DevOps CI/CD Assignment

Intern: Sumit Dhar
Project: Calculator Application (CI/CD pipeline with Jenkins & Docker)
Date: 09th September, 2025

---

1. Introduction

This report documents the implementation of a CI/CD pipeline for a Python-based **Calculator Application**. The pipeline automates code checkout, Docker-based builds, unit testing, and package creation, while also being extensible to deploy on AWS cloud.

Key Objectives:

- Learn DevOps culture and CI/CD best practices.
- Automate testing and builds using **Jenkins Pipelines**.
- Containerize applications with **Docker** for consistency.
- Establish GitHub → Jenkins integration with **webhooks**.
- Prepare for cloud deployment on **AWS**.

---

2. Pre-requisites

Before setup, the following tools were installed and configured:

1. **System & Tools**
   - Windows system (DG-provided)
   - Chocolatey package manager
2. **Installed via Chocolatey**
   - OpenJDK 17 (required by Jenkins)
   - Jenkins (CI/CD server)
   - Docker Desktop (containerization platform)
3. **Other Installations**
   - Git (version control)
   - Python 3.x (application runtime + unit testing)
   - ngrok (for exposing local Jenkins to GitHub webhooks)

---

3. Environment Setup

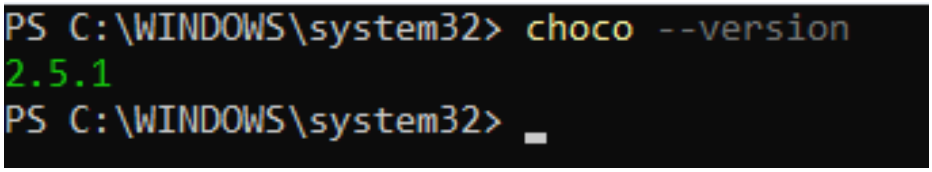
Step 1: Install Chocolatey (Windows Package Manager)

Chocolatey is used to simplify installation of dependencies like Java, Docker, and Jenkins.

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol = `
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

Verify Installation:

```
choco --version
```



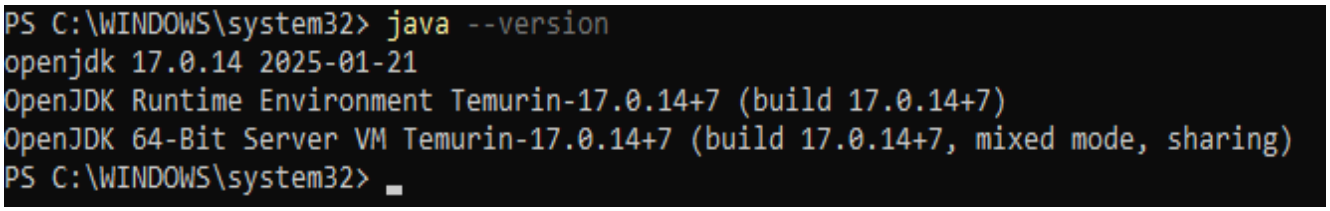
Step 2: Install Java (JDK 17+)

Jenkins requires Java to run.

```
choco install openjdk17 -y
```

Verify Installation:

```
java --version
```



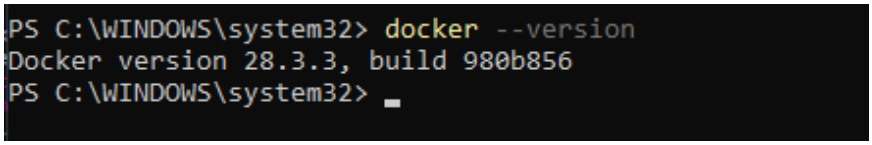
Step 3: Install Docker Desktop

Docker is used to build images and run containers for testing and packaging.

```
choco install docker-desktop -y
```

Verify Installation:

```
docker --version
```

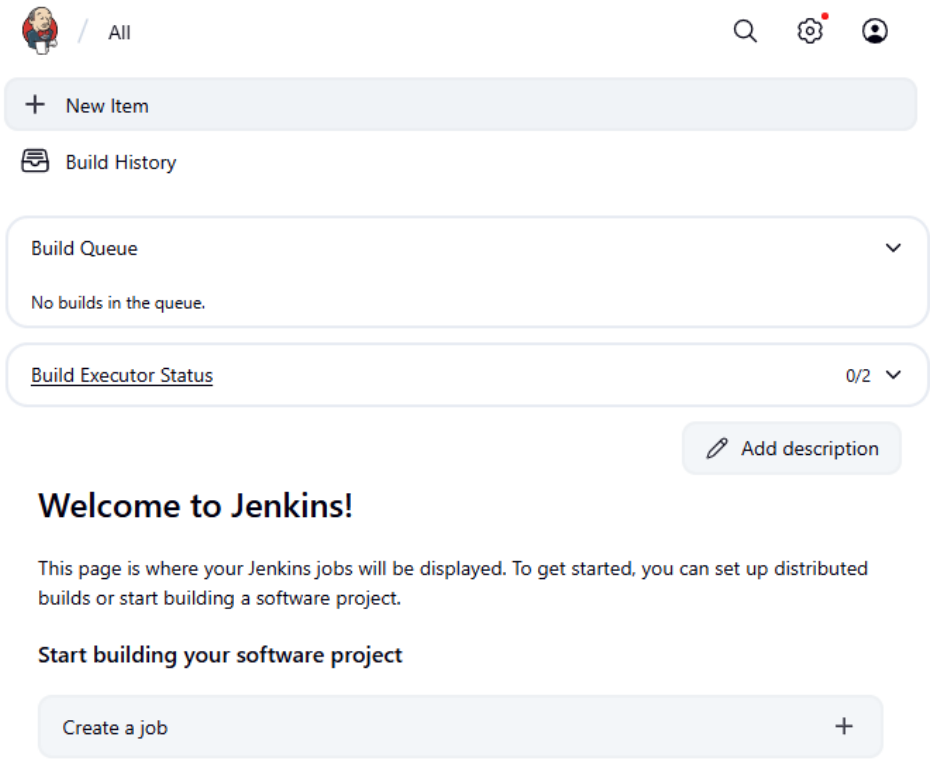


Step 4: Install Jenkins

Jenkins will orchestrate the CI/CD pipeline.

```
choco install jenkins -y
```

- Open Jenkins in browser: http://localhost:8080
- Unlock using initial Admin Password from: `C:\ProgramData\Jenkins\.jenkins\secrets\initialAdminPassword`
- Copy and paste it in the browser and sign in Jenkins.
- Install plugins: Git, Pipeline, Docker and other necessary ones.
- It will show the home page:



Step 5: Install Git

Git is used to clone repositories and integrate with Jenkins.

1. Go to Git Downloads
2. Download Windows installer and run with default options.

Verify Installation:

```
git --version
```

```
PS C:\WINDOWS\system32> git --version
git version 2.51.0.windows.1
PS C:\WINDOWS\system32>
```

Step 6: Install Python

Python is required to run unit tests and build packages.

1. Go to Python Downloads
2. Download latest 64-bit installer.
3. During installation, check “Add Python to PATH”.

Verify Installation:

```
Python --version
```

```
PS C:\WINDOWS\system32> python --version
Python 3.13.7
PS C:\WINDOWS\system32>
```

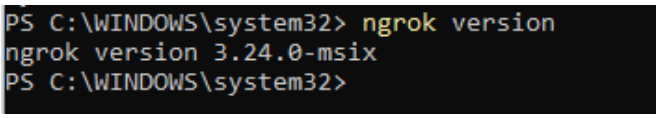
Step 7: Install ngrok

ngrok is used to expose local Jenkins webhook endpoints to GitHub.

1. Go to ngrok Downloads
2. Install via Microsoft Store or executable.

Verify Installation:

```
ngrok version
```



4. Set up authentication token:

```
ngrok config add-authtoken <YOUR_AUTHTOKEN>
```

---

4. Application Setup

Repository Structure:

```
calculator-app/
├─ calc_app/
│   ├─ __init__.py
│   └─ calc.py
├─ tests/
│   └─ test_calc.py
├─ .dockerignore
├─ .gitignore
├─ Dockerfile
├─ Jenkinsfile
├─ setup.py
└─ README.md
```

- `calc_app/` → Core calculator logic
- `tests/` → Unit tests using Python `unittest`
- `Dockerfile` → Defines container environment
- `Jenkinsfile` → CI/CD pipeline definition

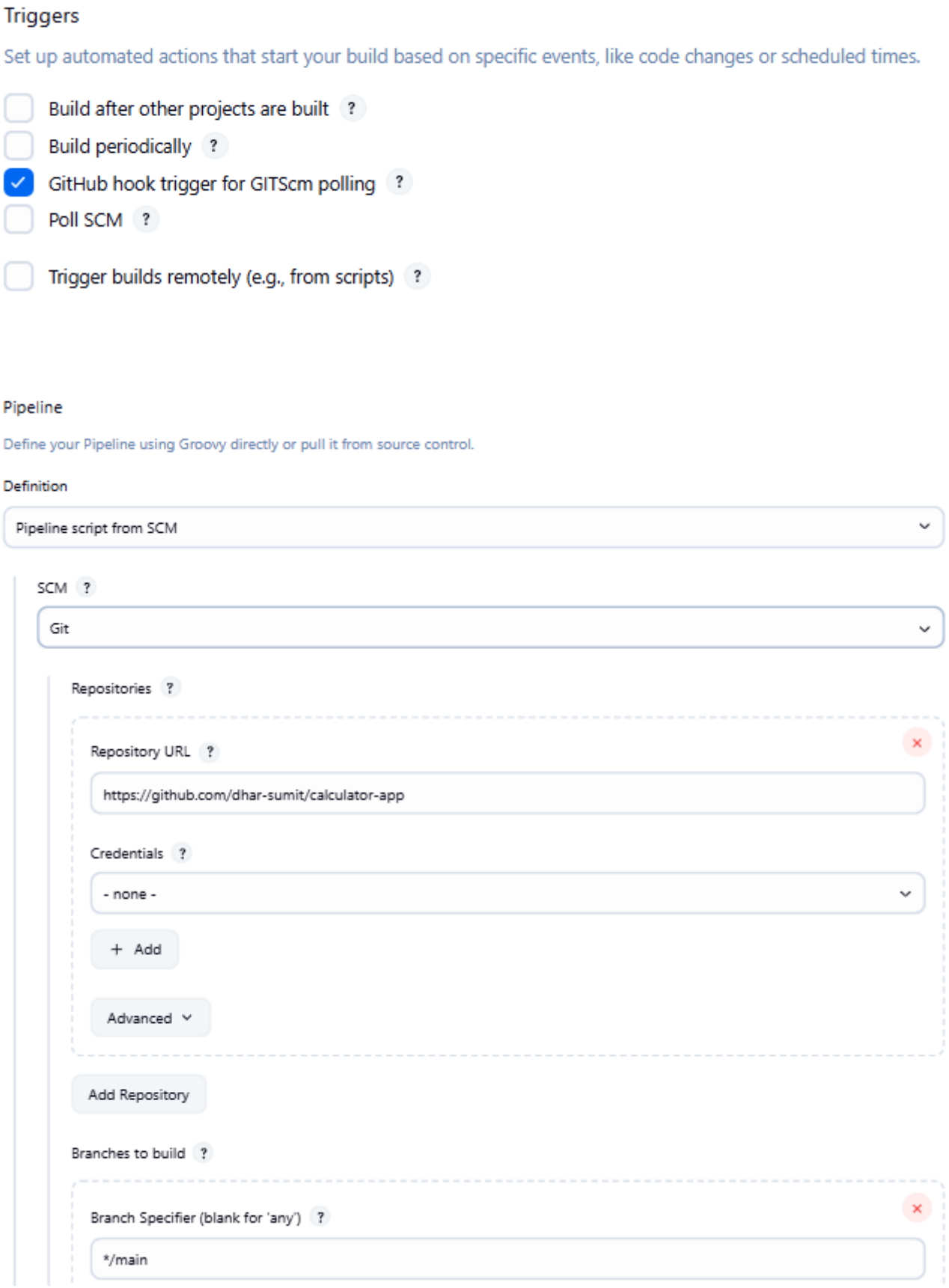
---

5. Jenkins Pipeline

Pipeline Stages

1. **Checkout** → Pull code from GitHub
2. **Build** → Build Docker image for calculator app
3. **Test** → Run Python `unittest` inside container
4. **Package** → Build Python wheel/distribution

Pipeline Configuration



Click Apply and Save it.

Jenkinsfile (Key Extract)

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/dhar-sumit/calculator-
app.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                bat 'docker build -t calculator-app .'
            }
        }

        stage('Run Tests in Container') {
            steps {
                // run container and execute tests
                bat 'docker run --rm calculator-app python -m unittest discover -s tests'
            }
        }

        stage('Build Python Package') {
            steps {
                // Assuming you have setup.py in your repo
                bat 'docker run --rm -v %CD%:/app -w /app calculator-app python setup.py
sdist bdist_wheel'
            }
        }
    }

    post {
        always {
            echo 'Cleaning up...'
            bat 'docker ps -a'
        }

        success {
            echo 'CI pipeline finished successfully!'
        }

        failure {
            echo 'CI pipeline failed!'
        }
    }
}
```

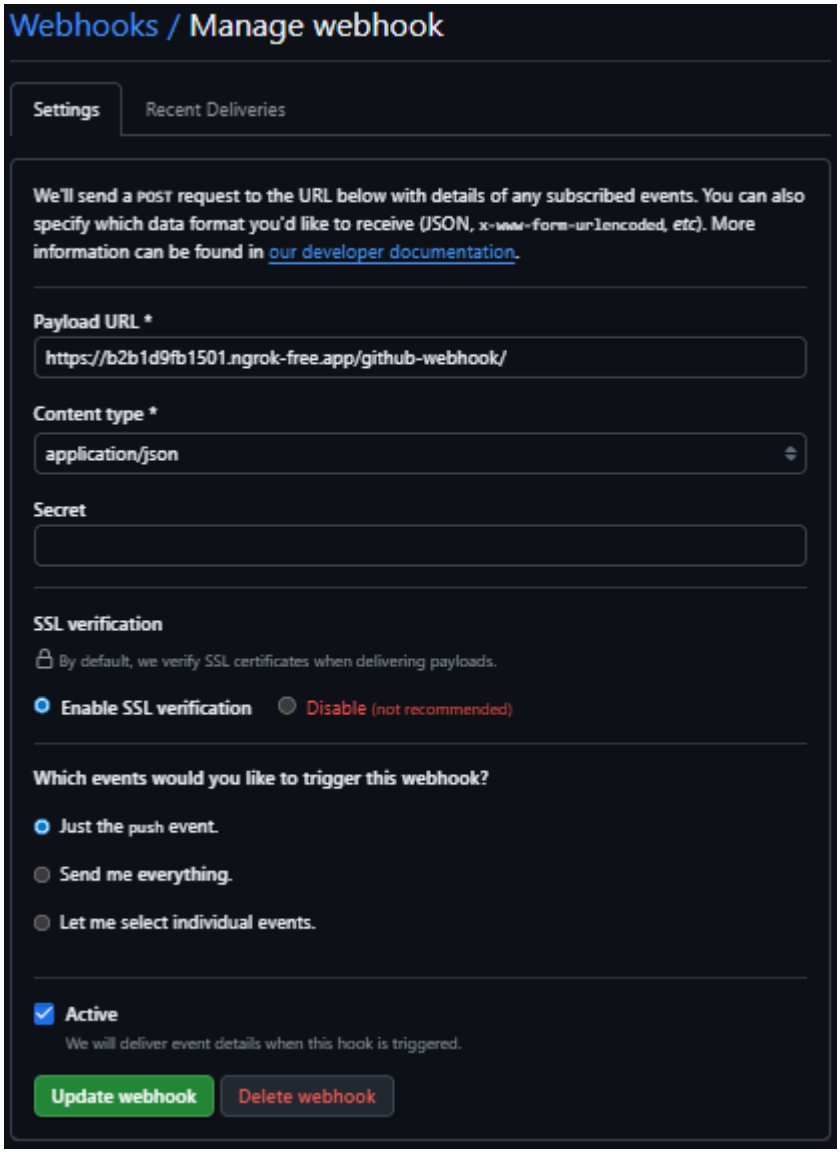
---

6. GitHub Webhook Integration

- Expose Jenkins using ngrok:

```
ngrok http 8080
```

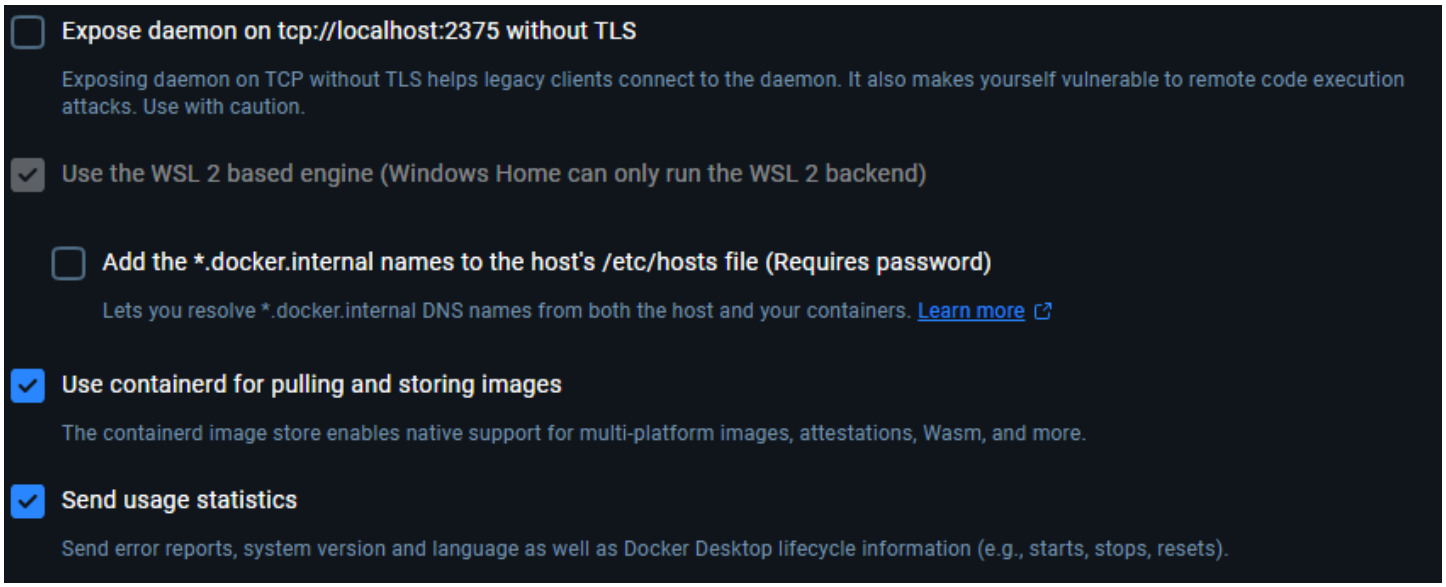
- Copy generated URL:
  `https://<ngrok_id>.ngrok-free.app/github-webhook/`
- Add webhook in GitHub repo:
  - **Payload URL** → above URL
  - **Content Type** → application/json
  - **Trigger** → push events



---

6. Docker Configuration

Go to Settings and update this in the General section:



---

7. Running the Pipeline

1. Push code changes to GitHub.
2. Webhook triggers Jenkins pipeline.
3. Pipeline executes:
    - Code checkout
    - Docker build
    - Unit tests execution
    - Package build
4. Jenkins console shows results → Success / Failure

Pipeline Execution: Failures & Success

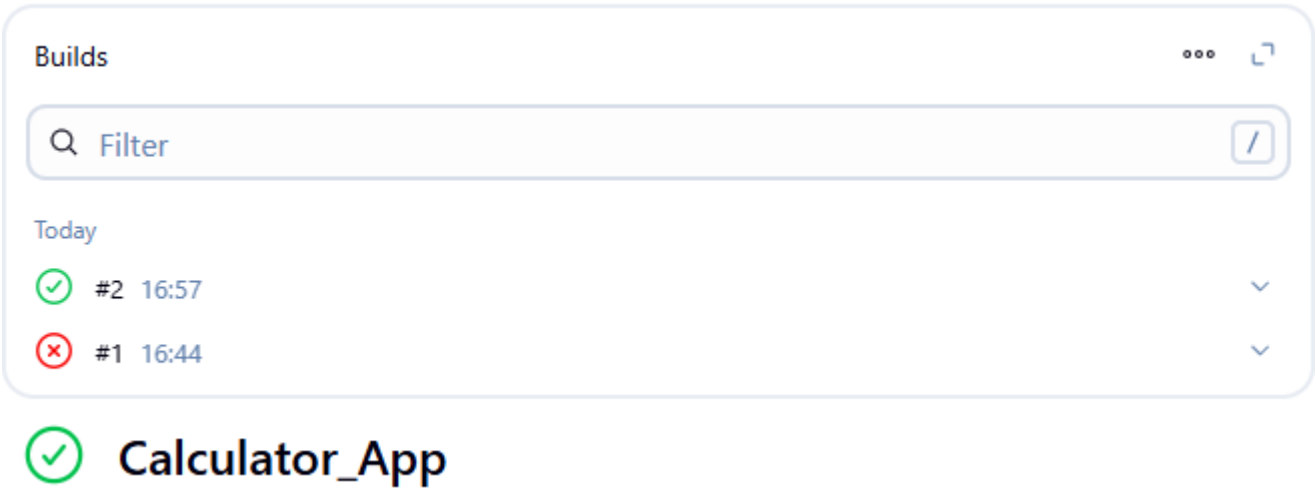
During the assignment, the pipeline was run multiple times. The initial runs failed due to configuration issues (such as missing `setup.py` and incorrect Docker paths). These were corrected, and the final run succeeded.

Step 1: Trigger the Pipeline

- Navigate to Jenkins Dashboard → **calculator-app** pipeline.
- Click **Build Now** or push a commit to GitHub (which triggers automatically via webhook).

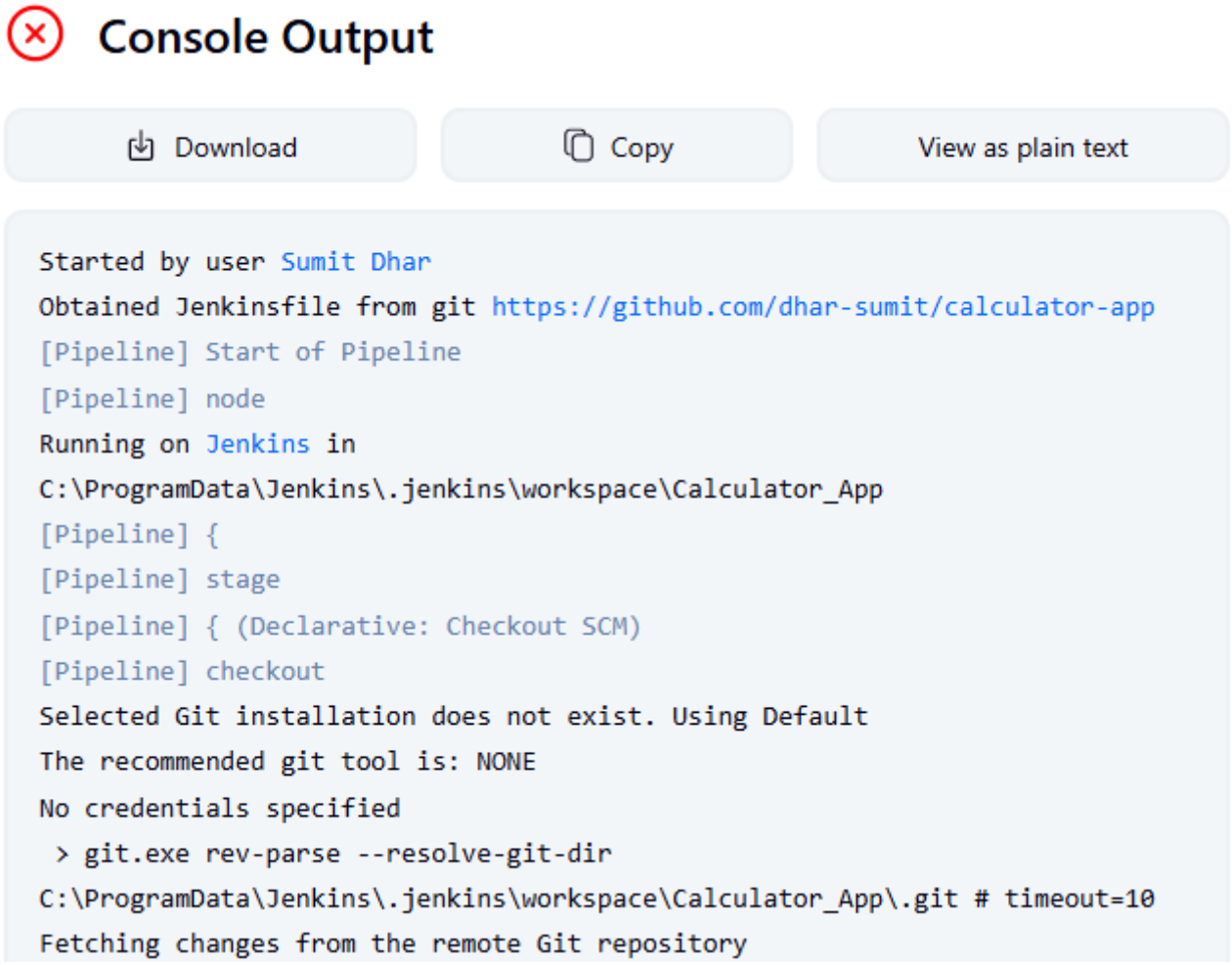
Step 2: Observe Build Status

- Jenkins UI shows build history with color codes:
    - **Red** = Failed Build
    - **Green** = Successful Build



Step 3: Debug Failures

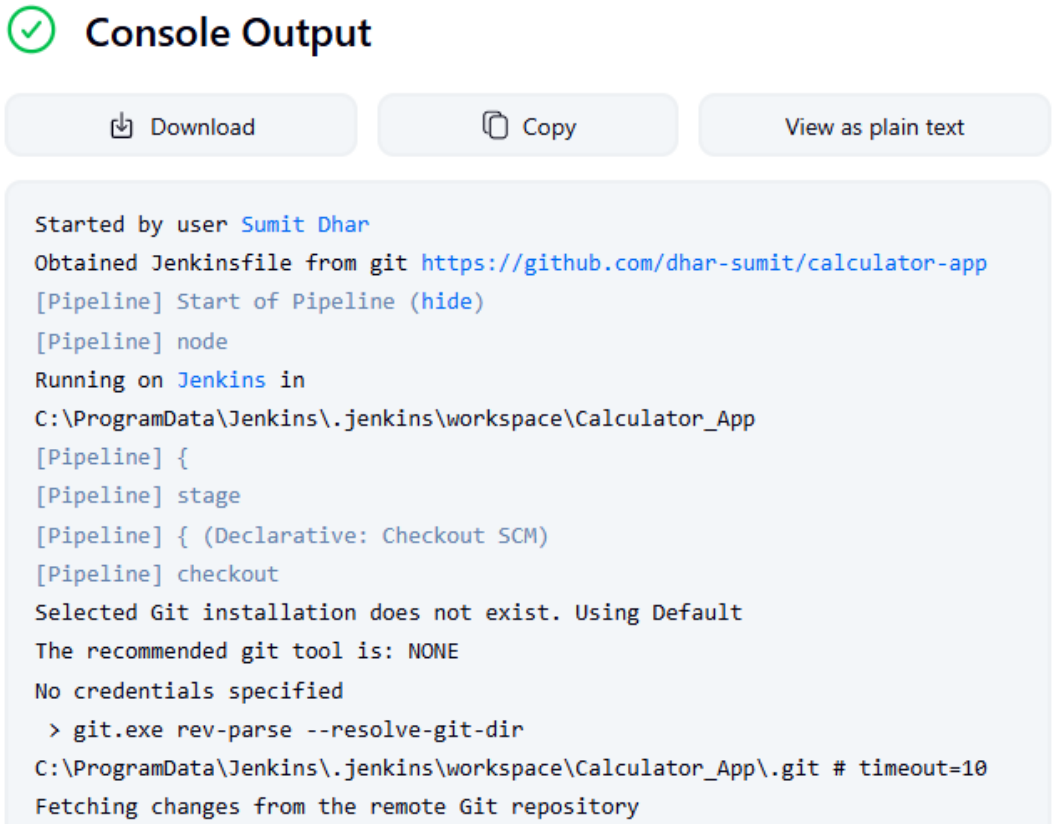
- Click on a failed build (e.g., `#1`).
- Open **Console Output** → review error logs (e.g., `setup.py not found`, test failures).
- Fix issues locally → commit & push → Jenkins retriggers.



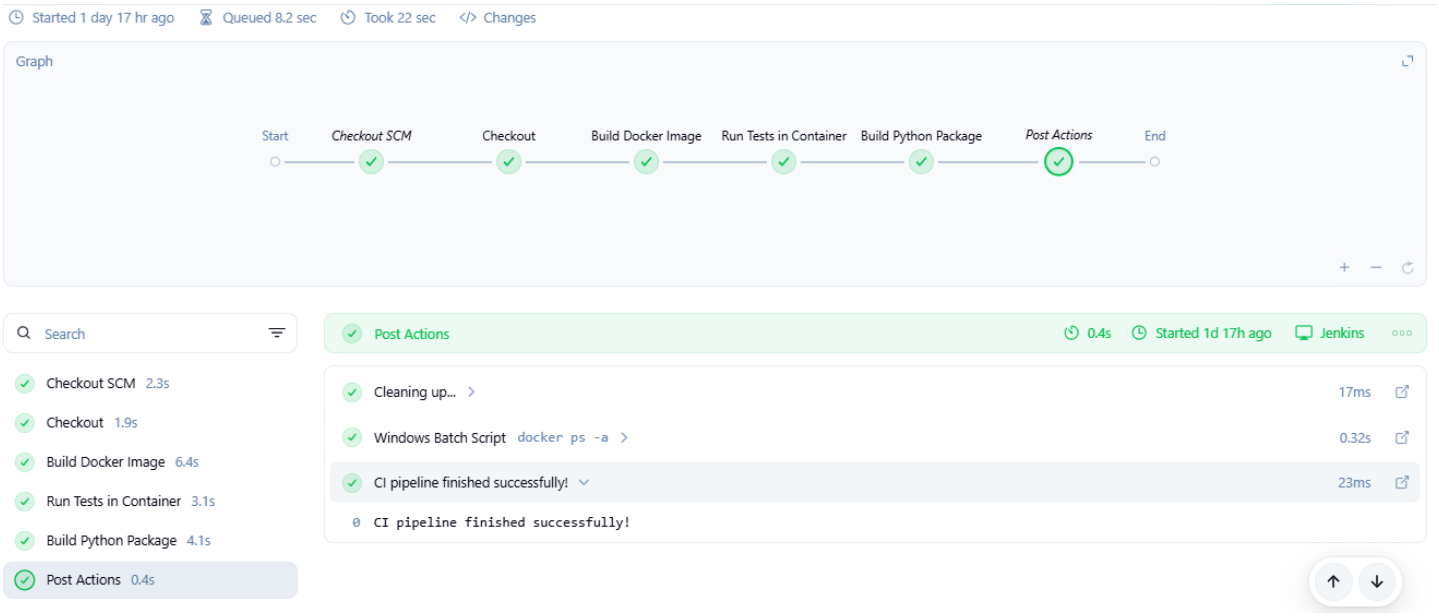
After fixing the error, again go to Step 1.

Step 4: Verify Success

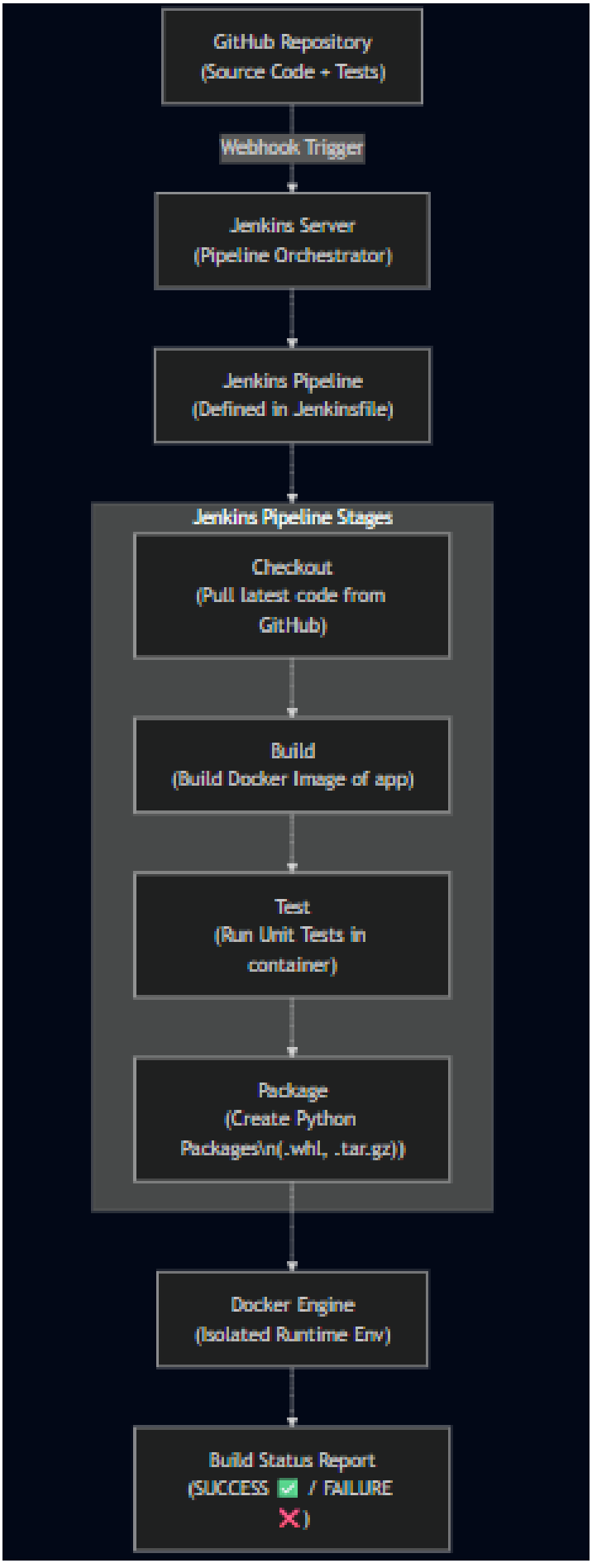
- On successful run, build is marked as **Success**.
- Console Output confirms:
  - Code checked out successfully
  - Docker image built successfully
  - Unit tests executed → all passed
  - Python package built (`dist/*.whl` created)



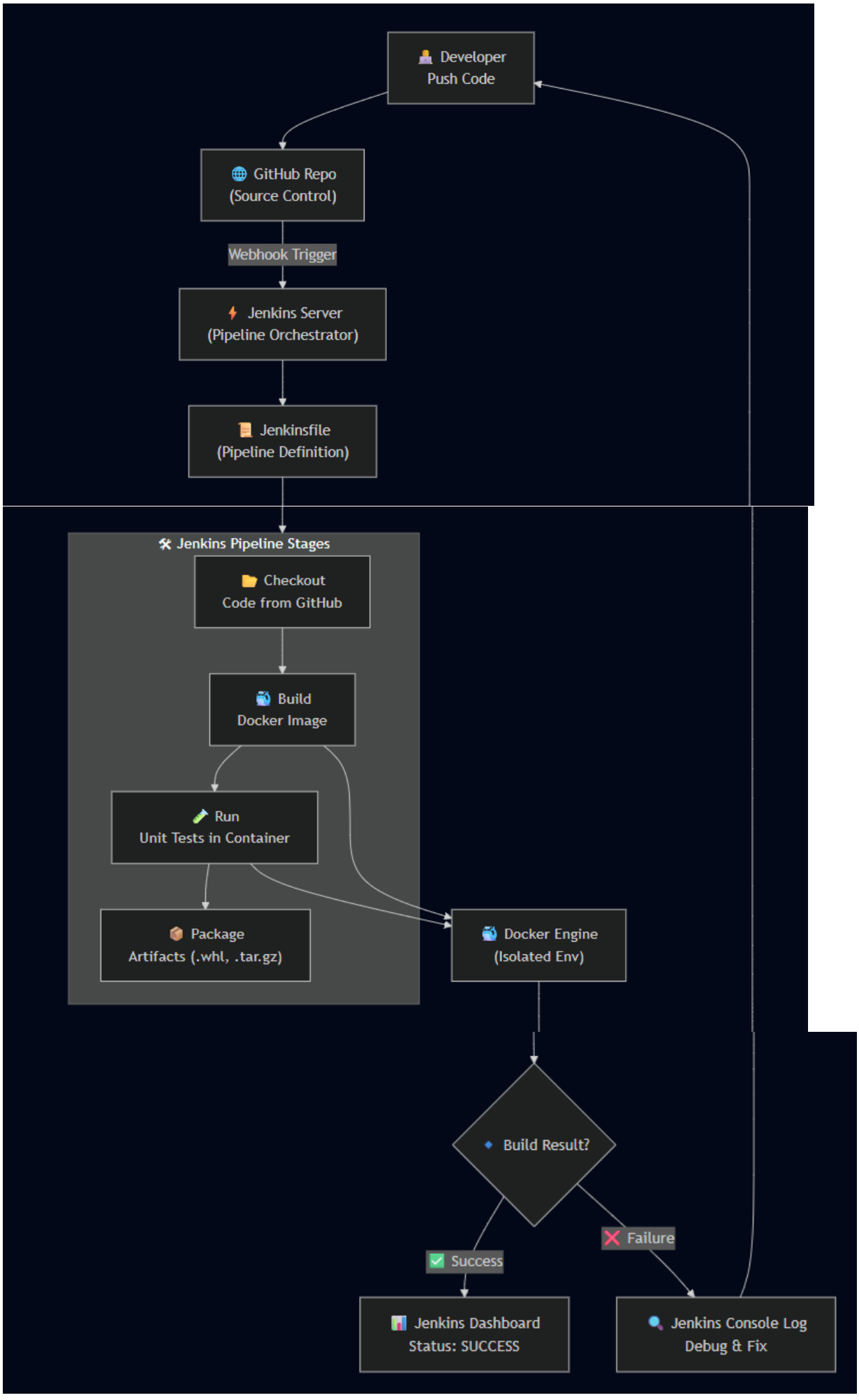
Pipeline Overview



8. Architecture Diagram



8. Flowchart Diagram



9. Key Learnings & Observations

- **Automation** → Reduced manual effort in testing/building.
- **Dockerized CI** → Consistent environment across dev & CI.
- **Faster Feedback** → Immediate test results on every commit.

---

10. Conclusion

This assignment successfully demonstrated setting up a **CI/CD pipeline** for a Python-based calculator application.

✓ Local development + unit testing
✓ GitHub version control
✓ Jenkins pipeline automation
✓ Docker-based builds & tests
✓ GitHub → Jenkins → Docker → Package workflow

Future scope:

- Extend to **AWS deployment** (ECS/Lambda)
- Integrate with monitoring/logging for production-readiness