

# DevOps CI/CD Assignment Report

**Intern:** Sumit Dhar

**Project:** Calculator Application (CI/CD pipeline with Jenkins & Docker)

**Date:** 09<sup>th</sup> 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
```

```
PS C:\WINDOWS\system32> choco --version
2.5.1
PS C:\WINDOWS\system32> _
```

---

## Step 2: Install Java (JDK 17+)

Jenkins requires Java to run.

```
choco install openjdk17 -y
```

## Verify Installation:

```
java --version
```

```
PS C:\WINDOWS\system32> java --version
openjdk 17.0.14 2025-01-21
OpenJDK Runtime Environment Temurin-17.0.14+7 (build 17.0.14+7)
OpenJDK 64-Bit Server VM Temurin-17.0.14+7 (build 17.0.14+7, mixed mode, sharing)
PS C:\WINDOWS\system32> _
```

---

## 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
```

```
PS C:\WINDOWS\system32> docker --version
Docker version 28.3.3, build 980b856
PS C:\WINDOWS\system32> _
```

---

## 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:



/ All



+ New Item



Build History

Build Queue



No builds in the queue.

Build Executor Status

0/2



Add description

## Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job



---

## 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
```

```
PS C:\WINDOWS\system32> ngrok version
ngrok version 3.24.0-msix
PS C:\WINDOWS\system32>
```

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

### Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

### Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

#### Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/dhar-sumit/calculator-app

Credentials ?

- none -

+ Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

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!'
        }
    }
}

```

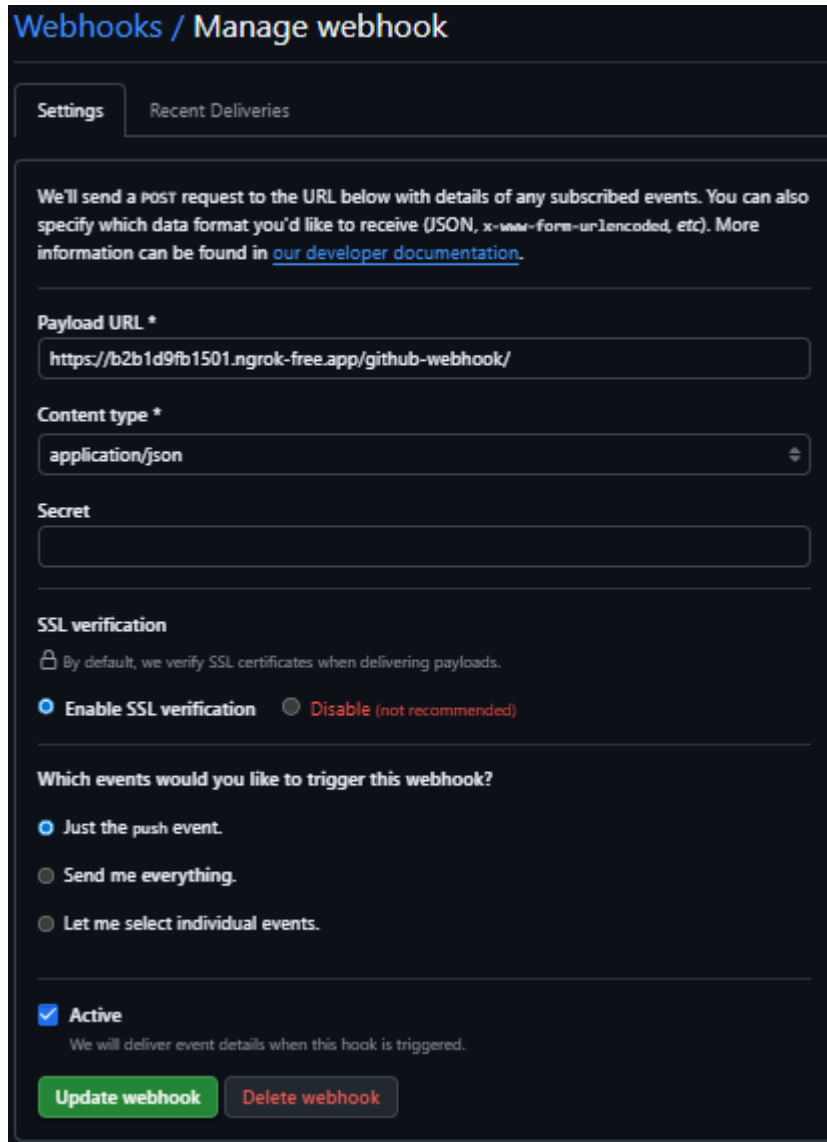
---

## 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



The screenshot shows the 'Webhooks / Manage webhook' interface. It has two tabs: 'Settings' (active) and 'Recent Deliveries'. The main content area explains that GitHub will send a POST request to the specified URL with event details. It includes a text input for 'Payload URL' containing 'https://b2b1d9fb1501.ngrok-free.app/github-webhook/'. Below this is a dropdown for 'Content type' set to 'application/json'. There is an empty text field for 'Secret'. The 'SSL verification' section has a lock icon and text 'By default, we verify SSL certificates when delivering payloads.', with radio buttons for 'Enable SSL verification' (selected) and 'Disable (not recommended)'. The 'Which events would you like to trigger this webhook?' section has radio buttons for 'Just the push event.' (selected), 'Send me everything.', and 'Let me select individual events.'. At the bottom, there is a checked checkbox for 'Active' with the text 'We will deliver event details when this hook is triggered.', and two buttons: 'Update webhook' (green) and 'Delete webhook' (red).

---

## 6. Docker Configuration

Go to Settings and update this in the General section:

☐ **Expose daemon on tcp://localhost:2375 without TLS**  
Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.

☒ **Use the WSL 2 based engine (Windows Home can only run the WSL 2 backend)**

☐ **Add the \*.docker.internal names to the host's /etc/hosts file (Requires password)**  
Lets you resolve \*.docker.internal DNS names from both the host and your containers. [Learn more](#) ↗

☒ **Use containerd for pulling and storing images**  
The containerd image store enables native support for multi-platform images, attestations, Wasm, and more.

☒ **Send usage statistics**  
Send error reports, system version and language as well as Docker Desktop lifecycle information (e.g., starts, stops, resets).

---

## 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



Builds

Filter

Today

✓ #2 16:57

✗ #1 16:44

## ✓ Calculator\_App

### 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.

## ✗ Console Output

Download

Copy

View as plain text

```
Started by user Sumit Dhar
Obtained Jenkinsfile from git https://github.com/dhar-sumit/calculator-app
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in
C:\ProgramData\Jenkins\.jenkins\workspace\Calculator_App
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> git.exe rev-parse --resolve-git-dir
C:\ProgramData\Jenkins\.jenkins\workspace\Calculator_App\.git # timeout=10
Fetching changes from the remote Git repository
```

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)



## Console Output



Download



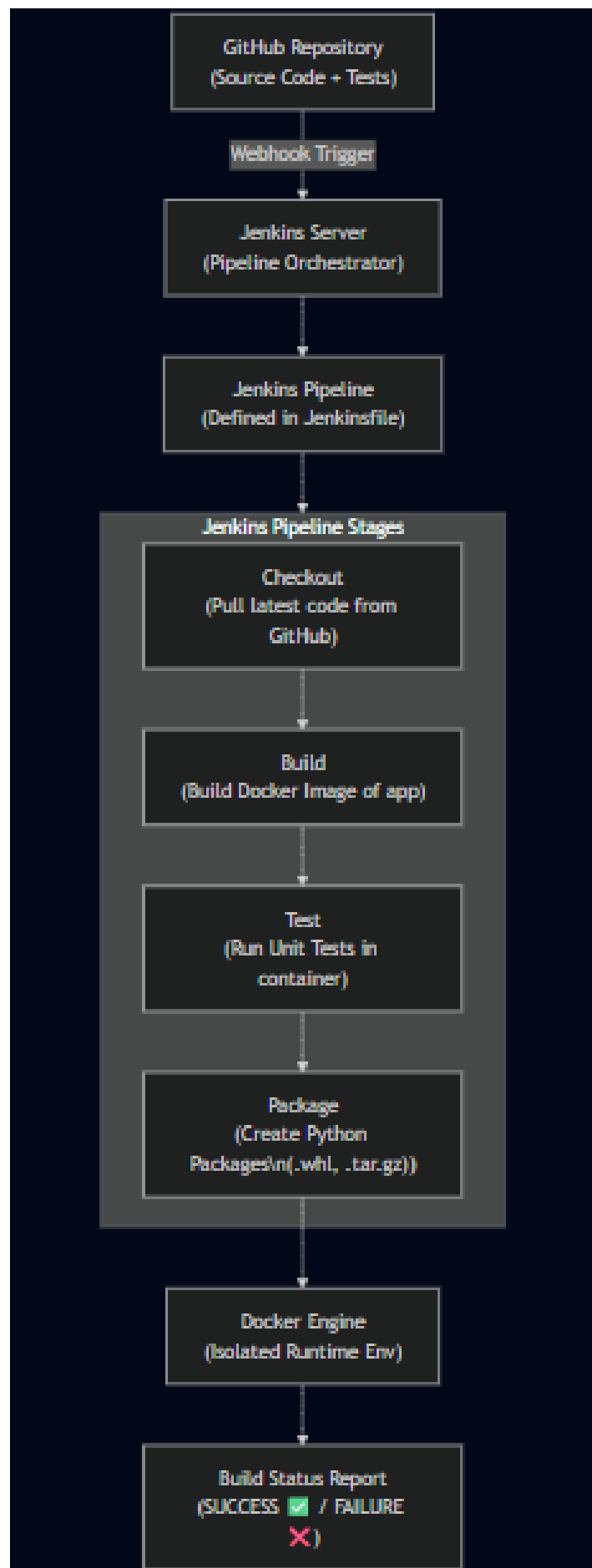
Copy

View as plain text

```
Started by user Sumit Dhar
Obtained Jenkinsfile from git https://github.com/dhar-sumit/calculator-app
[Pipeline] Start of Pipeline (hide)
[Pipeline] node
Running on Jenkins in
C:\ProgramData\Jenkins\.jenkins\workspace\Calculator_App
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
  > git.exe rev-parse --resolve-git-dir
C:\ProgramData\Jenkins\.jenkins\workspace\Calculator_App\.git # timeout=10
Fetching changes from the remote Git repository
```

---

## 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
-