

# Scientific Calculator with DevOps Pipeline

Tanish Pathania  
Roll Number: IMT2022049

October 9, 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  | Jenkins Credential Configuration                      | 6  |
| 7  | Project Workflow & Setup                              | 7  |
| 8  | Build & Test Project using Maven                      | 7  |
| 9  | Setting Up Project with Git and GitHub                | 10 |
| 10 | Containerizing the Project Using Docker               | 10 |
| 11 | Create New Jenkins Project                            | 11 |
| 12 | GitHub → Jenkins Integration via ngrok and SSH Tunnel | 15 |
| 13 | Ansible Setup   | 17 |
| 14 | Conclusion  | 18 |

# 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

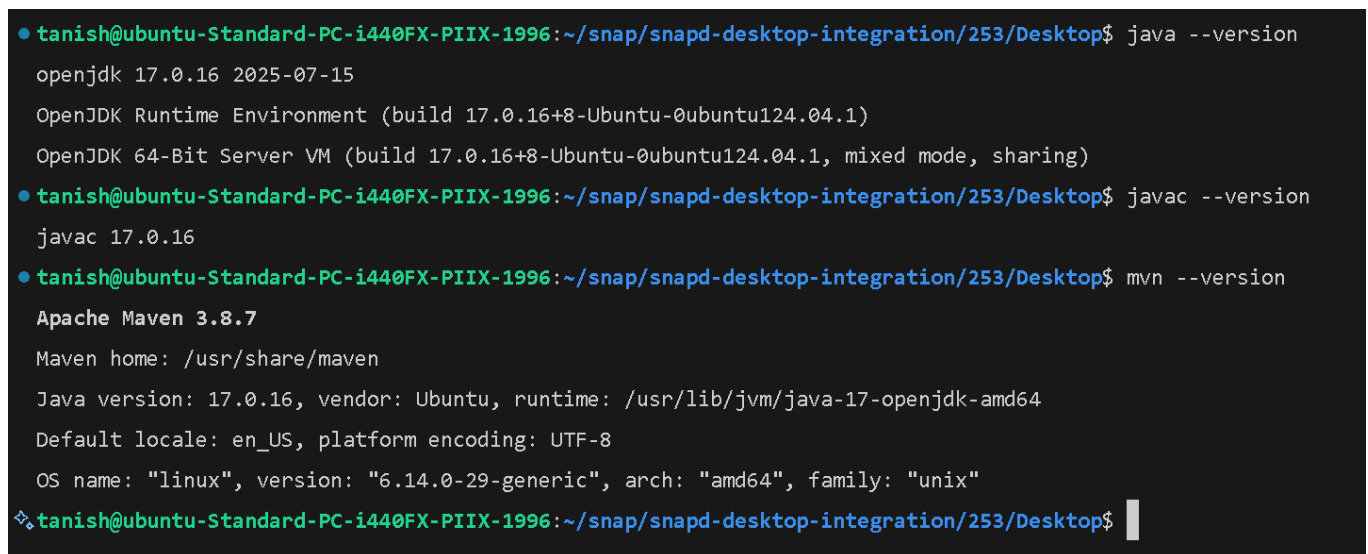
After installation, reload your terminal session:

```
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



```
tanish@ubuntu-standard-pc-i440fx-piix-1996:~/snap/snapd-desktop-integration/253/Desktop$ java --version
openjdk 17.0.16 2025-07-15
OpenJDK Runtime Environment (build 17.0.16+8-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 17.0.16+8-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)

tanish@ubuntu-standard-pc-i440fx-piix-1996:~/snap/snapd-desktop-integration/253/Desktop$ javac --version
javac 17.0.16

tanish@ubuntu-standard-pc-i440fx-piix-1996:~/snap/snapd-desktop-integration/253/Desktop$ mvn --version
Apache Maven 3.8.7
Maven home: /usr/share/maven
Java version: 17.0.16, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.14.0-29-generic", arch: "amd64", family: "unix"

tanish@ubuntu-standard-pc-i440fx-piix-1996:~/snap/snapd-desktop-integration/253/Desktop$
```

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

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop$ java --version
openjdk 17.0.16 2025-07-15
OpenJDK Runtime Environment (build 17.0.16+8-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 17.0.16+8-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop$ javac --version
javac 17.0.16
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop$ mvn --version
Apache Maven 3.8.7
Maven home: /usr/share/maven
Java version: 17.0.16, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.14.0-29-generic", arch: "amd64", family: "unix"
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop$
```

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

## 6 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 —

## 7 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.

## 8 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

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ mvn clean install -DskipTests
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ scientific-calculator ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO] Compiling 1 source file with javac [debug release 17] to target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:3.1.2:test (default-test) @ scientific-calculator ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ scientific-calculator ---
[INFO] Building jar: /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/target/scientific-calculator-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-shade-plugin:3.5.0:shade (default) @ scientific-calculator ---
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/target/scientific-calculator-1.0-SNAPSHOT.jar with /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/target/scientific-calculator-1.0-SNAPSHOT-shaded.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ scientific-calculator ---
[INFO] Installing /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/target/scientific-calculator-1.0-SNAPSHOT.jar to /home/tanish/.m2/repository/com/calc/scientific-calculator/1.0-SNAPSHOT/scientific-calculator-1.0-SNAPSHOT.jar
[INFO] Installing /home/tanish/snap/snapd-desktop-integration/253/Desktop/scientific-calculator/pom.xml to /home/tanish/.m2/repository/com/calc/scientific-calculator/1.0-SNAPSHOT/scientific-calculator-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.951 s
[INFO] Finished at: 2025-10-04T17:37:58+05:30
[INFO] -----
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$
```

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

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ mvn test
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ scientific-calculator ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:3.1.2:test (default-test) @ scientific-calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.calc.AppTest
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.098 s -- in com.calc.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.443 s
[INFO] Finished at: 2025-10-04T17:38:51+05:30
[INFO] -----
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$
```

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

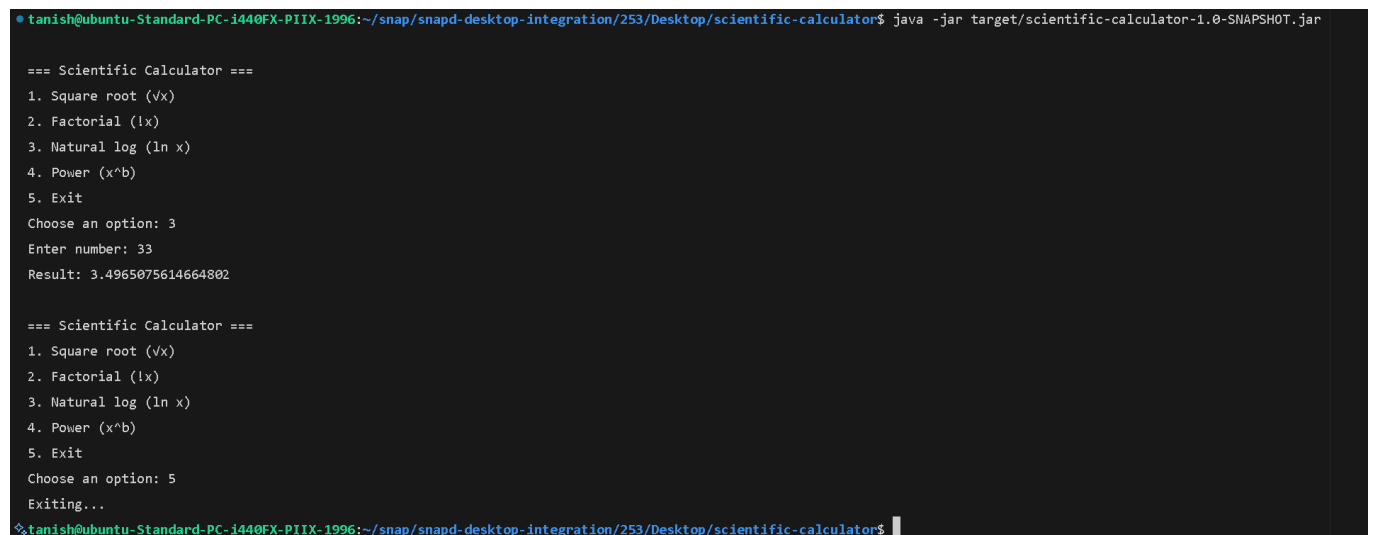
After a successful build, check the `target/` directory. You'll find:

- 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
```



```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ java -jar target/scientific-calculator-1.0-SNAPSHOT.jar

=== Scientific Calculator ===
1. Square root (sqrt)
2. Factorial (l!)
3. Natural log (ln x)
4. Power (x^b)
5. Exit
Choose an option: 3
Enter number: 33
Result: 3.4965075614664802

=== Scientific Calculator ===
1. Square root (sqrt)
2. Factorial (l!)
3. Natural log (ln x)
4. Power (x^b)
5. Exit
Choose an option: 5
Exiting...
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$
```

Figure 5: Interactive execution of the calculator JAR file.

### Explanation:

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

## 9 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

## 10 Containerizing the Project Using Docker

Containerization ensures consistent deployment across environments.

### Dockerfile

```
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:

```
docker build -t tanish688/scientific-calculator:1.0 .
docker login -u tanish688
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

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ docker build -t tanish688/scientific-calculator:1.0 .
[+] Building 3.5s (8/8) FINISHED
=> [internal] load build definition from Dockerfile                                docker:default
=> => transferring dockerfile: 201B                                             0.2s
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine         0.0s
=> [internal] load .dockerignore                                                0.1s
=> => transferring context: 2B                                                  0.0s
=> [1/3] FROM docker.io/library/openjdk:17-jdk-alpine                         0.0s
=> [internal] load build context                                                0.1s
=> => transferring context: 4.25kB                                             0.0s
=> CACHED [2/3] WORKDIR /app                                                  0.0s
=> [3/3] COPY target/scientific-calculator-1.0-SNAPSHOT.jar /app/calculator.jar 1.1s
=> exporting to image                                                         0.8s
=> => exporting layers                                                         0.5s
=> => writing image sha256:90120da0c2b2f23efb4e11769b488d91b78d961142bfdf35f821ada3cee08602 0.0s
=> => naming to docker.io/tanish688/scientific-calculator:1.0                0.1s
```

Figure 6: Docker image build.

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ docker run --rm -it tanish688/scientific-calculator:1.0
=== Scientific Calculator ===
1. Square root (√x)
2. Factorial (!x)
3. Natural log (ln x)
4. Power (x^b)
5. Exit
Choose an option: 4
Enter base: 3
Enter exponent: 5
Result: 243.0

=== Scientific Calculator ===
1. Square root (√x)
2. Factorial (!x)
3. Natural log (ln x)
4. Power (x^b)
5. Exit
Choose an option: 5
Exiting...
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS   NAMES
0c070f0df292   c5f1ef68bd59   "java -jar /app/calcu..." 47 minutes ago Up 47 minutes          scientific-calculator
```

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

```
tanish@ubuntu-Standard-PC-i440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ docker push tanish688/scientific-calculator:1.0
The push refers to repository [docker.io/tanish688/scientific-calculator]
15be4a552c7c: Pushed
d0fb1e5d2dcc: Layer already exists
34f7184834b2: Layer already exists
5836ece05bfd: Layer already exists
72e830a4dff5: Layer already exists
1.0: digest: sha256:85d3cfdb83286785b91cbabc249e11b6795f4513a97cf537611bde27de32b2a5 size: 1365
```

Figure 8: Docker image push to Docker Hub.

## 11 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.

## Pipeline Purpose and Setup

This Jenkins pipeline automates the build, test, and deployment process for the scientific calculator project. The goal is to ensure that every change pushed to the repository is verified, packaged, containerized, and deployed in a consistent environment.

Before creating the pipeline, ensure the following prerequisites are met:

- Jenkins is installed and running with administrator privileges.
- Required plugins are installed, including **Pipeline**, **Git**, **Email Extension**, **Docker Pipeline**, and **Ansible**.
- Credentials are configured under **Manage Jenkins** → **Credentials**:
  - DockerHub username/password for pushing images.
  - GitHub personal access token if using a private repository.
  - SMTP credentials for sending email notifications.
- The repository is accessible and the Jenkinsfile path is correctly specified.

The pipeline is implemented using declarative syntax to allow a clear and structured representation of stages, post-actions, and environment variables. This enhances readability, maintainability, and provides a single point of configuration for notifications and workspace cleanup.

## 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'
    }
}
}

post {
    success { mail to:'Tanish.Pathania@iiitb.ac.in', subject:"Build SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER}", body:"Pipeline succeeded." }
    failure { mail to:'Tanish.Pathania@iiitb.ac.in', subject:"Build FAILURE: ${env.JOB_NAME} #${env.BUILD_NUMBER}", body:"Pipeline failed." }
    always { cleanWs(); echo "Pipeline run over." }
}
}

```

## Pipeline Stage Details

- **Checkout:** Clones the repository from GitHub and ensures the correct branch is checked out. This is the foundation for reproducible builds.

- **Build & Test:** Runs unit tests using Maven. The pipeline will fail if any tests fail, ensuring that only verified code is packaged. Both compilation and test results are captured for later reporting.
- **Prepare Docker Images:** Pulls the base image (`openjdk:17-jdk-alpine`) to ensure a consistent build environment.
- **Docker Build:** Builds the Docker image for the application. Tagging the image ensures version control and traceability.
- **Docker Push:** Authenticates securely with DockerHub and pushes the newly built image. Output logs are captured for review in email notifications.
- **Deploy via Ansible:** Runs an Ansible playbook to deploy the container in the specified environment. The deployment is idempotent and will replace any existing container.
- **Verify Deployment:** Confirms that the Docker container is running and accessible, providing a final verification step in the pipeline.
- **Notifications:** Sends a single email summarizing the pipeline outcome. It includes test results, Docker push logs, and deployment verification. Notifications provide developers with immediate feedback on the build and deployment process.

## Additional Considerations

- **Pipeline Visualization:** Jenkins provides a graphical pipeline view to monitor stage execution times and detect bottlenecks.
- **Troubleshooting Tips:** Common issues include SMTP misconfigurations, Docker login failures, and Ansible deployment errors. Logs from each stage can be referenced in the Jenkins console or captured in the email summary.
- **Future Enhancements:** The pipeline can be extended to:
  - Capture only the last few lines of logs in emails to reduce verbosity.
  - Send Slack or Teams notifications in addition to email.
  - Implement automated rollback if deployment fails.
  - Parallelize long-running test stages to reduce total build time.

These additions provide a comprehensive overview of the Jenkins pipeline, its setup, execution flow, and maintainability considerations.

## 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.
- **Notifications:** Sends email on pipeline success or failure with concise status summary.

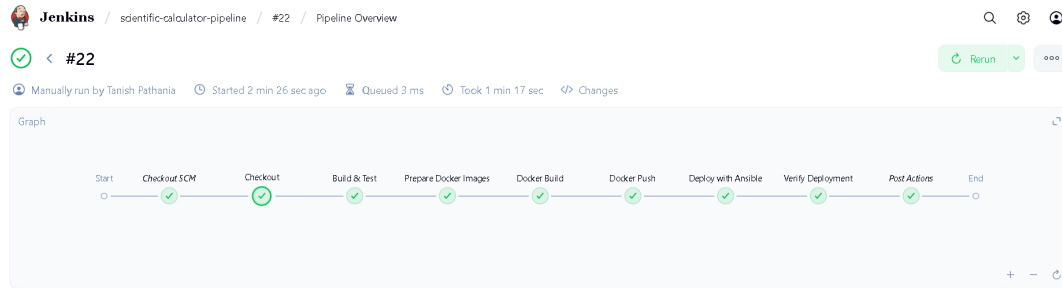


Figure 9: Jenkins pipeline graph.



Figure 10: Jenkins verification of deployment stage.

## 12 GitHub → Jenkins Integration via ngrok and SSH Tunnel

This section demonstrates how GitHub push events trigger Jenkins builds when Jenkins runs locally, using ngrok on a VM and reverse SSH tunneling.

### Step 1 & 2: ngrok on VM and Reverse SSH Tunnel on Laptop

#### Step 1: Start ngrok on VM

Expose the local port 8080 on the VM to the public internet:

```
ngrok http 8080
```

Listing 7: Start ngrok HTTP Tunnel

**Note:** Copy the generated forwarding URL (e.g., <https://278d933fd8da.ngrok-free.app>) for later use.

#### Step 2: Establish Reverse SSH Tunnel from Laptop

From the laptop terminal (Windows PowerShell or CMD), forward traffic from the VM to local Jenkins:

```
ssh -R 8080:localhost:8081 tanish@172.16.192.15
```

Listing 8: Reverse SSH Tunnel

- 8080 → Port on VM exposed via ngrok - 8081 → Jenkins port on laptop

```

ngrok

Call internal services from your gateway: https://ngrok.com/r/http-request

Session Status      online
Account             Tanish Pathania (Plan: Free)
Update              update available (version 3.30.0, Ctrl-U to update)
Version             3.28.0
Region              India (in)
Latency              27ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://278d933fd8da.ngrok-free.app -> http://localhost:8080

Connections
  ttl  opn  rt1  rt5  p50  p90
    3    0   0.00 0.00 30.03 30.07

HTTP Requests
-----
15:12:03.051 IST POST /github-webhook/ 200 OK
15:09:16.990 IST POST /github-webhook/ 200 OK
15:08:32.768 IST POST /github-webhook/ 200 OK

```

Figure 11: ngrok terminal output on the VM.

```

PS C:\Users\offic> ssh -R 8080:localhost:8081 tanish@172.16.192.15
tanish@172.16.192.15's password:
Permission denied, please try again.
tanish@172.16.192.15's password:
Warning: remote port forwarding failed for listen port 8080
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-29-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

8 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

27 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sat Sep 20 22:47:16 2025 from 172.16.133.118

```

Figure 12: Reverse SSH tunnel terminal output on laptop.

## Step 3: Configure GitHub Webhook

In the GitHub repository:

1. Go to **Settings** → **Webhooks** → **Add Webhook**.
2. Set **Payload URL** to the ngrok URL: `https://278d933fd8da.ngrok-free.app/github-webhook/`
3. Content type: `application/json`.
4. Trigger: `push`.
5. Save.

## Step 4: Configure Jenkins URL and Job Trigger

1. Navigate to **Manage Jenkins** → **Configure System**.
2. Set **Jenkins URL**: `https://278d933fd8da.ngrok-free.app/`
3. For the pipeline job, enable: *"GitHub hook trigger for GITScm polling"*

## Step 5: Test the End-to-End Flow

1. Make a code change in the local repository.
2. Stage, commit, and push:
3. Verify on GitHub Webhooks that the request shows a 200 OK response.
4. Confirm Jenkins automatically starts a new build.

**Result:** After these steps, GitHub push events flow through:

GitHub → ngrok (VM) → SSH Tunnel → Local Jenkins (Laptop)

triggering automatic pipeline builds.



## 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.

# Ansible Deployment

```
tanish@ubuntu-Standard-PC-1440FX-PIIX-1996:~/snap/snapd-desktop-integration/253/Desktop/scientific-calculator$ ansible local -i ansible/hosts.ini -m ping
localhost | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Figure 13: Ansible connectivity test with ping.

## Testing the Setup

Verify Ansible connectivity with the inventory using the ping module:

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

Listing 9: 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 an end-to-end DevOps pipeline for a Java-based scientific calculator. By integrating GitHub, Jenkins, Docker, and Ansible, the workflow automates build, test, containerization, and deployment.

Key takeaways:

- CI/CD pipeline ensures consistent and reliable delivery.
- Docker containers provide environment portability and reduce deployment errors.
- Ansible enables repeatable automated deployment.
- Jenkins improves integration and delivery efficiency.

This setup exemplifies modern DevOps practices, enabling faster releases, fewer errors, and higher software quality.