Scientific Calculator with DevOps Pipeline

Tanish Pathania
Roll Number: IMT2022049

October 4, 2025

Contents

1 Project Details **2**

2 Introduction **2**

3 What is DevOps **2**

4 How to Implement DevOps **2**

5 Tools Used & Environment Configuration **3**

6 Credentials Required **6**

7 Jenkins Credential Configuration **7**

8 Project Workflow & Setup **7**

9 Build & Test Project using Maven **8**

10 Setting Up Project with Git and GitHub **10**

11 Containerizing the Project Using Docker **11**

12 Create New Jenkins Project **12**
12.1 Jenkins 14

13 Ansible Setup **14**
13.1 Ansible Deployment 15

14 Conclusion **16**

1 Project Details

Name: Tanish Pathania
Roll Number: IMT2021049
GitHub Repository: https://github.com/Tanish-pat/scientific-calculator.git
Docker Hub Repository:https://hub.docker.com/repository/docker/tanish688/scientific-calculator

2 Introduction

This project demonstrates the complete **DevOps pipeline** for a *scientific calculator application*. The primary goal is to **automate** the development, testing, and deployment lifecycle using modern DevOps tools such as **GitHub**, **Jenkins**, **Docker**, and **Ansible**.

The workflow seamlessly integrates **code versioning**, **automated builds**, **containerization**, and **environment consistency** — delivering a *reliable* and *repeatable* software delivery pipeline.

3 What is DevOps

DevOps is a modern development approach that evolved from earlier models like *Agile* and *Waterfall*. While **Waterfall** follows a *sequential flow* and **Agile** introduces *iterative cycles*, **DevOps** merges **development** and **operations** into a *continuous, collaborative process*.

DevOps breaks down the traditional barriers between teams. **Development**, **testing**, and **operations** work together in *real time* to deliver **frequent** and **stable updates**. This continuous **feedback loop** ensures that new features are deployed *rapidly*, issues are resolved *immediately*, and the software remains **stable** across all environments.

The core advantage of **DevOps** is **integration** — developers **write** and **commit code**, **automated systems** build and test it, and **operations teams** deploy it seamlessly. The result is **faster delivery**, **reduced human errors**, and **enhanced collaboration** across teams.

4 How to Implement DevOps

Implementing **DevOps** effectively requires leveraging the **right tools** to automate every step — from **code creation** to **deployment**. *Collaboration* and *automation* form the backbone of the DevOps model.

IDE (Integrated Development Environment)

A robust **IDE** supports **version control** (like *Git*) and **build tools** (such as *Maven*). Developers use it to **write**, **build**, and **test applications efficiently** while maintaining **synchronization** with version control systems.

Maven

Maven automates the **building**, **testing**, and **packaging** of Java applications. It ensures **dependency management** is handled automatically and that the **build process** is **uniform** across all environments.

Git & GitHub

Git is the local *version control system* used to manage code changes. **GitHub** serves as a **remote repository**, enabling **collaboration** among multiple developers. Through **branching** and

merging, GitHub simplifies **code integration** and provides **visibility** into all project changes.

Jenkins

Jenkins automates **continuous integration (CI)** by building the project whenever new code is **pushed to GitHub**. Using **webhooks**, Jenkins triggers **automatic builds**, eliminating manual intervention and ensuring **rapid feedback**.

Jenkins Pipeline

The **CI/CD pipeline** in Jenkins automates the **build**, **test**, and **deployment** stages. It supports **dependency chaining**, meaning each phase executes only after the **previous step succeeds**, ensuring **reliability** in the release process.

Docker

Docker containerizes applications, guaranteeing that they run **identically** across all environments. Containers are **lightweight**, **portable**, and consume **fewer resources** compared to virtual machines.

Docker Hub

Docker Hub is a **registry for container images**. Teams can **push** and **pull images** easily, enabling **rapid deployment** and **testing**.

Ansible

Ansible automates **configuration management** and **deployment**. It ensures that every environment has **consistent configurations**, eliminating the *"works on my machine"* problem.

5 Tools Used & Environment Configuration

Since the local environment was preconfigured, an Ubuntu 20.04 Docker container was created to demonstrate installation of all DevOps tools.

Java & Maven Setup

Java Installation

Java is a cross-platform, object-oriented programming language. In this project, Java executes backend components and supports build automation with Maven.

```
sudo apt update
sudo apt install fontconfig openjdk-17-jre
java -version
```

Listing 1: Java Installation

Sample Output:

```
openjdk version "17.0.8" 2023-07-18
OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode)
```

Maven Installation

Maven automates project compilation, dependency resolution, and testing.

```
wget https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz
tar -xvf apache-maven-3.9.9-bin.tar.gz
sudo mv apache-maven-3.9.9 /opt/
export M2_HOME=/opt/apache-maven-3.9.9
export PATH=$M2_HOME/bin:$PATH
```

Listing 2: Maven Installation Commands

After installation, reload your terminal session:

```
source ~/.profile
mvn --version
```

Sample Output:

```
Apache Maven 3.9.9
Java version: 17.0.12, vendor: Ubuntu
OS name: "linux", arch: "amd64"
```

Java and Maven Versions

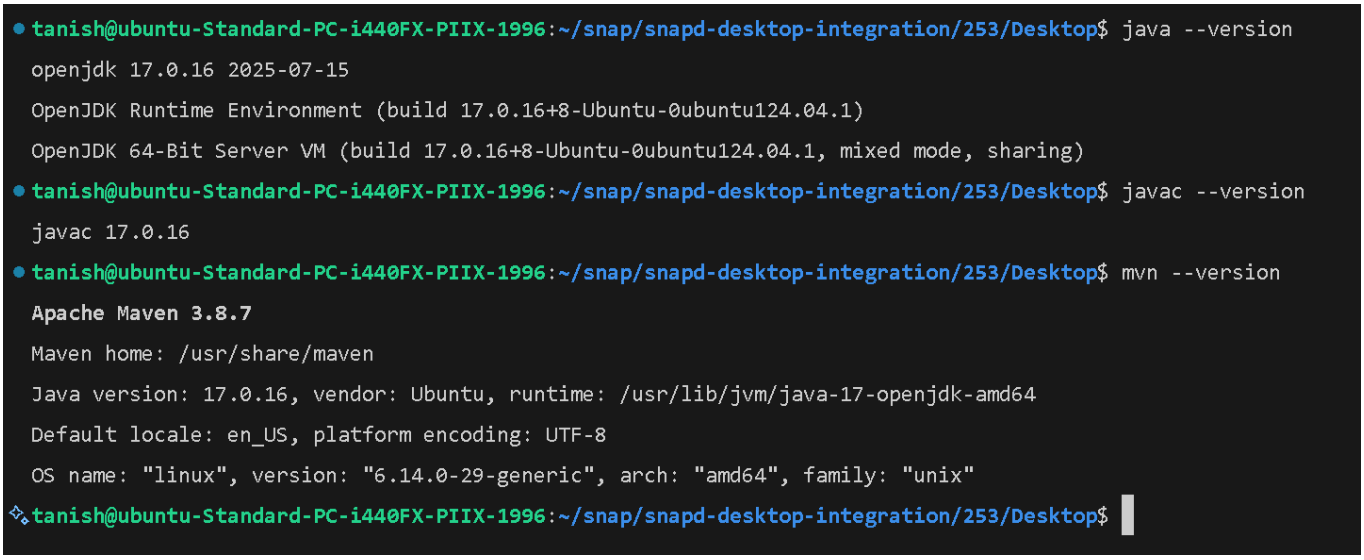

Figure 1: Verification of Java and Maven versions.

Git & GitHub

Git is a distributed version control system that enables version tracking and collaborative development. GitHub serves as the remote repository for synchronization, issue tracking, and CI integration.

```
sudo apt install git
git --version
```

Listing 3: Git Installation

Usage:

- Local version control managed via Git CLI.
- Remote synchronization via GitHub for collaborative workflows.
- Integrated with Jenkins for automated CI/CD pipelines.

Docker

Docker enables consistent deployment across platforms through lightweight containers. It ensures environmental consistency and accelerates CI/CD operations.

Docker Installation

```
# Remove conflicting packages
for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd
    runc; do
  sudo apt-get remove -y $pkg
done

# Add Docker GPG key and repository
sudo apt-get update
sudo apt-get install -y ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.
    asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] \
  https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Docker Verification

```
sudo docker ps
sudo groupadd docker
sudo usermod -aG docker $USER
```

Jenkins

Jenkins automates build, test, and deployment tasks. It continuously integrates GitHub changes into the CI/CD pipeline.

Jenkins Installation

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/" | \
sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update
sudo apt-get install jenkins
sudo systemctl start jenkins
```

```
sudo systemctl status jenkins
```

Once running, visit `http://localhost:8080`, enter the initial admin password, and install suggested plugins.

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Listing 4: Retrieve Jenkins Admin Password

Ansible

Ansible automates deployment and environment configuration using YAML playbooks. In this project, it pulls Docker containers from remote repositories and deploys them consistently.

Ansible Installation

```
pip3 install ansible
ansible --version
```

Listing 5: Ansible Installation Commands

Sample Output:

```
ansible [core 2.13.13]
python version = 3.8.10
jinja version = 3.1.4
libyaml = True
```

Git, Docker, Jenkins and Ansible Versions

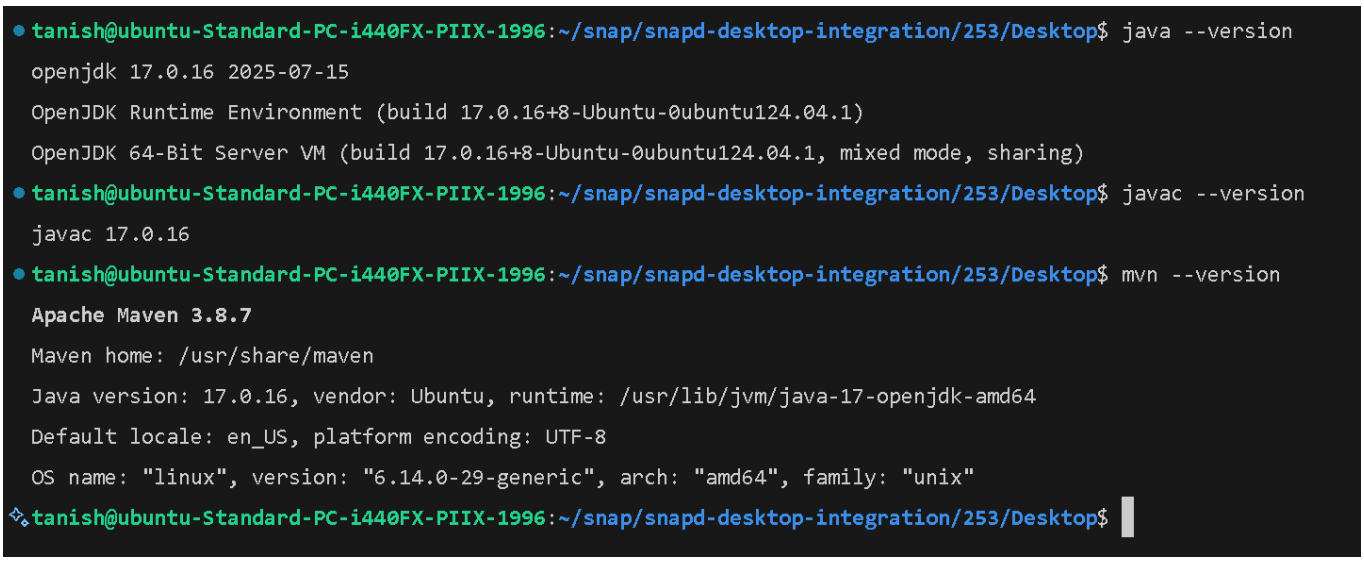

Figure 2: Verification of Git, Docker, Jenkins and Ansible versions.

6 Credentials Required

Before initializing CI/CD, ensure the following credentials are configured for authentication and integration.

GitHub

- **Username:** GitHub account username
- **Personal Access Token (PAT):** Authenticates private repositories

Docker Hub

- **Username:** Docker Hub username
- **Password/PAT:** Authentication for image push/pull operations
- **PAT Creation:** `https://app.docker.com/settings/personal-access-tokens/create` (Grant Read/Write access)

7 Jenkins Credential Configuration

Use Jenkins credentials to securely store authentication details instead of embedding them in the `Jenkinsfile`.

Credential IDs

- `github-cred-Tanish` — GitHub username + PAT (for repo access)
- `private_registry_creds` — Docker Hub username + password (for image push/pull)

Jenkins Path: Dashboard → Manage Jenkins → Credentials → System → Global Credentials (unrestricted) → Add Credentials —

8 Project Workflow & Setup

Create a Maven Project using VS Code

1. Open VS Code and press `Ctrl + Shift + P`.
2. Search for `Maven: New Project`.
3. Select `maven-archetype-quickstart`.
4. Enter your desired *Package Name* and *Artifact ID*.

This creates a minimal Java project structure with boilerplate files:

```
.
|- pom.xml
|- src
|  |- main/java/com/example/App.java
|  \- test/java/com/example/AppTest.java
\- target/
   |- classes/com/example/
   \- test-classes/com/example/
```

Listing 6: Project Directory Structure

Directory Descriptions

- **pom.xml** – Defines dependencies, build plugins, and project metadata.
- **src/main/java** – Contains the main application logic.
- **App.java** – Core logic of the scientific calculator.
- **src/test/java** – Houses test cases.
- **AppTest.java** – Unit test class for validating functionality.
- **target/** – Contains compiled and packaged output.

—

9 Build & Test Project using Maven

Build the Project

Compile and package the source code:

```
mvn clean install
```

Explanation:

- `clean` – Removes previous build artifacts.
- `install` – Compiles code, runs tests, and installs the artifact to the local Maven repository.

Maven Build and Test

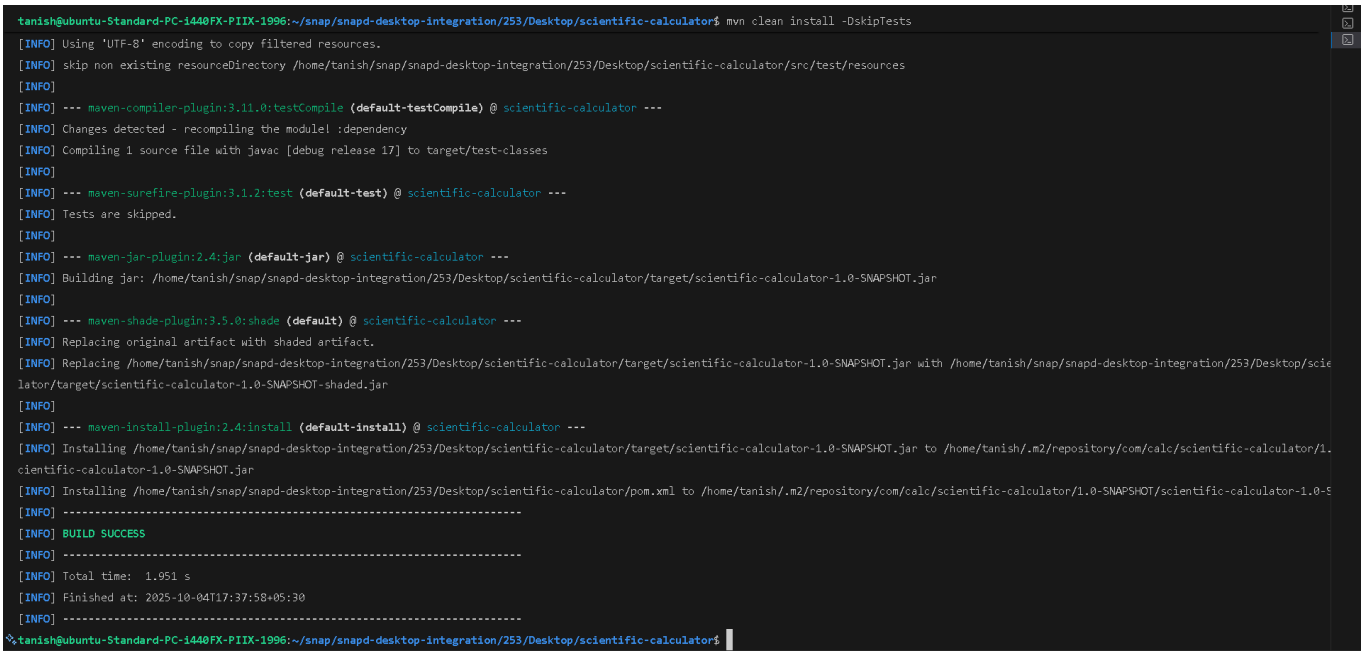

Figure 3: Maven build output (‘mvn clean install‘).

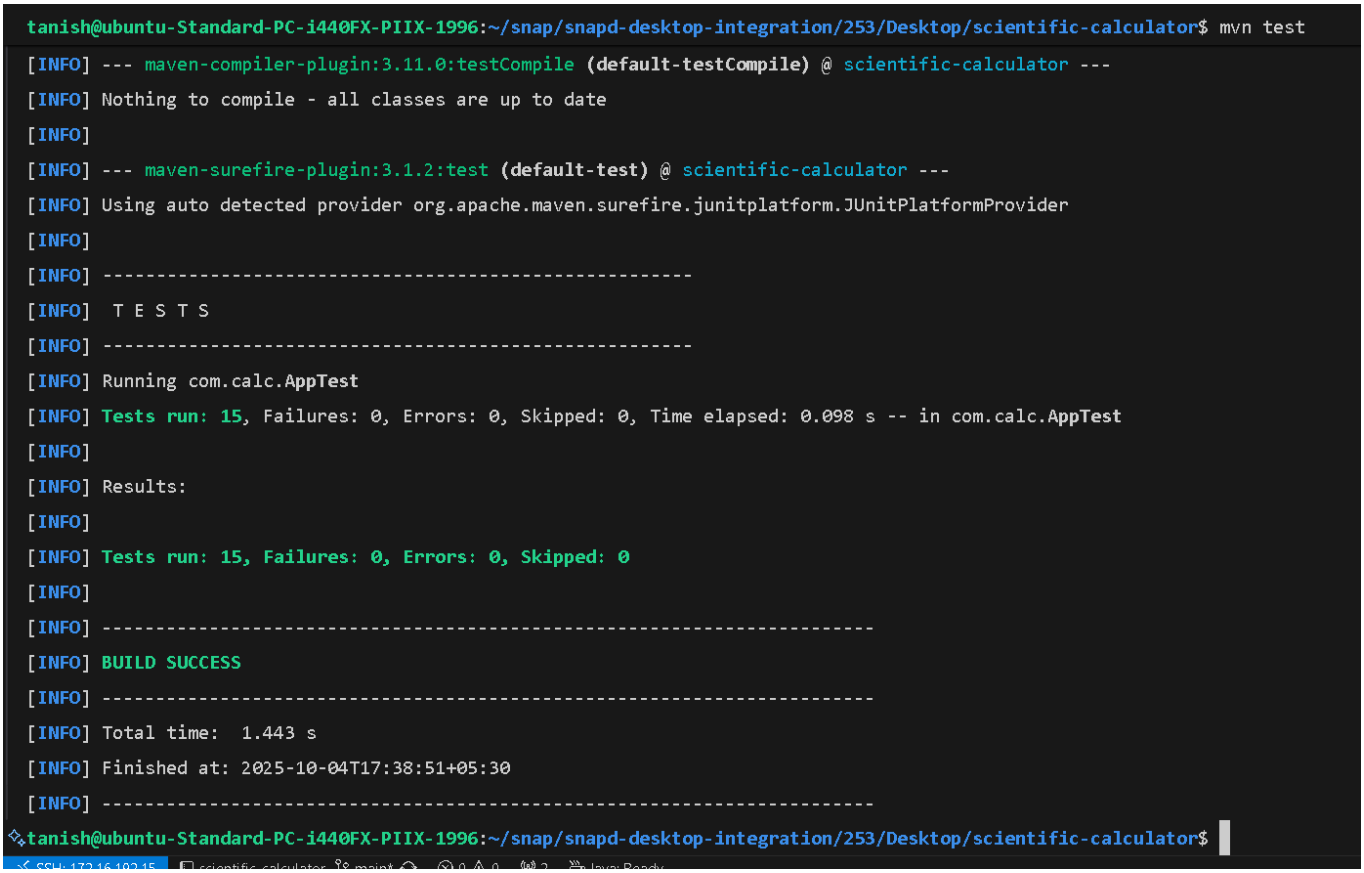

Figure 4: Maven test output (‘mvn test‘).

Test the Project

Run unit tests with:

```
mvn test
```

This executes all test cases in `src/test/java`. Failures or errors are displayed in the terminal, with reports stored in `target/surefire-reports/`.

Clean the Project

To remove all build outputs and start fresh:

```
mvn clean
```

Full Lifecycle Command

Run a complete clean, build, and test process:

```
mvn clean install
```

Verify Build Results

After a successful build, check the `target/` directory. You’ll find:

- Packaged `.jar` or `.war` file.
- Compiled class files.
- Test reports under `surefire-reports/`.

Run the JAR File

To execute the final build output:

```
java -jar target/scientific-calculator-1.0-SNAPSHOT.jar
```

Explanation:

- `java` – Invokes the Java runtime.
- `-jar` – Runs the packaged JAR file.

Running the JAR

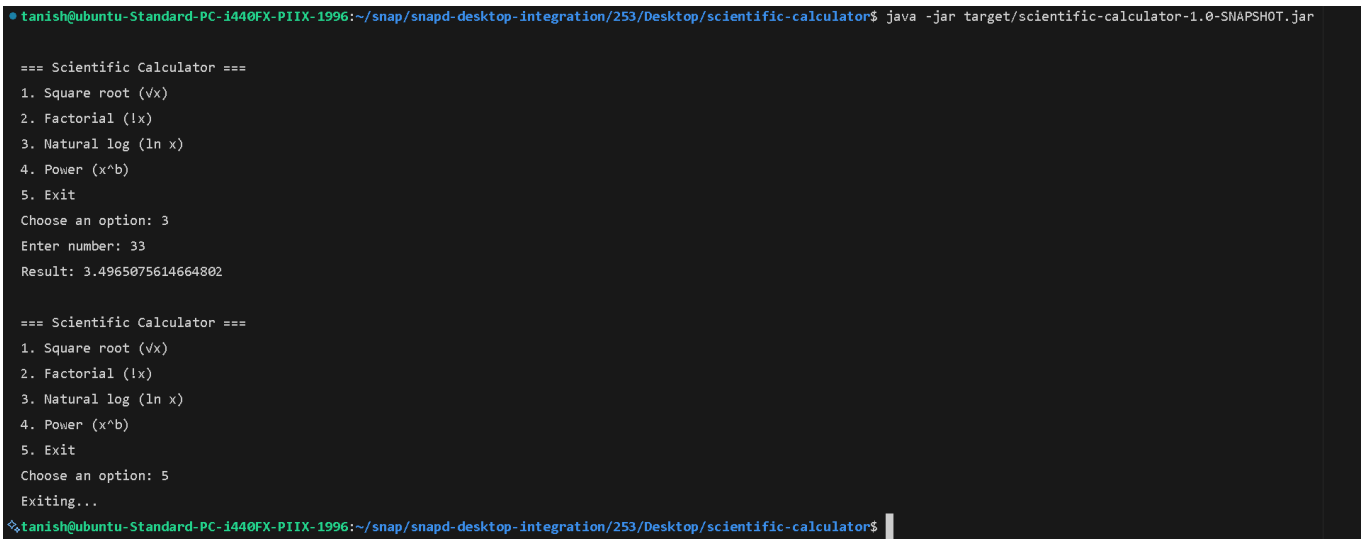

Figure 5: Interactive execution of the calculator JAR file.

10 Setting Up Project with Git and GitHub

This section covers initializing a Git repository, connecting it to GitHub, and pushing project code.

Initialize Git Repository

Navigate to the root directory of your Maven project and run:

```
cd /path/to/your/maven/project
git init
```

Stage and Commit Changes

Stage all project files and commit:

```
git add .
git commit -m "Initial commit for Maven project"
```

Create GitHub Repository

Create a GitHub repository and connect local repository:

```
git remote add origin https://github.com/Tanish-pat/scientific-calculator.git
git push -u origin main
```

Collaborate

After pushing, collaborators can:

- Make changes locally
- Push updates to remote
- Use GitHub pull requests for feature integration

11 Containerizing the Project Using Docker

Containerization ensures consistent deployment across environments.

Dockerfile

```
# Build and Run the application
FROM openjdk:17-jdk-alpine
WORKDIR /app
COPY target/scientific-calculator-1.0-SNAPSHOT.jar /app/calculator.jar
ENTRYPOINT ["java", "-jar", "/app/calculator.jar"]
```

Push Docker Image

Build and push the Docker image to Docker Hub:

```
# Build the Docker image from the Dockerfile in the current directory
docker build -t tanish688/scientific-calculator:1.0 .

# Log in to Docker Hub
docker login -u tanish688

# Push the tagged image to Docker Hub
docker push tanish688/scientific-calculator:1.0
```

This process creates a portable Docker image of the Scientific Calculator application and makes it publicly available for deployment on any host or via configuration management tools such as Ansible.

Docker Workflow

Figure 6: Docker image build.

Figure 7: Running the Docker container and verifying it via `docker ps`.

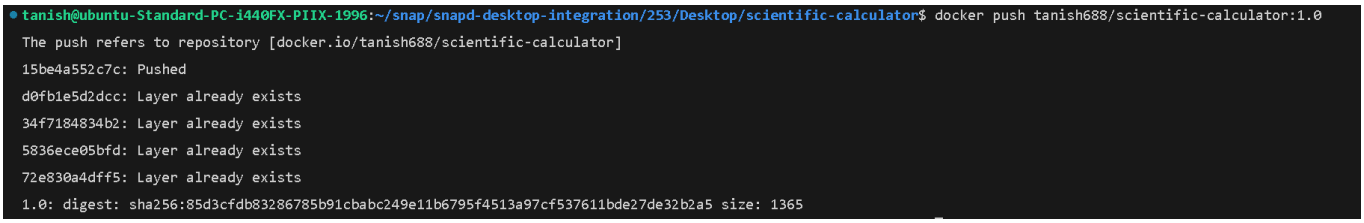

Figure 8: Docker image push to Docker Hub.

12 Create New Jenkins Project

Before starting, ensure credentials have been configured as described earlier.

Create Pipeline

- Open Jenkins dashboard and select **New Item**.
- Name the project (e.g., scientific-calculator-pipeline) and select **Pipeline**.
- Click **OK**.

Configure Pipeline

- **Build Trigger:** Select **GitHub hook trigger for GITScm polling**.
- **Pipeline Script from SCM:** Choose **Git**, enter repository URL and GitHub credentials.
- Specify the Jenkinsfile path if not at root.

Jenkinsfile Pipeline Overview

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = "tanish688/scientific-calculator:1.0"
        DOCKERHUB_CREDENTIALS = "dockerhub-creds"
    }
    stages {
        stage('Checkout') {
            steps {
                git url: 'https://github.com/Tanish-pat/scientific-calculator.git'
            }
        }
        stage('Build & Test') {
            agent {
                docker {
                    image 'maven:3.9.2-eclipse-temurin-17'
                    args '-v /var/run/docker.sock:/var/run/docker.sock'
                }
            }
            steps {
                sh 'mvn clean test && mvn package'
            }
        }
        stage('Prepare Docker Images') {
            steps {
                sh 'docker pull openjdk:17-jdk-alpine || true'
            }
        }
        stage('Docker Build') {
            steps {
                sh "docker build -t ${DOCKER_IMAGE} ."
            }
        }
        stage('Docker Push') {
            steps {
                withCredentials([usernamePassword(credentialsId: "${DOCKERHUB_CREDENTIALS
   }",passwordVariable: 'DOCKER_PASS',usernameVariable: 'DOCKER_USER')]) {
                    sh 'echo $DOCKER_PASS | docker login -u $DOCKER_USER --password-stdin'
                    sh "docker push ${DOCKER_IMAGE}"
                }
            }
        }
        stage('Deploy with Ansible') {
            steps {
                sh 'ansible-playbook -i ansible/hosts.ini ansible/deploy.yml'
            }
        }
        stage('Verify Deployment') {
            steps {
                sh 'docker ps | grep scientific-calculator'
            }
        }
    }
}
```

Pipeline Stages Purpose

- **Check tools:** Verifies Docker and Docker Compose installation.
- **Run tests:** Executes unit tests using Maven; pipeline stops if tests fail.
- **Build Docker Image:** Builds Docker image with latest tag.
- **Login to DockerHub:** Authenticates to push images securely.
- **Push Docker Image:** Pushes Docker image to private Docker Hub repository.
- **Deploy via Ansible:** Deploys application in configured environment.

12.1 Jenkins

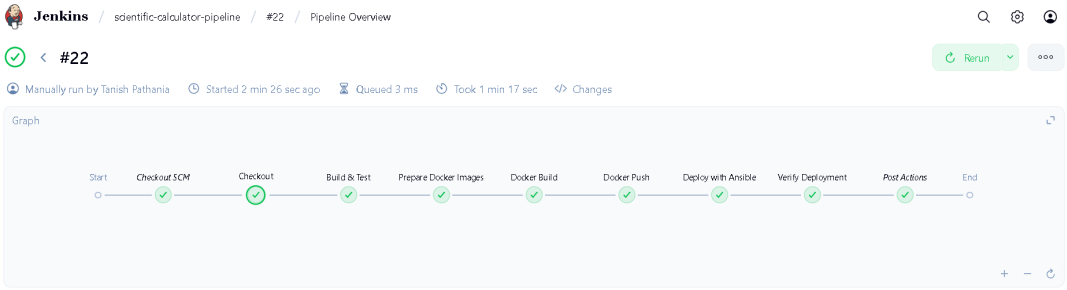

Figure 9: Jenkins pipeline graph.

Figure 10: Jenkins verification of deployment stage.

13 Ansible Setup

To set up Ansible for deploying our application, we need to configure inventory and playbook files, generate SSH keys for secure access, and verify the setup. Below are the detailed steps.

Hosts File (hosts.ini)

The Hosts file specifies the hosts on which the Ansible playbook will run.

```
[local]
localhost ansible_connection=local
```

Explanation:

- **[local]:** Defines a group of hosts named ‘local‘. Playbooks can target this group when executed.
- **localhost:** Specifies the host within the ‘local‘ group. Here, it refers to the local machine.
- **ansible_connection=local:** Tells Ansible to execute tasks directly on the local machine without using SSH.

Playbook File (deploy.yml)

The playbook contains the instructions that Ansible will execute on the specified hosts.

```
---
- name: Deploy Scientific Calculator Docker container
  hosts: local
  tasks:
    - name: Pull Docker image from Docker Hub
      community.docker.docker_image:
        name: tanish688/scientific-calculator
        tag: "1.0"
        source: pull
    - name: Stop and remove existing container if exists
      community.docker.docker_container:
        name: scientific-calculator
        state: absent
    - name: Run the scientific calculator container
      community.docker.docker_container:
        name: scientific-calculator
        image: tanish688/scientific-calculator:1.0
        state: started
        restart_policy: unless-stopped
        tty: yes
        interactive: yes
```

Explanation:

- **name:** Provides a descriptive title for the play or task.
- **hosts: local:** Targets the 'local' host group defined in your hosts.ini file.
- **tasks:** Contains the sequence of operations Ansible will perform on the targeted hosts.
- **community.docker.docker_image:** Pulls the specified Docker image from Docker Hub.
- **community.docker.docker_container:** Manages the Docker container lifecycle: stops/removes any existing container and starts a new one with the specified image, restart policy, and interactive settings.

13.1 Ansible Deployment

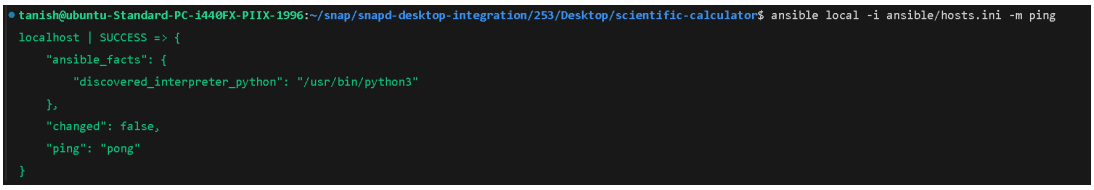

Figure 11: Ansible connectivity test with `ping`.

Figure 12: Ansible playbook deployment output.

Testing the Setup

Verify Ansible connectivity with the inventory using the ping module:

```
ansible local -i hosts.ini -m ping
```

Listing 7: Ping Remote Hosts

If configured correctly, a success message will be returned from each host.

Summary: Setting up Ansible with an inventory and playbook enables automated deployment and configuration on remote servers. SSH keys provide secure, passwordless access, and testing with ping ensures connectivity.

14 Conclusion

This project demonstrates the end-to-end implementation of a DevOps pipeline for a Java-based scientific calculator application. By integrating GitHub, Jenkins, Docker, and Ansible, the workflow achieves automated build, test, containerization, and deployment with minimal manual intervention.

Key takeaways include:

- Streamlined CI/CD pipeline ensures consistent and reliable delivery.
- Docker containers provide environment portability and reduce deployment errors.
- Ansible enables repeatable, automated deployment across multiple environments.
- Jenkins facilitates continuous integration and continuous delivery, improving team efficiency.

Overall, this setup exemplifies modern DevOps practices by combining version control, automated testing, containerization, and infrastructure as code, resulting in faster release cycles, reduced errors, and enhanced software quality.