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

Rishabh Dixit (IMT2022051)

---

**Problem Statement**

Create a scientific calculator program with user menu driven operations:
- Square root function - √x
- Factorial function - !x
- Natural logarithm (base - e) ln(x)
- Power function - x^b

**What and Why of DevOps?**

DevOps is the integration and automation of software development and information technology operations. The main objective is to eliminate barriers between development, testing, and operations teams by automating the entire software delivery process.

It focuses on collaboration, automation, and continuous feedback across key phases of planning, coding, building, testing, releasing, deploying, operating, and monitoring executed in a continuous loop.

- **Continuous Integration (CI):** The practice of merging all developers' working copies to a shared mainline several times a day. The main goal is to detect integration errors as quickly as possible.
- **Continuous Delivery/Deployment (CD):** A software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.

This project uses a DevOps pipeline to demonstrate these principles by automating the build, test, and deployment of a simple scientific calculator .

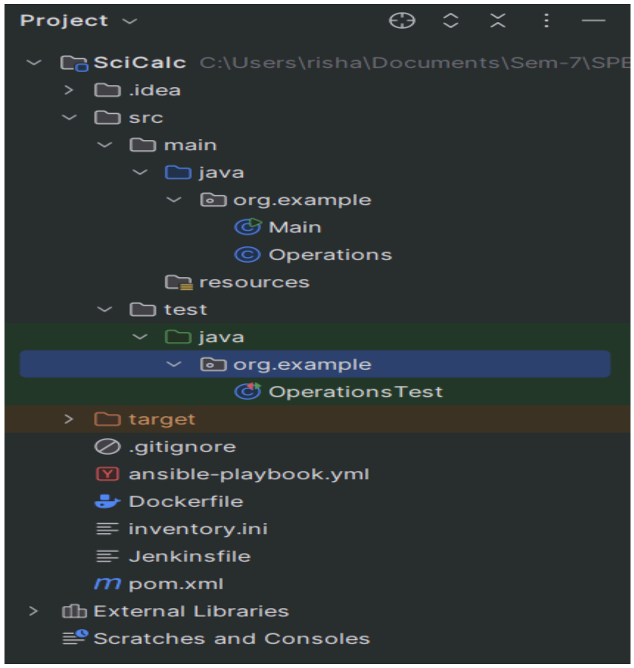
**Important Links:**

1. **GitHub Repository: Scientific Calculator**
2. **Docker Hub Repository: 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 Environment (IDE) | Used for writing the Java source code, organizing project structure, and local testing. |

**Directory Structure**




# Workflow

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

### 1. Coding the calculator:

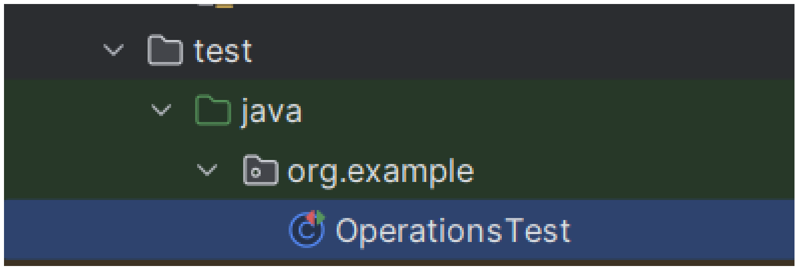
**a. Operations.java:**

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:**

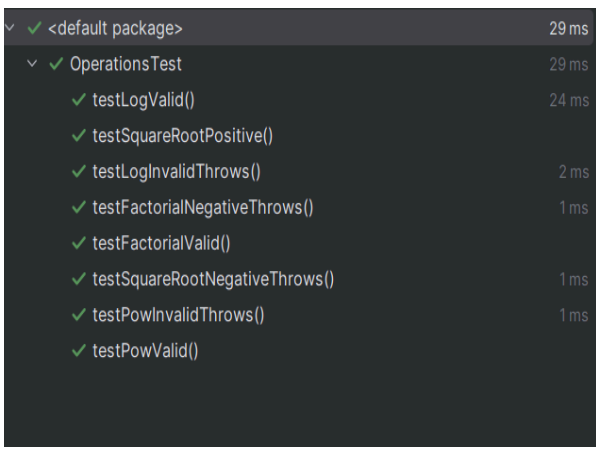
This file contains the source code for displaying the menu and calling the corresponding operations from `Operations.java` based on user inputs.

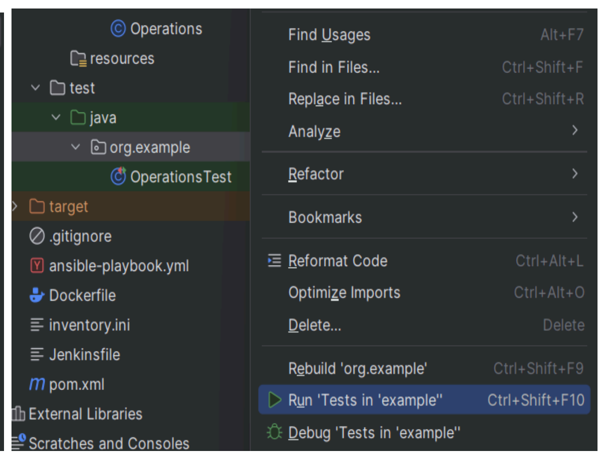
### 2. Writing unit tests, using JUnit 5:

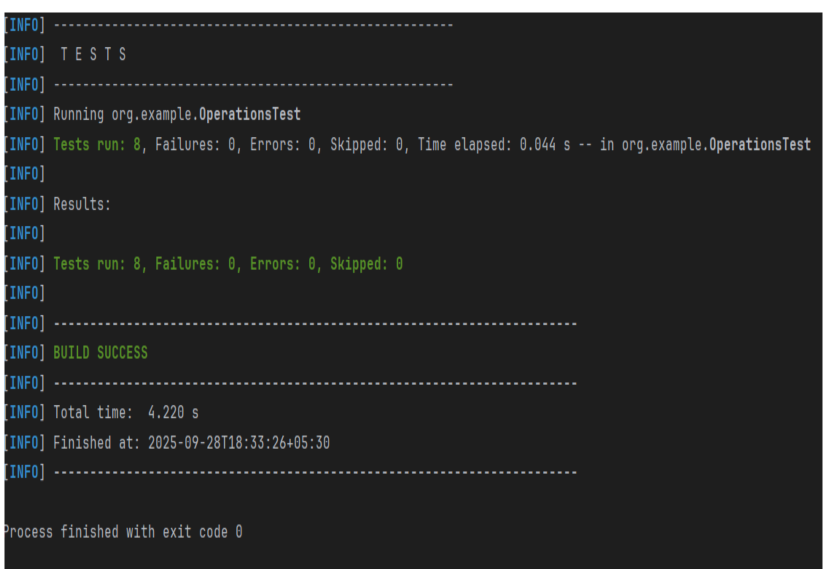



JUnit was used to write unit tests for the calculator operations. Test cases were designed to ensure a good code coverage, specifically employing negative testing to assert that critical operations—like √x for negative inputs or ln(x) for zero—correctly throw expected mathematical exceptions.

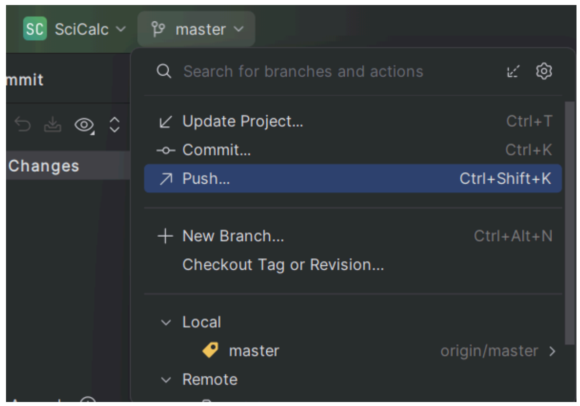
IntelliJ provides extensive features to perform unit tests using JUnit. So I just had to write several `@Test` annotations and then run the entire test-suite with a single click or simply clicking on *Maven test*.



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 the process of building Java applications, handling dependencies and plugins via the `pom.xml` file. It standardizes the repo structure with directories like *src/main/java* and *src/test/java*.

   The Maven **Shade Plugin** was added to create a single, executable (fat) JAR file named *scientific-calculator-executable.jar.* This resolves runtime dependency issues within the container.

## B. Phase - 2: Continuous Integration and Containerization:

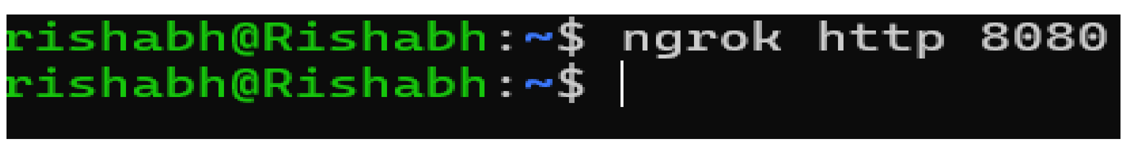
1. **CI Tool Setup - Jenkins:**
   Jenkins was installed on a local Ubuntu environment (via WSL) to serve as the CI orchestration engine. Jenkins automates the integration and deployment of code, ensuring that all changes are tested and deployed seamlessly.
   - JDK 21 and Maven 3 were installed and configured in Manage Jenkins > Tools.
   - The Email Extension Plugin was configured for notification.

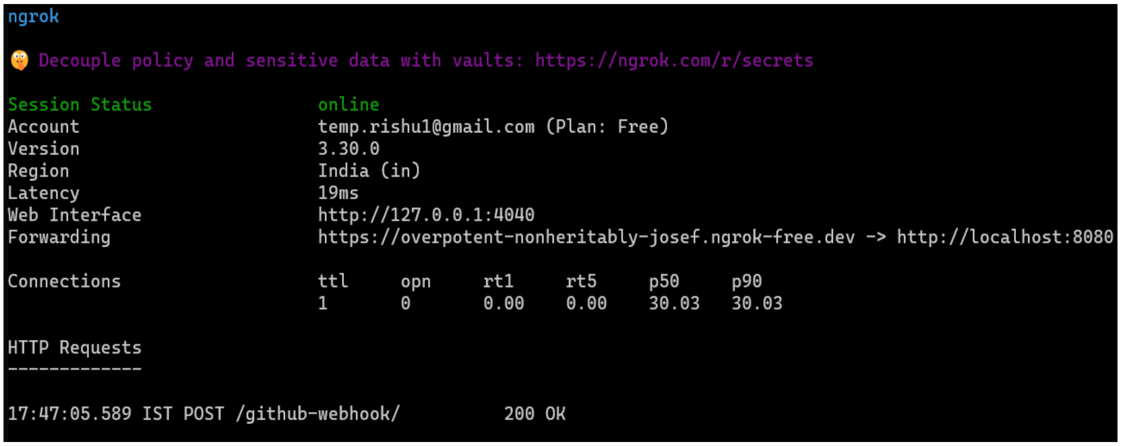
- Docker Hub login credentials were stored securely in Jenkins Credentials Manager.
- The pipeline definition was stored in a ***Jenkinsfile*** in the project root, enabling the **Stage View** visualization.

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:

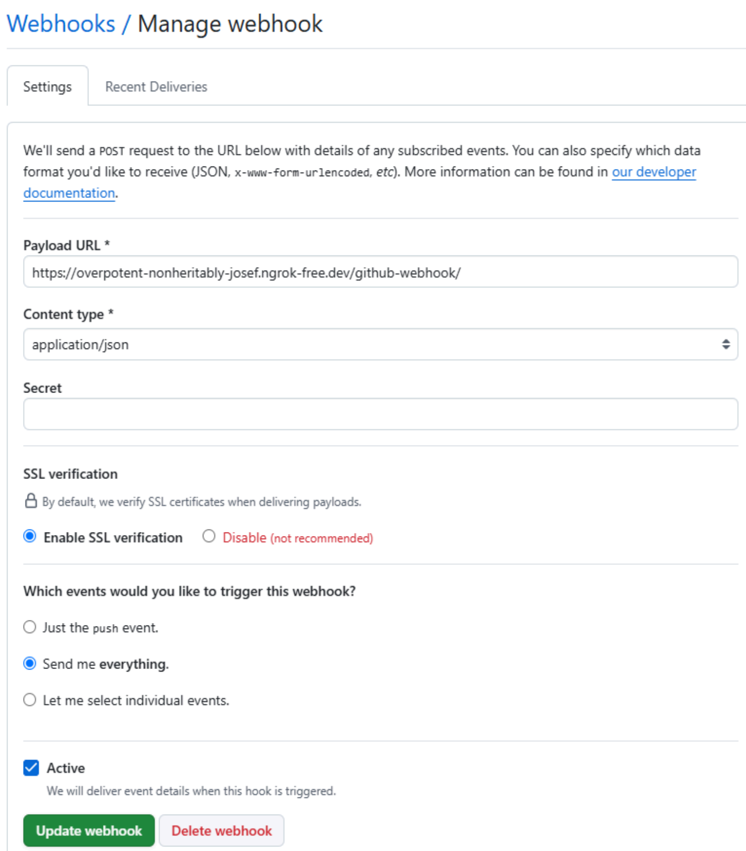
- **Pull GitHub Repo**: This stage pulls the latest source code from the GitHub repository, which is triggered by the webhook upon a code push.
- **Run Test and Build**: This stage executes the Maven build command (`mvn clean install`), which first runs all **JUnit test cases** and then compiles and 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 (WSL), 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.

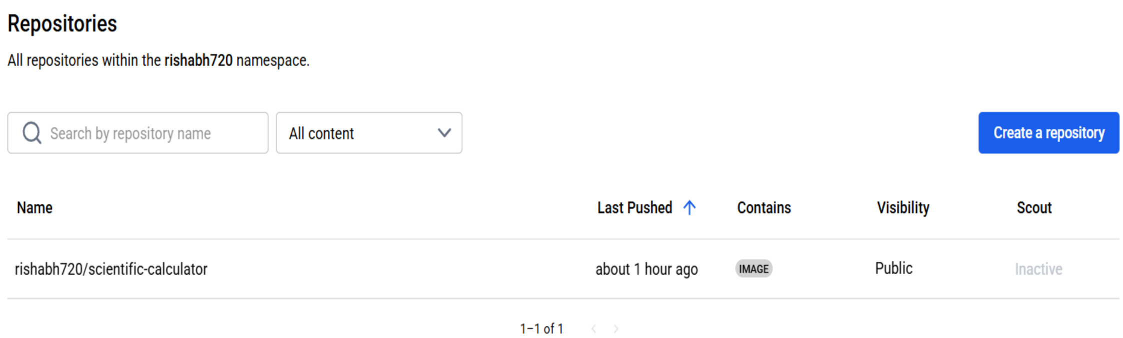
   Key Configurations:

   - Build Stage: Uses maven:3.9.9-eclipse-temurin-21 for compilation.

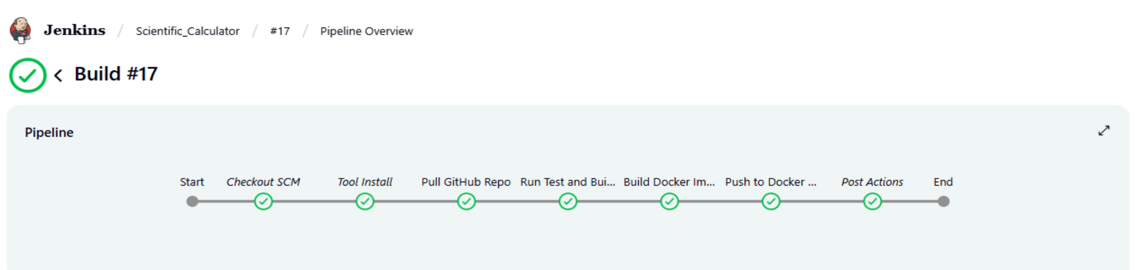
- Runtime Stage: Uses the lightweight eclipse-temurin:21-jre-alpine for the final image.
- Fix: The COPY command was explicitly targeted at the executable JAR: ***scientific-calculator-executable.jar.***

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

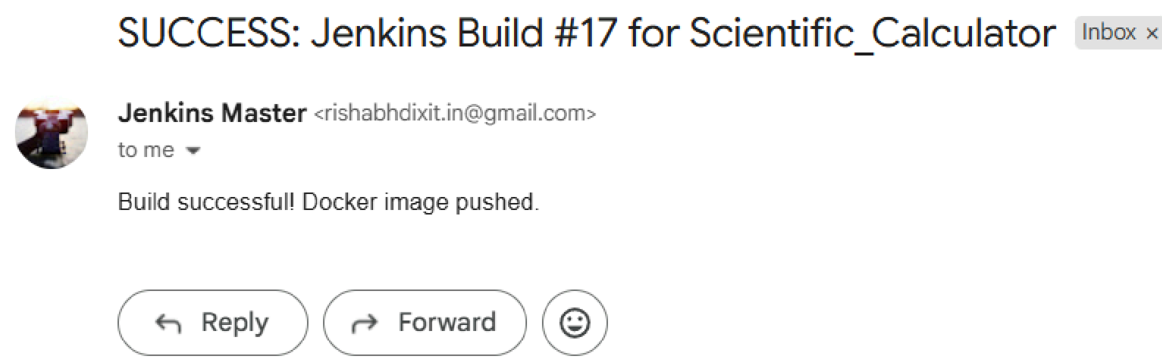
4. **Docker Hub Registry:**
   The pipeline pushed the final image to a public Docker Hub repository.




(DockerHub Repository)




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




(Email Notification: About Build Success/Failure)

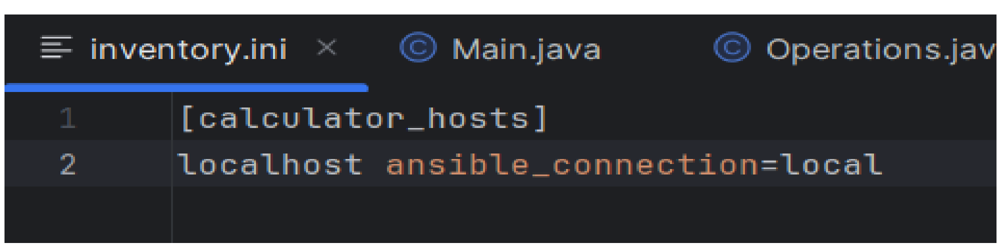
## C. Phase - 3: Deployment with Configuration Management:

### 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 were installed in WSL.
- 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.

```yaml
---
- name: Deploy Scientific Calculator Docker Container
  hosts: calculator_hosts
  gather_facts: no

  vars:
    docker_image: "rishabh720/scientific-calculator:latest"
    container_name: "scientific_calculator_app"

  tasks:
    - name: Ensure Docker service is running
      ansible.builtin.service:
        name: docker
        state: started
      become: yes

    - name: Stop and remove existing container (if it exists)
      community.docker.docker_container:
        name: "{{ container_name }}"
        state: absent
        force_kill: yes
      ignore_errors: yes
      become: yes
```

3. **Deployment Command:**
   ansible-playbook -i inventory.ini ansible-playbook.yml -K, followed by the WSL password.

```
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$ ansible-playbook -i inventory.ini ansible-playbook.yml -K
BECOME password:

PLAY [Deploy Scientific Calculator Docker Container] ****************************************************************************************************

TASK [Ensure Docker service is running] ****************************************************************************************************************
[WARNING]: Platform linux on host localhost is using the discovered Python interpreter at /usr/bin/python3.10, but future installation of another Python interpreter could change
the meaning of that path. See https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]

TASK [Stop and remove existing container (if it exists)] ***********************************************************************************************
changed: [localhost]

TASK [Pull the latest Docker image from Docker Hub] ****************************************************************************************************
ok: [localhost]

TASK [Run the scientific calculator container on the managed host] *************************************************************************************
changed: [localhost]

TASK [Verification message] ****************************************************************************************************************************
ok: [localhost] => {
    "msg": "The scientific calculator is running in container 'scientific_calculator_app'. Use 'docker logs scientific_calculator_app' to interact."
}

PLAY RECAP *********************************************************************************************************************************************
localhost                  : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Checking deployment status:

```
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$ docker ps
CONTAINER ID   IMAGE                                    COMMAND              CREATED         STATUS         PORTS     NAMES
64a35f1f18d4   rishabh720/scientific-calculator:latest   "java -jar app.jar"   2 minutes ago   Up 2 minutes             scientific_calculator_app
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$ docker logs scientific_calculator_app
Scientific Calculator Menu
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^b)
5. Exit
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$
```

## D. <u>Phase - 4: Application Verification and Conclusion:</u>

1. **Application Verification:**
   The application was verified by executing commands within the running container and demonstrating all four required scientific operations.

   Verification Command:

docker exec - it scientific_calculator_app java - jar app . jar

**OUTPUT:**

```
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$ docker exec -it scientific_calculator_app java -jar app.jar
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):
16
Output: 4.0
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): 5
Thanks for visiting!
rishabh@Rishabh:/mnt/c/Users/risha/Documents/Sem-7/SPE/SciCalc$
```

2. **Conclusion:**
This project successfully implemented a complete end-to-end DevOps pipeline for a Java Scientific Calculator. Despite initial challenges related to environment variable configuration in the WSL environment, the pipeline now reliably and automatically builds, tests, containers, and deploys the application following every code push to GitHub.

Whenever we perform a GitHub push, the configured webhook is triggered, which instantly notifies Jenkins to pull the latest code, run the full test and build pipeline, containerize the application, and push the resulting image to Docker Hub, completing the Continuous Integration (CI) cycle.

After this we can use Ansible playbook to retrieve the container and run the application on the local machine.