DevOps Project Report

Scientific Calculator using CI/CD Pipeline with GitHub, Jenkins, Docker, DockerHub, and Ansible

Name: **Vaibhav Bajoriya**
Roll Number: **IMT2022574**
International Institute of Information Technology Bangalore
October 11, 2025

1 Introduction

This project implements the DevOps lifecycle for deploying a Scientific Calculator application written in C++. The process integrates GitHub for version control, Jenkins for CI/CD automation, Docker and DockerHub for containerization, and Ansible for deployment.

The aim is to demonstrate automation of the build, test, and deployment pipeline so that the entire lifecycle can be replicated by beginners.

2 What and Why of DevOps

DevOps is a combination of **Development (Dev)** and **Operations (Ops)** practices. It emphasizes collaboration between software developers and IT operations teams to automate and integrate processes for faster and more reliable software delivery.

Why DevOps?

- **Faster Delivery:** Automates build, test, and deployment.
- **Improved Collaboration:** Breaks the wall between development and operations.
- **Scalability:** Easily scale deployments using tools like Docker and Ansible.
- **Reliability:** Ensures consistent environments through automation.
- **Continuous Feedback:** Rapid detection and fixing of issues.

3 Tools Used

For this project, we used a set of DevOps tools to implement a complete CI/CD pipeline for deploying the Scientific Calculator application. Each tool plays a crucial role in the lifecycle.

1. GitHub (Source Code Management)

- GitHub was used as the version control system to store and manage the source code of the Scientific Calculator.
- Provides collaboration features, commit history, and integration with Jenkins.
- Public link: GitHub - Scientific Calculator

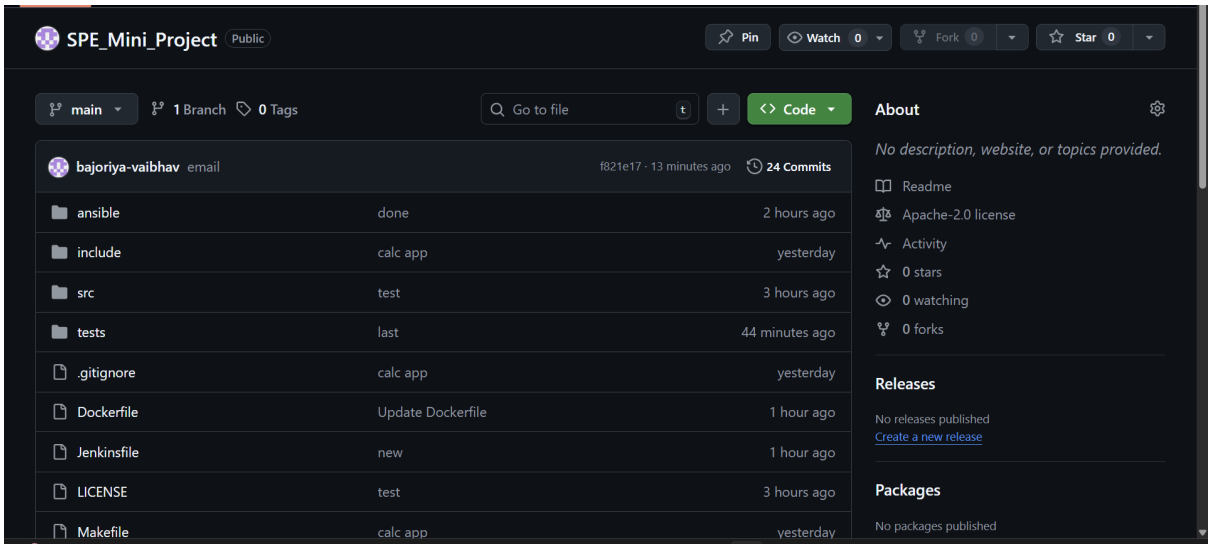

Figure 1: GitHub repository showing project files and commits

2. Jenkins (Continuous Integration / Continuous Deployment)

- Jenkins was used to automate the build, test, and deployment pipeline of the Scientific Calculator project.
- Configured Make for building the C++ project, Docker for containerization, and Ansible for deployment.
- GitHub Webhook was set up so that each commit in the repository triggers the Jenkins pipeline automatically.
- Credentials for GitHub and DockerHub were securely stored in Jenkins using **Manage Credentials**.
- Public Jenkinsfile: Jenkins Pipeline Script

Commands used to start Jenkins in WSL:

```
# Start Jenkins service
sudo service jenkins start

# Check Jenkins status
sudo service jenkins status

# Access Jenkins in browser
http://localhost:8080
```

Pipeline Structure

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "spe_mini_calculator"
        IMAGE_TAG = "latest"
        EMAIL_ID_TO_SEND = "vaibhav.bajoriya@iiitb.ac.in"
    }

    stages { ... }
    post { ... }
}
```

Environment Variables

- `IMAGE_NAME`: Defines the Docker image name
- `IMAGE_TAG`: Version tag for the Docker image
- `EMAIL_ID_TO_SEND`: Email address for build notifications

Stage 1: Checkout

```
stage('Checkout') {
    steps {
        git branch: 'main',
            url: 'https://github.com/bajoriya-vaibhav/SPE_Mini_Project.git'
    }
}
```

Purpose: Clones the latest code from the GitHub repository's main branch.

Stage 2: Build

```
stage('Build') {
    steps {
        sh 'make clean && make'
    }
}
```

Purpose: Compiles the C++ source code using the Makefile. The `make clean` removes old build artifacts, and `make` compiles the calculator application with static linking.

Stage 3: Test

```
stage('Test') {
    steps {
        sh 'make test'
    }
}
```

Purpose: Runs unit tests to verify all calculator operations work correctly. Tests include validation for square root, factorial, natural log, and power functions.

Stage 4: Docker Build & Push

```
stage('Docker Build & Push') {
    steps {
        withCredentials([usernamePassword(
            credentialsId: 'dockerhub-creds',
            usernameVariable: 'DOCKERHUB_USER',
            passwordVariable: 'DOCKERHUB_PASS')]) {
            script {
                def DOCKER_IMAGE = "${DOCKERHUB_USER}/${IMAGE_NAME}:${IMAGE_TAG}"
                sh "DOCKER_BUILDKIT=0 docker build -t $DOCKER_IMAGE ."
                sh """
                    echo $DOCKERHUB_PASS | docker login -u $DOCKERHUB_USER --password
                    docker push $DOCKER_IMAGE
                """
            }
        }
    }
}
```

Purpose:

- Securely retrieves DockerHub credentials from Jenkins
- Builds Docker image using multi-stage Dockerfile
- Disables BuildKit to avoid compatibility issues
- Logs into DockerHub and pushes the image to the registry

Stage 5: Deploy with Ansible

```
stage('Deploy with Ansible') {
    steps {
        sh 'ansible-playbook -i ansible/inventory.ini ansible/playbook.yml'
    }
}
```

Purpose: Executes Ansible playbook to:

- Remove existing containers
- Pull the latest image from DockerHub
- Run the calculator container on localhost

Post Actions

On Success:

```
success {
    emailext(
        to: "${EMAIL_ID_TO_SEND}",
        subject: "SUCCESS: Pipeline ${env.JOB_NAME} #${env.BUILD_NUMBER}",
        body: "Build successful!...",
        mimeType: 'text/html'
    )
}
```

Sends HTML email notification with build details and Jenkins URL.

On Failure: Similar email notification but with failure message and debugging information.

Cleanup:

```
cleanup {
    cleanWs()
}
```

Cleans workspace after pipeline execution to free up disk space.

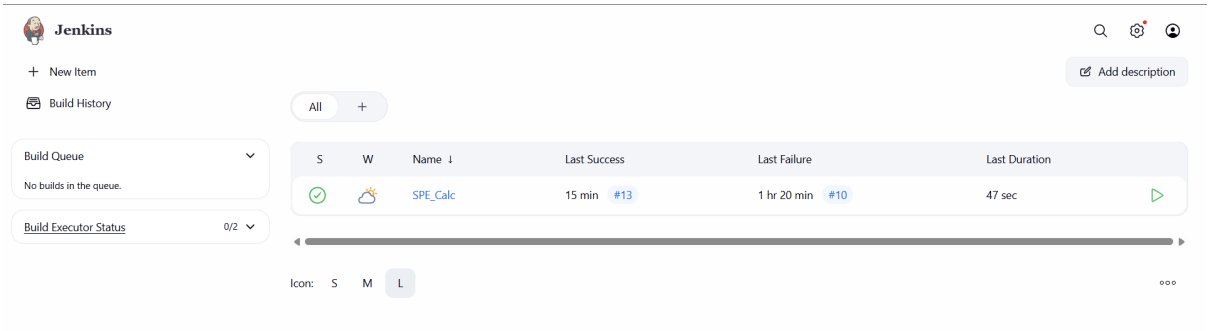

Figure 2: Jenkins dashboard showing the configured Scientific Calculator pipeline

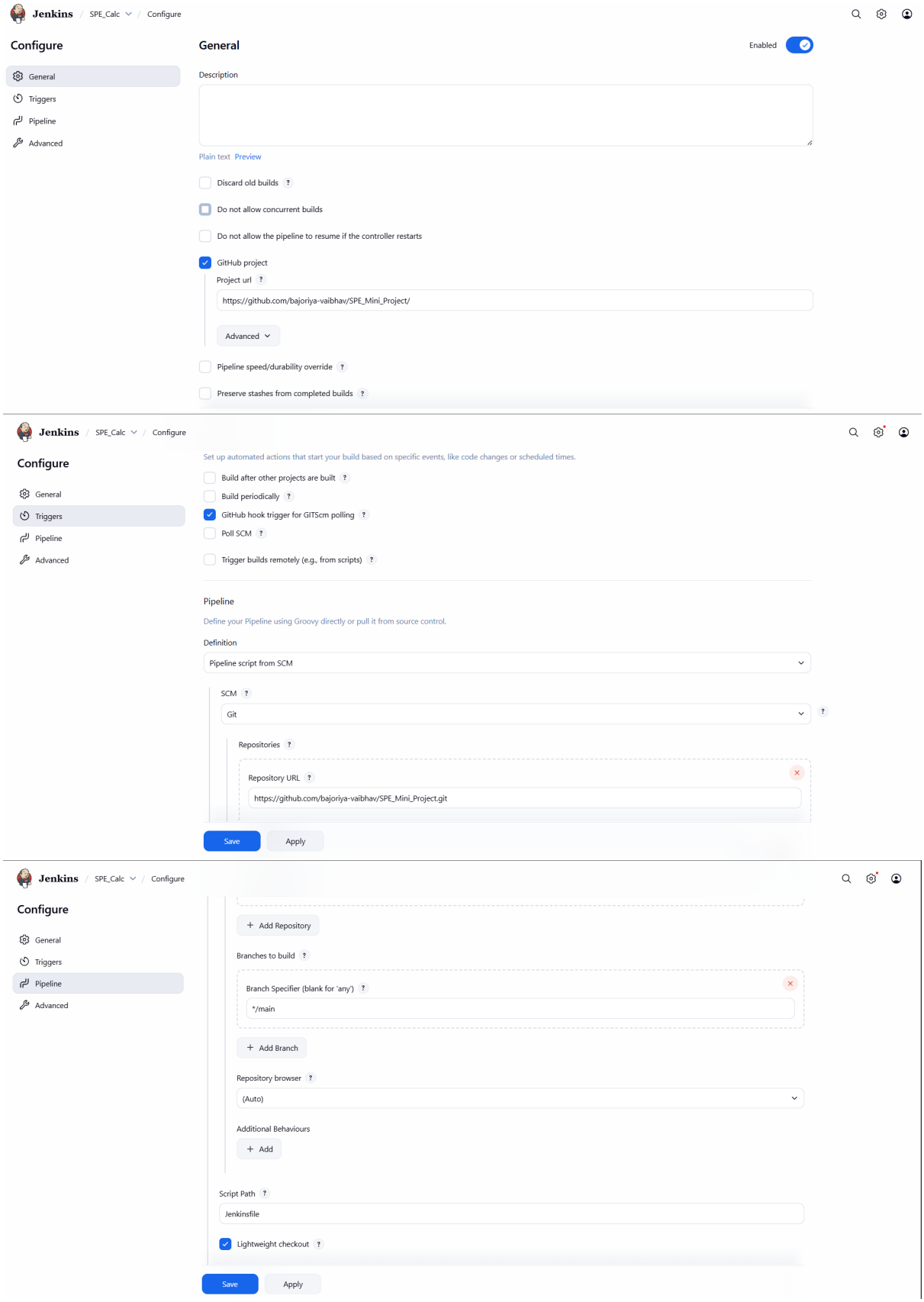

Figure 3: Jenkins pipeline configuration

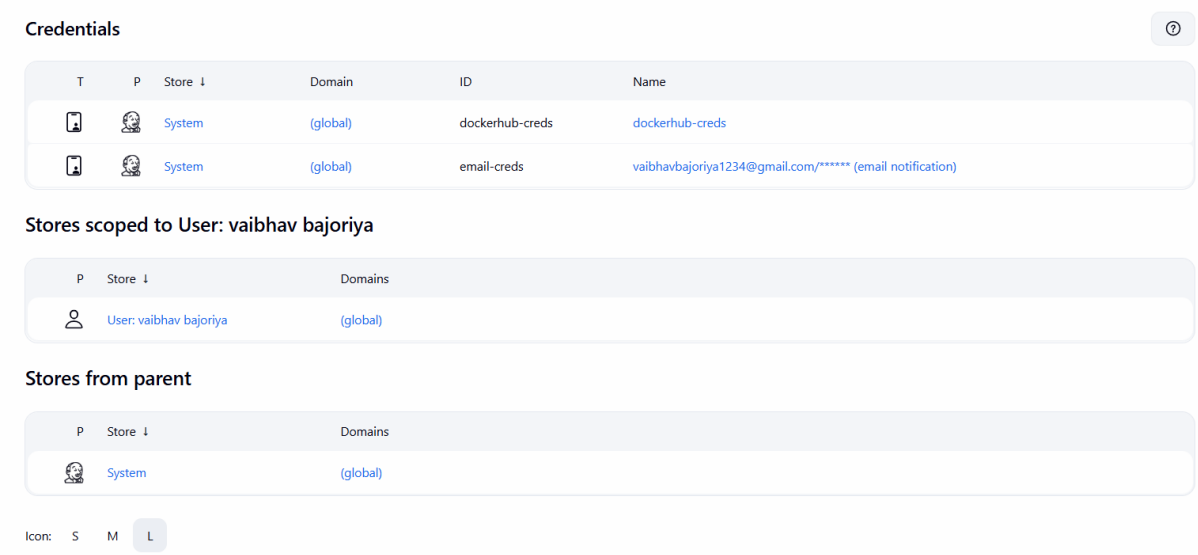

Figure 4: Jenkins Manage Credentials page showing configured credentials

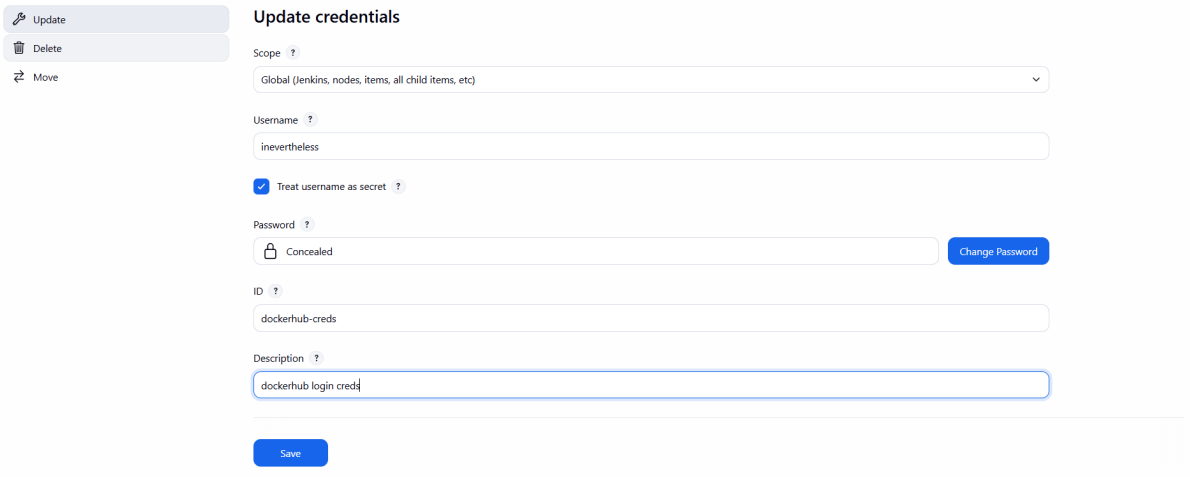

Figure 5: DockerHub credentials securely stored in Jenkins

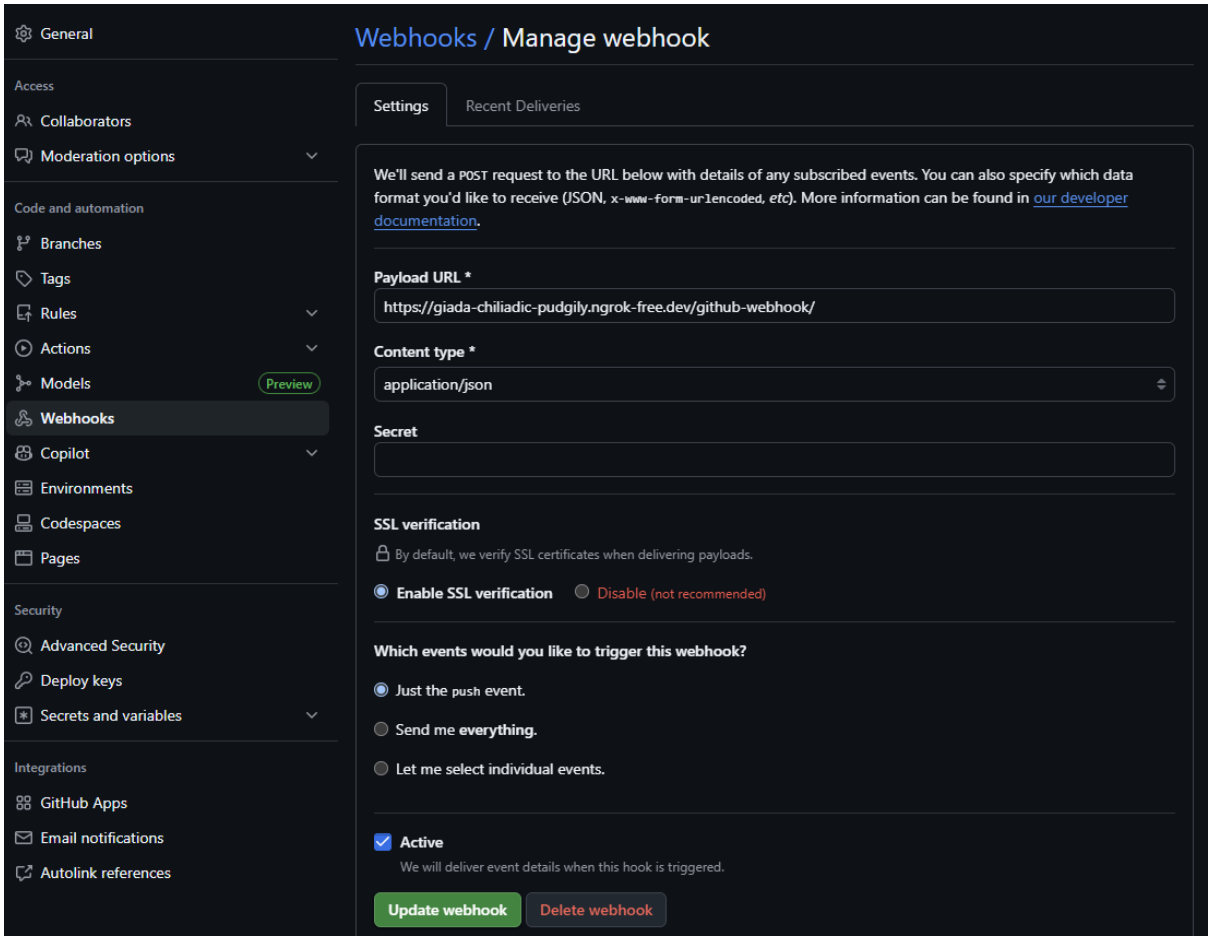

Figure 6: GitHub Webhook configured to trigger Jenkins pipeline on push events

3. Ngrok (Exposing Local Jenkins to GitHub)

- Since Jenkins was running locally on WSL (Windows Subsystem for Linux), it was not directly accessible from the internet.
- To allow GitHub Webhooks to trigger Jenkins jobs, we used **ngrok** to expose the local Jenkins server running at `http://localhost:8080`.
- Ngrok provides a temporary public URL that tunnels requests to the local server, making Jenkins accessible externally.
- The generated forwarding URL was copied and added to the GitHub Webhook configuration in the repository.

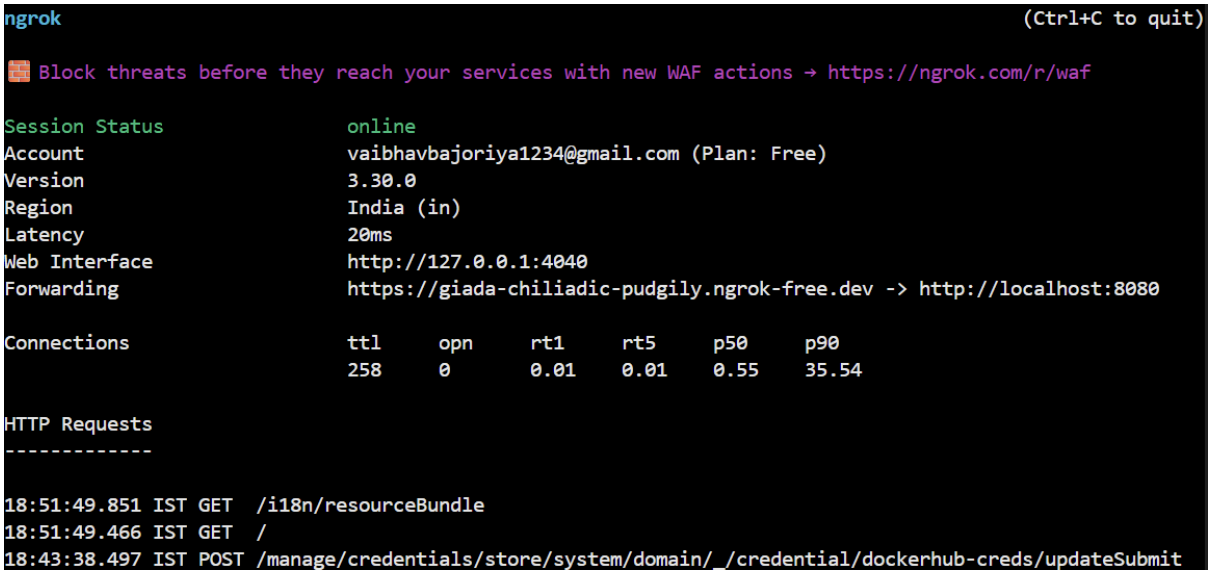

Figure 7: ngrok forwarding URL exposing local Jenkins instance

4. Docker (Containerization)

- Docker was used to containerize the Scientific Calculator application so that it can run in a consistent environment across different systems.
- A `Dockerfile` was written using a multi-stage build:
  - Stage 1: Uses `gcc:latest` image to compile the C++ source code using Make.
  - Stage 2: Creates a minimal container from `scratch` containing only the compiled binary.
- The Jenkins pipeline automatically builds the Docker image after compiling the C++ project using the Makefile.
- Static linking (`-static` flag) ensures the binary is self-contained and doesn't require external libraries at runtime.

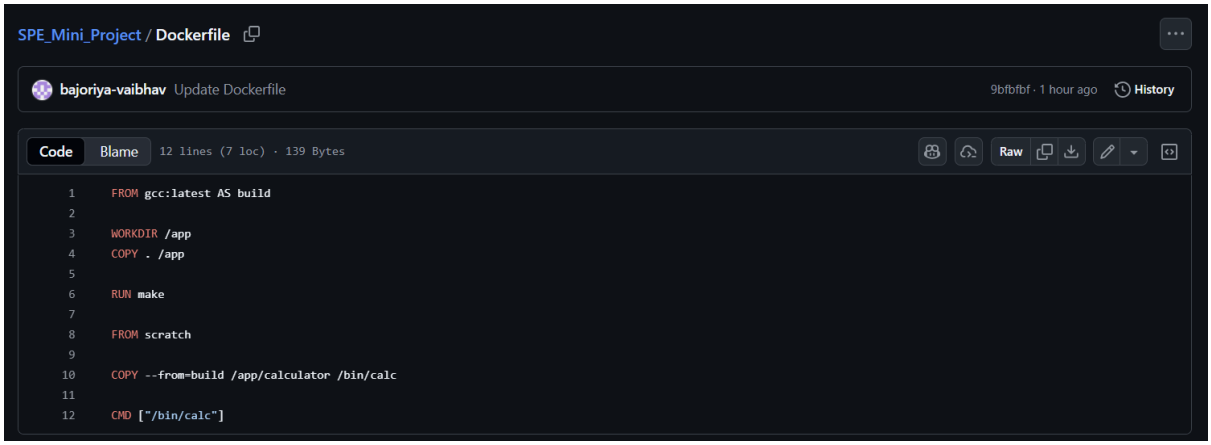

Figure 8: Dockerfile used to containerize the Scientific Calculator application

Docker Hub (Container Registry)

- The built Docker image was tagged with the project name and version and pushed to Docker Hub.
- Docker Hub acts as a central registry, allowing the image to be pulled and run on any server.
- Public Docker Hub Repository: Scientific Calculator on Docker Hub

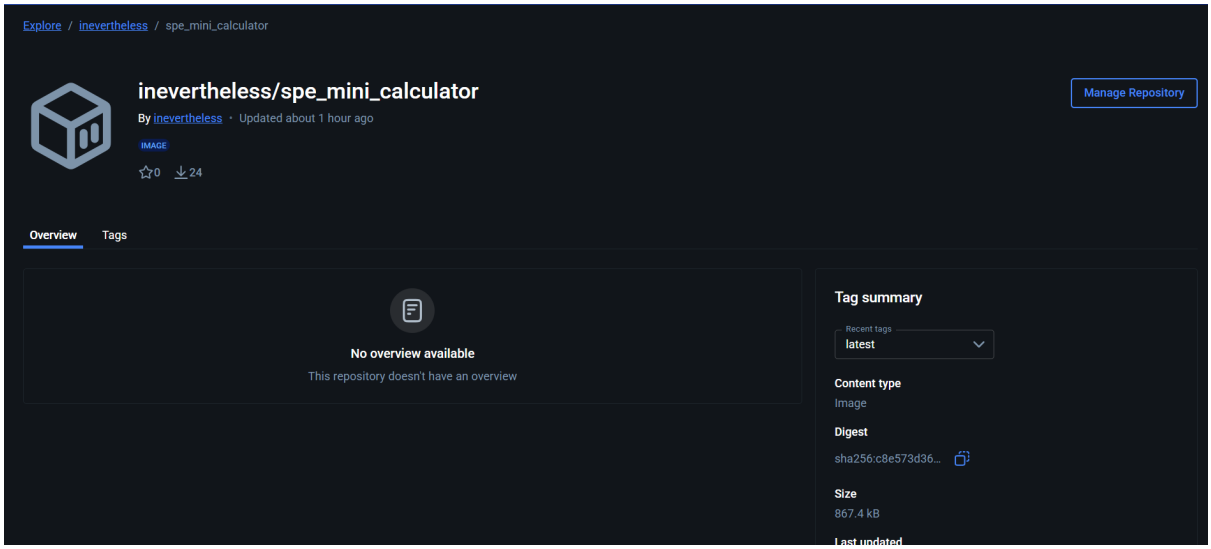

Figure 9: Scientific Calculator Docker image pushed to Docker Hub

5. Deployment (with Ansible)

- Ansible was used to automate the deployment of the Scientific Calculator Docker container on the target host.
- A playbook (`playbook.yml`) was written to:
  - Remove all existing containers to ensure a clean state.
  - Pull the latest Docker image from Docker Hub.
  - Run the container in detached mode, exposing port `8080`.
- The inventory file (`inventory.ini`) specifies localhost as the target.
- This removes manual deployment steps and makes the process repeatable and reliable.
- Jenkins pipeline was configured to trigger the Ansible playbook after pushing the Docker image to Docker Hub.

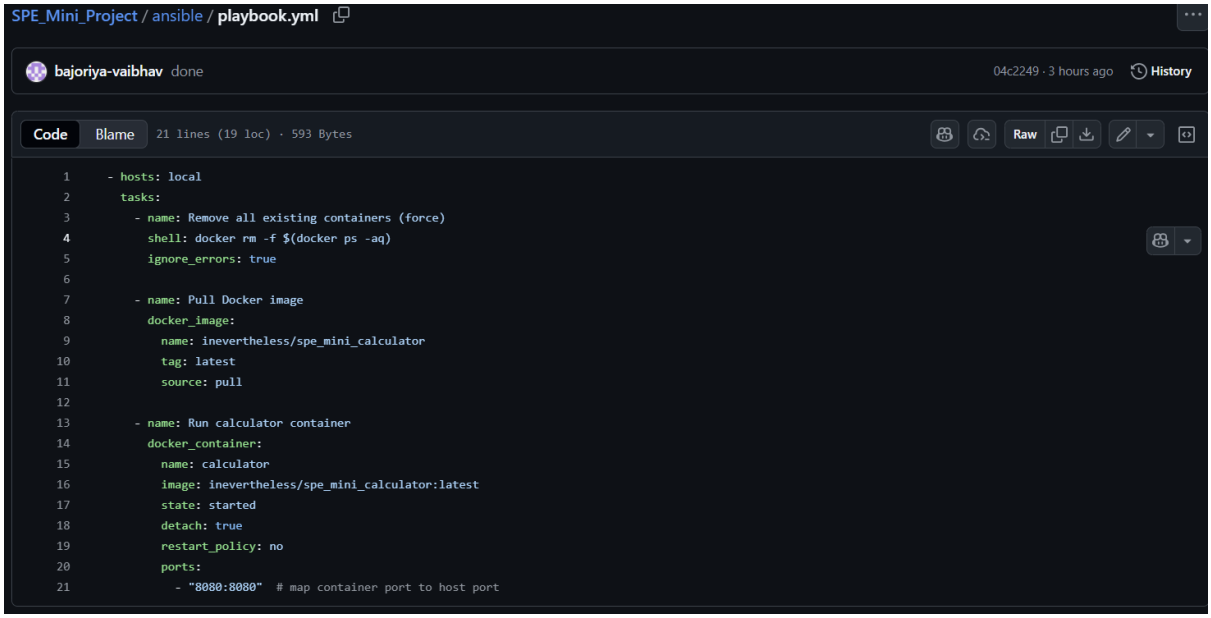

Figure 10: Ansible playbook (`playbook.yml`) for deploying the Docker container

Figure 11: Ansible playbook execution deploying the container on target host

Application Architecture

The Scientific Calculator application is built using C++ with a modular architecture:

- **Header File (`calculator.h`):** Defines the Calculator class with static methods for mathematical operations.
- **Implementation (`calculator.cpp`):** Implements four operations:
  - Square Root with input validation

  - Factorial with iterative calculation
  - Natural Logarithm with domain checking
  - Power function using standard library

- **Main Program (`main.cpp`):** Provides an interactive menu-driven interface.
- **Tests (`test_calculator.cpp`):** Unit tests using assertions to verify correctness.
- **Build System (`Makefile`):** Automates compilation with targets for building, testing, and cleaning.

End-to-End CI/CD Pipeline Execution

This section demonstrates the complete CI/CD pipeline flow for the **Scientific Calculator** project, starting from a code commit on GitHub and ending with the deployed Docker container running on the target host.

1. **Code Commit on GitHub:** A commit pushed to the GitHub repository triggers the pipeline via webhook.
2. **Jenkins Pipeline Execution:** Jenkins automatically fetches the updated code, builds the project using Make, runs unit tests, creates a Docker image, and pushes it to Docker Hub. It then runs the Ansible playbook to deploy the container on the host machine.
3. **Email Notifications:** Jenkins sends email notifications on pipeline success or failure to the configured email address.
4. **Deployment Verification:** Finally, the Docker container is launched and verified using `docker ps`, confirming that the Scientific Calculator application is running successfully.


```
 > git config core.sparsecheckout # timeout=10
 > git checkout -f dc5f7b0cbd4e899518f0c93acd613ae5d7f7ba68 # timeout=10
Commit message: "test"
 > git rev-list --no-walk fc26e313c25081bfdaa93ac8ce4b31b93950ab5b # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] git
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/SPE_Calc/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/bajoriya-vaibhav/SPE_Mini_Project.git # timeout=10
Fetching upstream changes from https://github.com/bajoriya-vaibhav/SPE_Mini_Project.git
 > git --version # timeout=10
 > git --version # 'git version 2.34.1'
 > git fetch --tags --force --progress -- https://github.com/bajoriya-vaibhav/SPE_Mini_Project.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision dc5f7b0cbd4e899518f0c93acd613ae5d7f7ba68 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f dc5f7b0cbd4e899518f0c93acd613ae5d7f7ba68 # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
```

Figure 12: Pipeline Execution – Checkout Stage

```
Commit message: "test cases updated"
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] sh
+ make clean
rm -f calculator test_calculator
+ make
g++ -std=c++11 -Wall -static -Iinclude -o calculator src/main.cpp src/calculator.cpp
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] sh
+ make test
g++ -std=c++11 -Wall -static -Iinclude -o test_calculator tests/test_calculator.cpp src/calculator.cpp
```

Figure 13: Pipeline Execution – Build Stage (make build)

```
==================================
Running Calculator Unit Tests...
==================================

[TEST 1] Square Root Function
  Input: 49
  Expected: 7
  Actual: 7
  Status: PASSED ✓

[TEST 2] Factorial Function
  Input: 4
  Expected: 24
  Actual: 24
  Status: PASSED ✓

[TEST 3] Natural Logarithm Function
  Input: 2.718280
  Expected: ~1.0
  Actual: 0.999999
  Difference: 0
  Status: PASSED ✓

[TEST 4] Power Function
  Input: base=2, exponent=5
  Expected: 32
  Actual: 32.000000
  Status: PASSED ✓

==================================
All tests passed! ✓✓✓✓
Total Tests: 4 | Passed: 4 | Failed: 0
```

Figure 14: Pipline Execution - Unit Testing

```
Sending build context to Docker daemon  5.099MB

Step 1/7 : FROM gcc:latest AS build
 ---> e28cb2272e65
Step 2/7 : WORKDIR /app
 ---> Using cache
 ---> b761f24515ad
Step 3/7 : COPY . /app
 ---> ce8b24c10d7c
Step 4/7 : RUN make
 ---> Running in 88e5d1b5ffac
make: Nothing to be done for 'all'.
 ---> Removed intermediate container 88e5d1b5ffac
 ---> db8b592b02ee
Step 5/7 : FROM scratch
 --->
Step 6/7 : COPY --from=build /app/calculator /bin/calc
 ---> Using cache
 ---> d0213ac7460a
Step 7/7 : CMD ["/bin/calc"]
 ---> Using cache
 ---> 4182627ff954
Successfully built 4182627ff954
Successfully tagged ****/spe_mini_calculator:latest
[Pipeline] sh
```

Figure 15: multi-stage Docker build

```
+ echo ****
+ docker login -u **** --password-stdin
Login Succeeded
+ docker push ****/spe_mini_calculator:latest
The push refers to repository [docker.io/****/spe_mini_calculator]
acf4fdb7d4f5: Preparing
acf4fdb7d4f5: Layer already exists
latest: digest: sha256:c8e573d369057c72ed75c7be72a7f78234d3dba74078a77e50a65d5e94dc6558 size: 527
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy with Ansible)
[Pipeline] sh
+ ansible-playbook -i ansible/inventory.ini ansible/playbook.yml

PLAY [local] ******************************************************************

TASK [Gathering Facts] ********************************************************
[WARNING]: Platform linux on host localhost is using the discovered Python
interpreter at /usr/bin/python3.10, but future installation of another Python
interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-
core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]
```

Figure 16: pushing a Docker image to a registry.

```
TASK [Pull Docker image] *********************************************************
ok: [localhost]

TASK [Run calculator container] **********************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Successfully executed the pipeline
[Pipeline] emailext
Sending email to: vaibhav.bajoriya@iiitb.ac.in
[Pipeline] cleanWs
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] done
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Figure 17: deployment using an Ansible playbook followed by the final post-run actions and success status of the CI/CD pipeline.

```
vaibhav@Vaibhav:/mnt/d/Desktop/SPE_Mini_Project$ docker exec -it calculator /bin/calc

=== Scientific Calculator ===
1. Square Root
2. Factorial
3. Natural Logarithm (ln)
4. Power Function (x^b)
5. Exit
Enter your choice: 4
Enter base: 4
Enter exponent: 3
4^3 = 64

=== Scientific Calculator ===
1. Square Root
2. Factorial
3. Natural Logarithm (ln)
4. Power Function (x^b)
5. Exit
Enter your choice: 5
Exiting...
```

Figure 18: Docker container running the Scientific Calculator application after deployment

Conclusion

In this project, a complete CI/CD pipeline was successfully implemented for the Scientific Calculator application using modern DevOps tools. The workflow begins with a code commit to GitHub, which triggers Jenkins to automatically build the C++ project using Make and run unit tests. The compiled application is then containerized with Docker using a multi-stage build to create a minimal, efficient image. The Docker image is pushed to Docker Hub, making it easily available for deployment.

Ansible is used to automate the deployment of the Docker container on the target host, ensuring reproducibility and reducing manual intervention. Email notifications keep developers informed about build status. Finally, the running container is validated, confirming that the system works end-to-end.

This pipeline demonstrates how continuous integration, containerization, and automated deployment can be effectively combined to streamline the software delivery process for C++ applications, making them portable, scalable, and reliable.

References and Resources

- **GitHub Repository:** SPE Calculator on GitHub
- **Docker Hub Repository:** SPE Calculator on Docker Hub