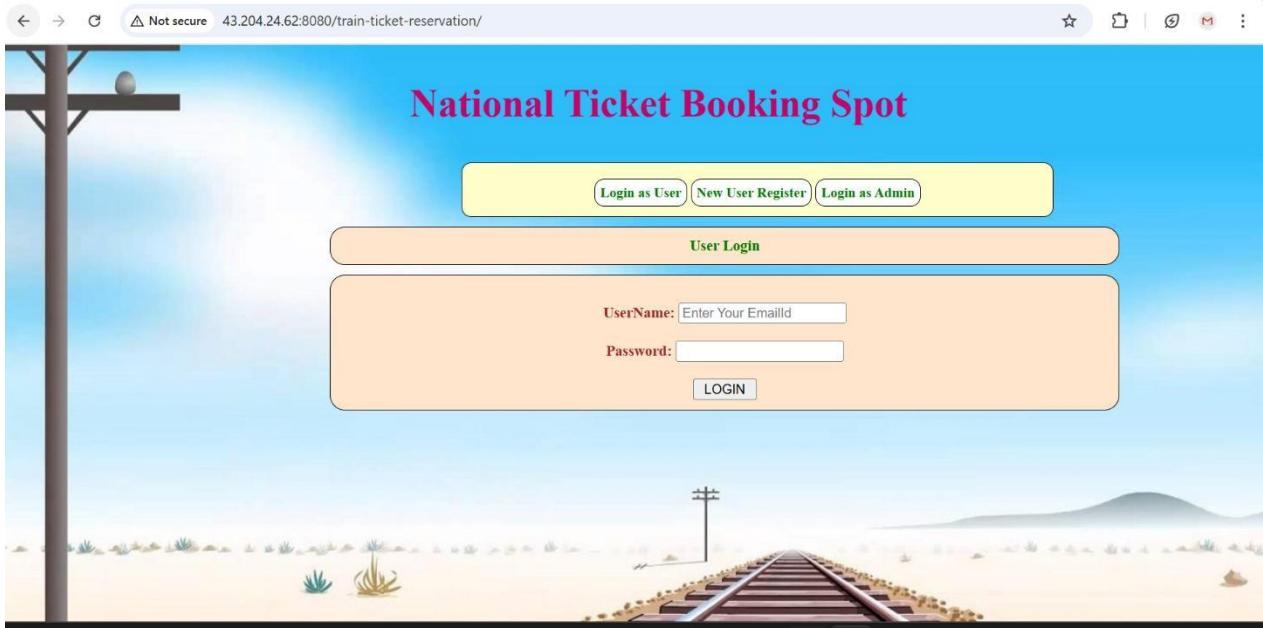
The Java web application was deployed successfully across all four environments— DEV, TEST, PRE-PROD, and PROD—ensuring uniform functionality and consistent configuration on each Apache Tomcat server hosted within AWS EC2.

Fig:

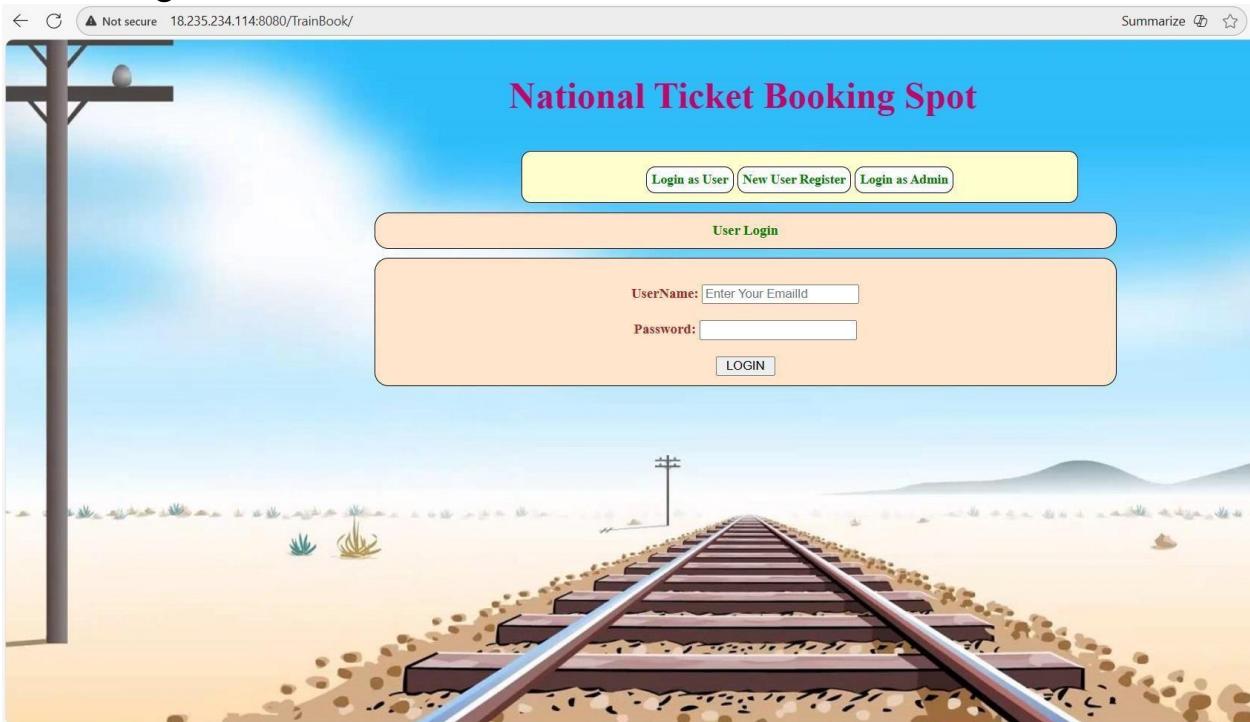
7 Application Deployed into Tomcat

Deployment Workflow Across Environments

Deployment to the **DEV environment** is configured to run automatically without requiring any manual approval.

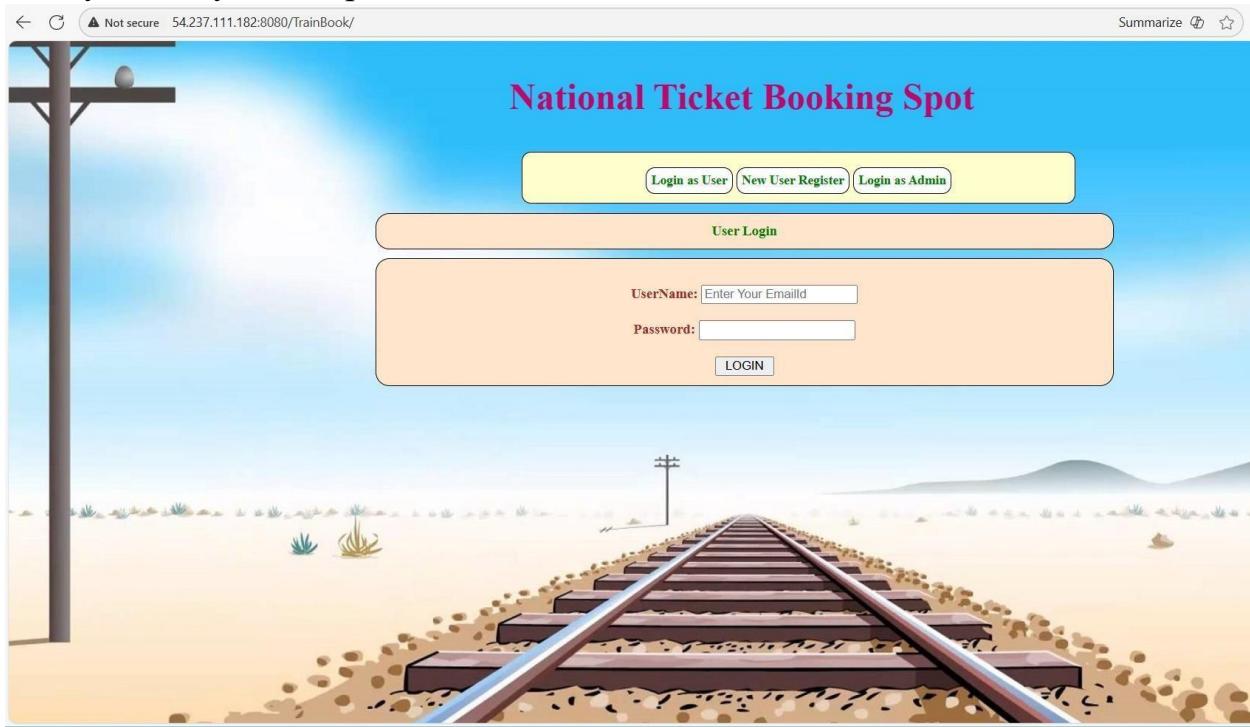


TEST Environment-Deployment to the TEST environment runs automatically without requiring manual approval, enabling quick validation of new changes.

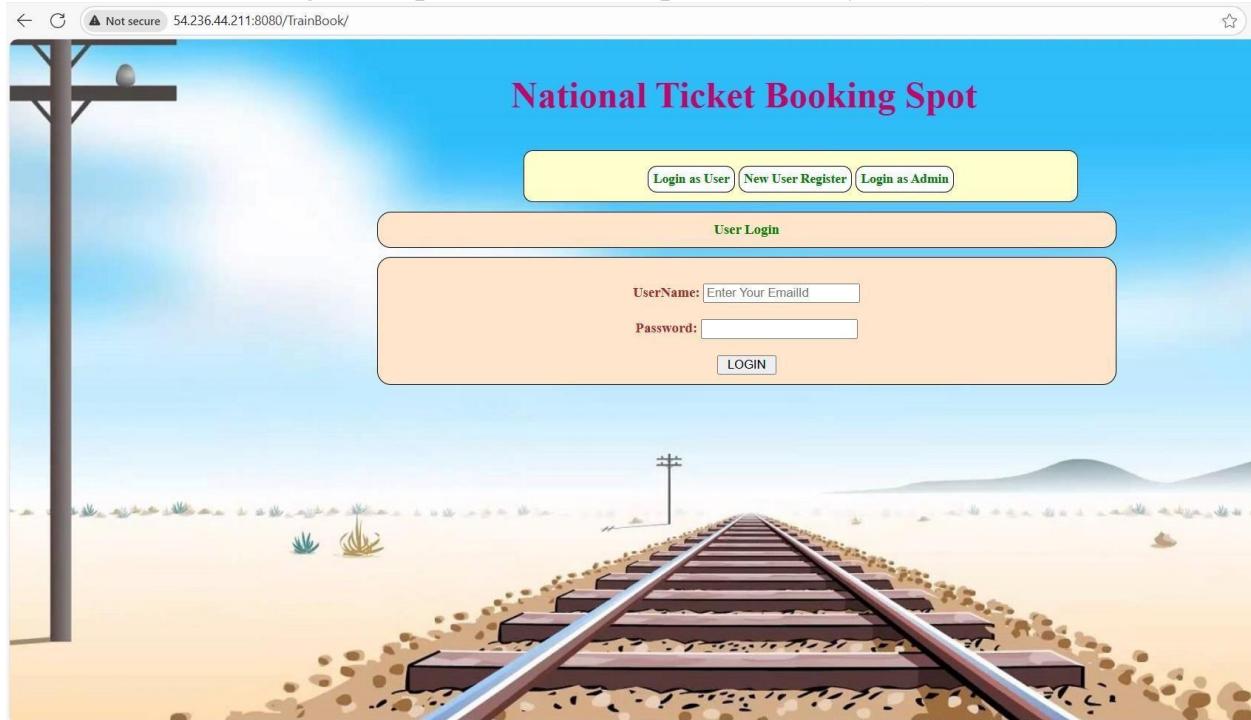


PRE-PROD Environment-Deployment to the PRE-PROD environment also runs automatically without requiring manual approval, allowing teams

to verify stability before production release.



PROD Environment-Deployment to the PROD environment is gated by a **manual approval process**. This safeguard ensures that only reviewed and validated code changes are promoted to the production system.



Project Conclusion

When a developer commits or pushes code to GitHub, the **GitHub Actions workflow** defined in **tomcat.yaml** is automatically triggered. The developer then selects the target environment—DEV, TEST, PRE-PROD, or PROD.

The **CI pipeline** compiles the application with Maven and generates a .war file, while the **CD pipeline** securely transfers this artifact to the designated Apache Tomcat server hosted on AWS EC2.

Tomcat subsequently deploys the application from the webapps/ directory. For higher environments, such as PROD, **manual approval steps** are enforced to provide an additional layer of verification, ensuring controlled and secure deployments while preventing accidental releases.

--Thank You--