CS 816 - Software Production Engineering
Mini Project - Scientific Calculator with DevOps

Trupti Khodwe (IMT2022007)

GitHub Repository: Scientific Calculator Github IMT2022007
Docker Repository: Scientific calculator Docker IMT2022007

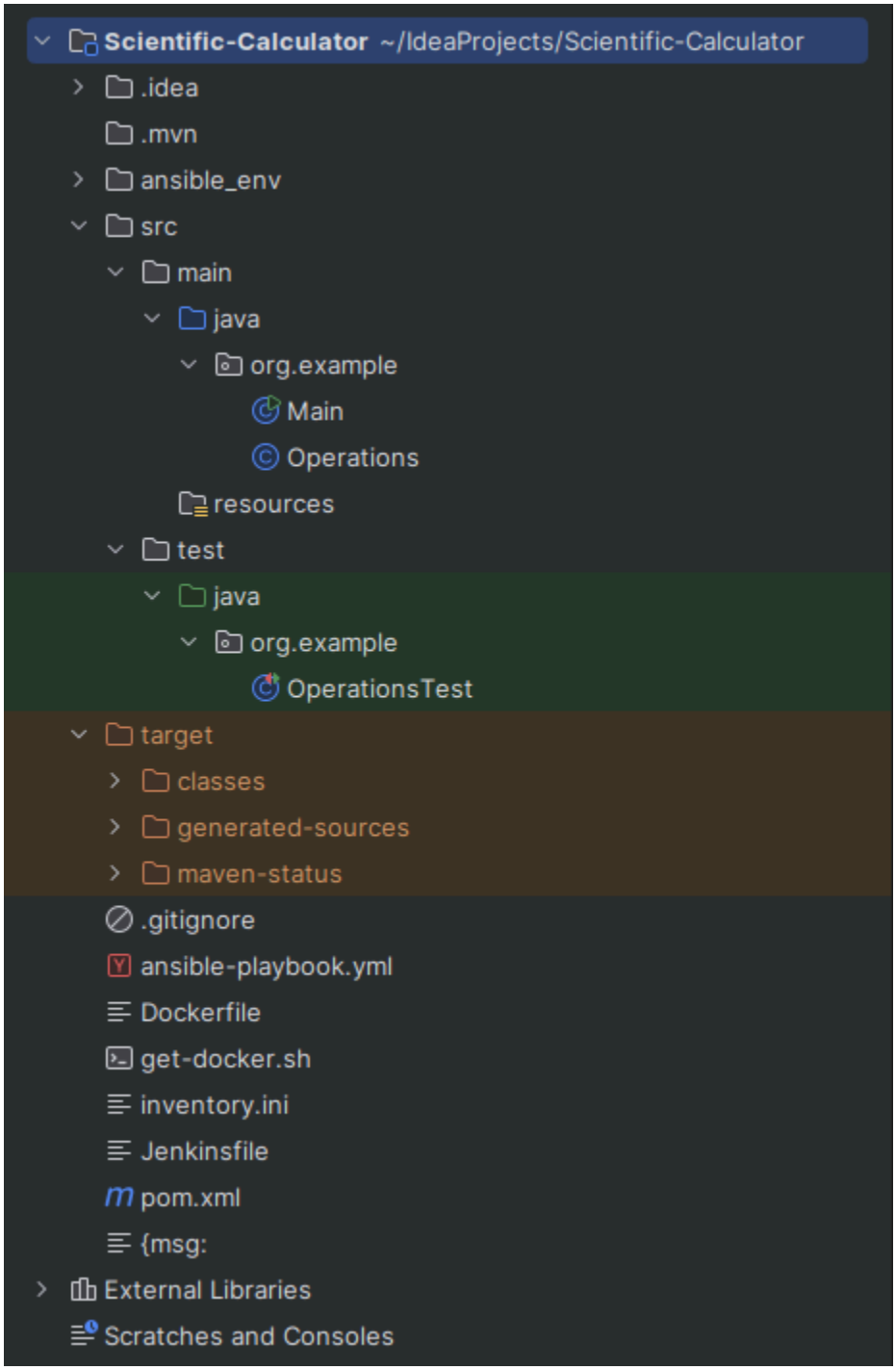
This project uses a DevOps pipeline to demonstrate the principles of Continuous Integration and Continuous Delivery/deployment (CI and CD) by automating the build, test, and deployment of a simple scientific calculator .

Tools used

Name	Category	Role in project
GitHub	Source Control Management (SCM)	Stores the source code and tracks changes.
JUnit	Testing	A framework to write and run automated unit tests for the calculator's functions.
Maven	Build Automation	Compiles the Java code and packages it into a distributable file.
Jenkins	Continuous Integration (CI)	An automation server that runs the pipeline to build, test, and containerize the application.
Docker	Containerization	Packages the application and its dependencies into a portable container.
Docker Hub	Registry	A public repository to store and share the Docker container image.
Ngrok	Networking/Tunneling	Provides a secure tunnel that enables public port forwarding , allowing the GitHub webhook to trigger CI.
Ansible	Configuration Management & Deployment	Automates the deployment of the Docker container on the managed host.
IntelliJ	Integrated Development	Used for writing the Java source code, organizing

	Environment (IDE)	project structure, and local testing.

Directory Structure



Workflow

A. Phase - 1: Source code, Testing & Building:

1. Coding the calculator:

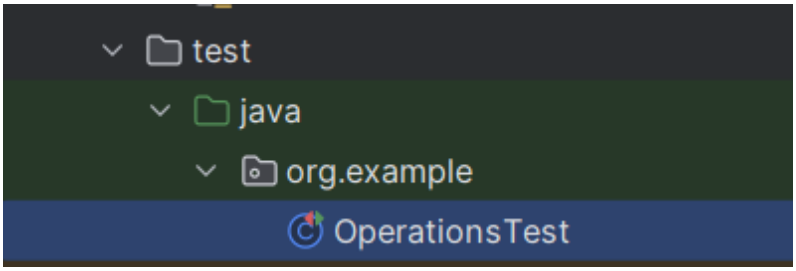
a. Operations.java:

This file contains all the operations to be performed (square root, power etc.). These operations also handle invalid cases and throw an IllegalArgumentException wherever needed.

b. Main.java:

This file contains the source code for displaying the menu and calling the corresponding operations from Operations.java based on user inputs.

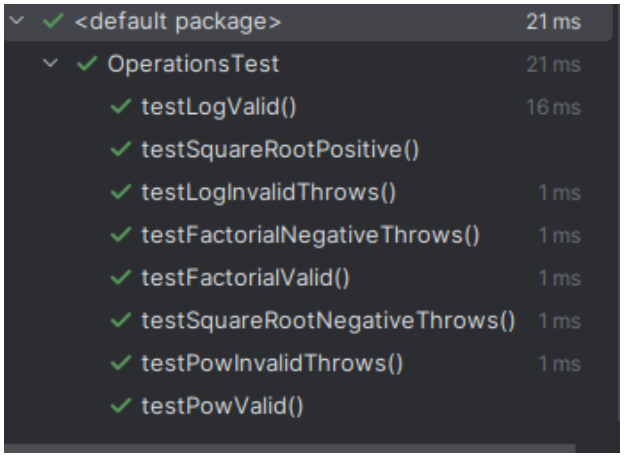
2. Writing unit tests, using JUnit 5:



JUnit was used to write unit tests for the calculator functions. The tests were made to cover different cases, including negative testing to check that important operations—like √x with negative numbers or ln(x) with zero—throw the right exceptions.

IntelliJ makes it easy to run JUnit tests, so I just added @Test annotations to the methods and ran all the tests together with one click or by using the Maven test command.

```
[INFO] Compiling 1 source file with javac [debug target 17] to target/test-classes
[INFO]
[INFO] --- surefire:3.2.5: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 org.example.OperationsTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.058 s -- in org.example.OperationsTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.661 s
[INFO] Finished at: 2025-10-10T17:57:29+05:30
[INFO] ------------------------------------------------------------------------
```



3. **Pushing source code to SCM tool (e.g., GitHub)**
   After completion of the source code, the changes were committed and pushed to the GitHub repository.
   Instead of using Git commands, I utilized the Version Control/Git features provided by IntelliJ. This made it easier and faster to perform commit, push etc., making the CI verification part faster as well.

4. **Using Maven for Build**
   Maven automates building Java projects by managing dependencies and plugins through the pom.xml file. It also keeps a standard folder structure with directories like src/main/java and src/test/java.

The Maven Shade Plugin was used to create a single executable (fat) JAR file called scientific-calculator-executable.jar. This helps avoid runtime dependency issues when running the app inside a container.

B. Phase - 2: Continuous Integration and Containerization:

1. CI Tool Setup - Jenkins:

Jenkins was installed on a local Ubuntu environment (linux) to serve as the CI orchestration engine. Jenkins automates the integration and deployment of code, ensuring that all changes are tested and deployed seamlessly.

- JDK 17 and Maven 3 were installed and configured in Manage Jenkins > Tools.
- The Email Extension Plugin was configured for notification.
- Docker Hub login credentials were stored securely in Jenkins Credentials Manager.
- The pipeline definition was stored in a Jenkinsfile in the project root, enabling the **Stage View** visualization.

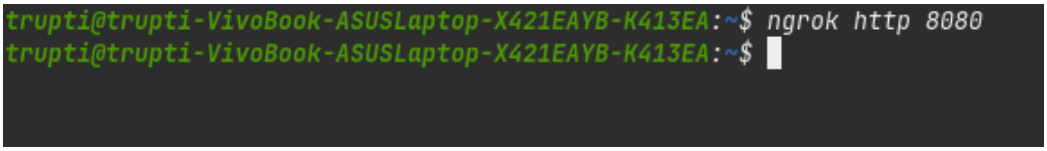
The initial part of Jenkinsfile contains tool definitions and environment configurations (DockerHub username, Docker credential id, GitHub url etc.). After this it defines the entire CI/CD workflow through its structured stages. These stages cover the complete process from code commit to container image availability:

- **Pull GitHub Repo**: This stage pulls the latest source code from the GitHub repository, which is triggered by the webhook upon a code push.
- **Run Test and Build**: This stage executes the Maven build command (mvn clean install), which first runs all **JUnit test cases** and then compiles and packages the Java code into the executable JAR artifact.
- **Build Docker Image**: This stage uses the Dockerfile to create a final, portable Docker image containing the tested application artifact.
- **Push to Docker Hub**: This final CI stage authenticates with Docker Hub and pushes the newly created image, making the tested artifact available for deployment.

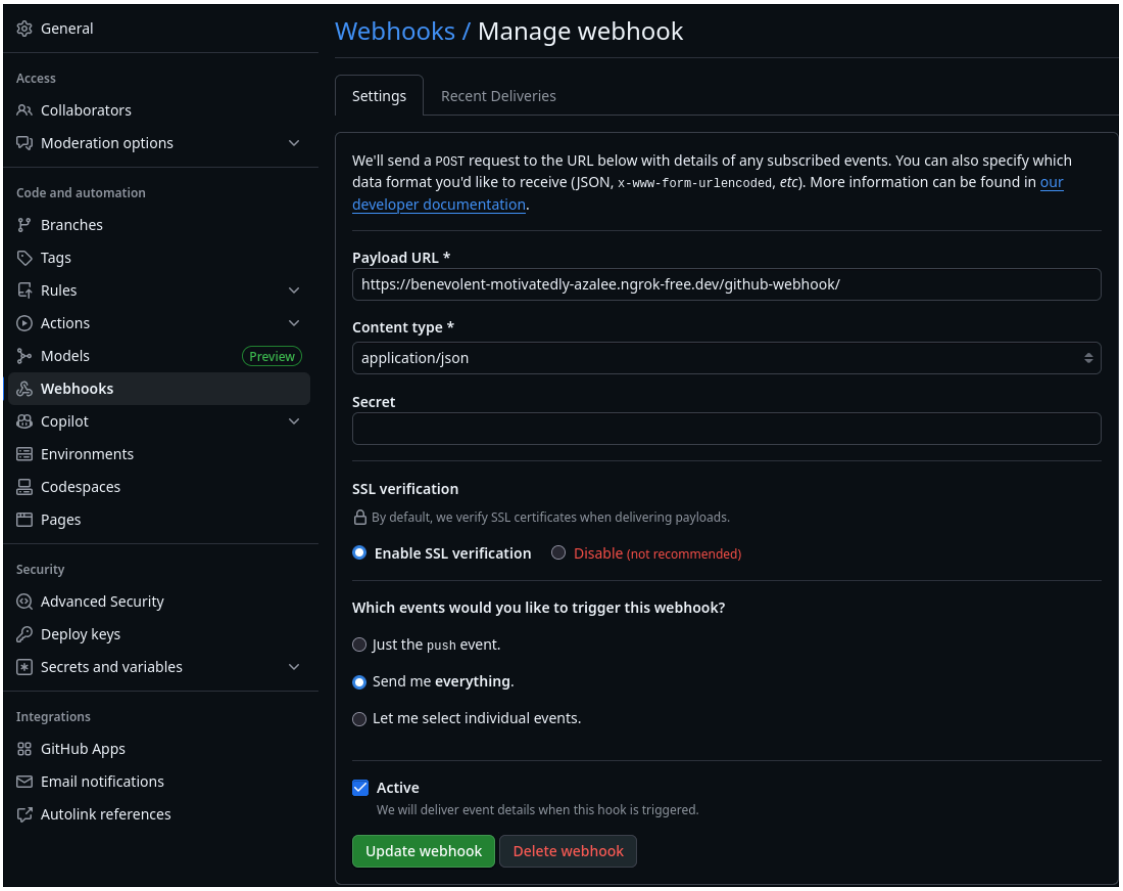
2. Webhook Trigger and Pipeline Execution:

To satisfy the requirement for automatic triggering, the following was implemented:

- **Ngrok Tunnel:** Due to the local network setup, Ngrok was used to create a temporary public HTTPS URL, allowing GitHub to reach the local Jenkins server.



- **GitHub Webhook:** The GitHub repository was configured with a webhook pointing to the active Ngrok URL.



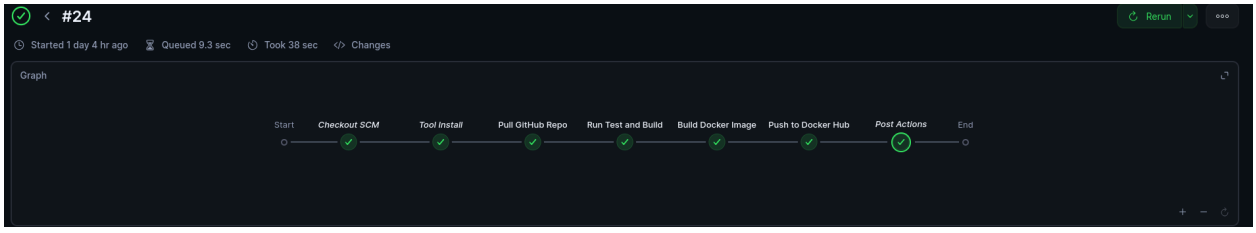
3. **Containerization - Docker**
   Docker was used to create the image. A multi-stage build pattern was used for efficiency and security.

   Key Configurations:
   - Build Stage: Uses maven:3.9.9 for compilation.
   - Runtime Stage: Uses the lightweight eclipse-temurin:21-jre-alpine for the final image.
   - Fix: The COPY command was explicitly targeted at the executable JAR: scientific-calculator-executable.jar.

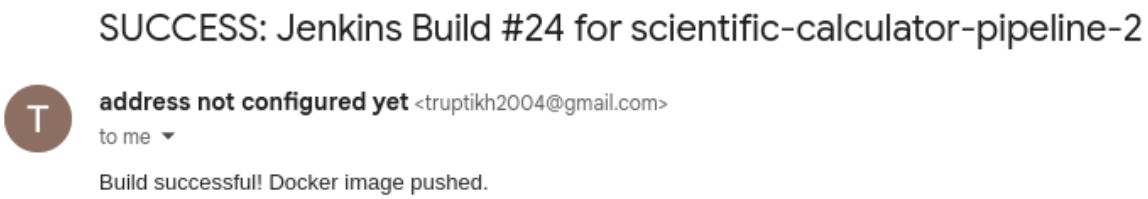
   These configurations were coded in the Dockerfile placed in the root directory.

4. **Docker Hub Registry:**
   The pipeline pushed the final image to a public Docker Hub repository.

(DockerHub Repository)



(Build Pipeline Stage View - Executed each time Git notifies of a change)



(Email Notification: About Build Success/Failure)

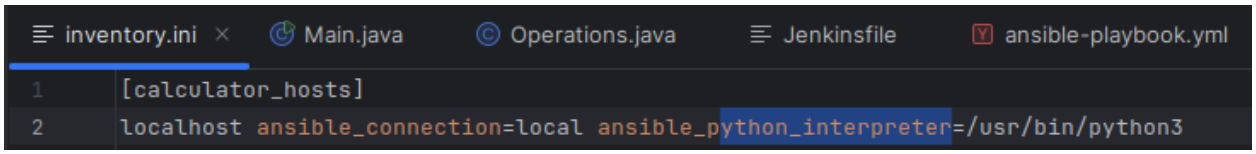
C. Phase - 3: Deployment with Configuration Management:

1. Deployment Tool - Ansible:

Ansible was used for Configuration Management and Deployment. The deployment targets the local machine (running Docker in WSL) as the managed host.

Key Configurations:

- Prerequisites: Ansible and the community.docker collection was installed.
- Inventory: The inventory.ini file defines the local machine using a local connection.



2. Deployment Script (Ansible Playbook):

The playbook ensures Docker is running, removes any old containers, pulls the latest image, and runs the container in a persistent, interactive mode (tty: yes, interactive: yes) necessary for the Java CLI application to accept input.

The ansible-playbook.yml file was also placed in the root directory and stores all the necessary details needed for deployment of the Docker image.

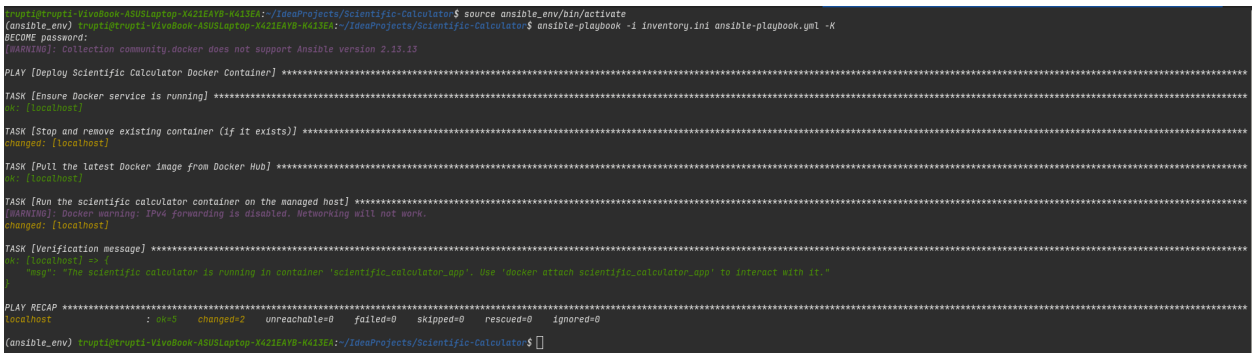
3. Deployment Command:

For me, the deployment of the app was only happening in a separate python environment because there were conflicts in the kernel version.

Commands:

source ansible_env/bin/activate

ansible-playbook -i inventory.ini ansible-playbook.yml -K



Checking deployment status:

```
trupti@trupti-VivoBook-ASUSLaptop-X421EAYB-K413EA:~/IdeaProjects/Scientific-Calculator$ sudo docker ps
[sudo] password for trupti:
CONTAINER ID   IMAGE                                      COMMAND              CREATED              STATUS              PORTS     NAMES
c47db7cb6ba8   trupti1812/scientific-calculator:latest   "java -jar app.jar"   About a minute ago   Up About a minute             scientific_calculator_app
trupti@trupti-VivoBook-ASUSLaptop-X421EAYB-K413EA:~/IdeaProjects/Scientific-Calculator$
```

D. Phase - 4: Application Verification and Conclusion:

1. Application Verification:

The application was verified by executing commands within the running container and demonstrating all four required scientific operations.

Verification Command:

sudo docker run -it --rm trupti1812/scientific-calculator:latest

OUTPUT:

```
trupti@trupti-VivoBook-ASUSLaptop-X421EAYB-K413EA:~/IdeaProjects/Scientific-Calculator$ sudo docker run -it --rm trupti1812/scientific-calculator:latest
WARNING: IPv4 forwarding is disabled. Networking will not work.
Calculator is ready. Attach and press Enter to begin.


----- Scientific Calculator Menu -----
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
Enter your choice (1-5): 2
Enter a non-negative integer (x): 4
Output: 24.0

----- Scientific Calculator Menu -----
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
Enter your choice (1-5): 1
Enter number (x): -9
Error: x must be >= 0

----- Scientific Calculator Menu -----
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
Enter your choice (1-5): 3
Enter number (x): 0
Error: x must be > 0

----- Scientific Calculator Menu -----
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
Enter your choice (1-5): 3
Enter number (x): 7
Output: 1.945910149055313 2

----- Scientific Calculator Menu -----
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
Enter your choice (1-5): 5
Thanks for visiting!
trupti@trupti-VivoBook-ASUSLaptop-X421EAYB-K413EA:~/IdeaProjects/Scientific-Calculator$
```