SPE Mini Project

Bhavya Kapadia
IMT2022095

October 10, 2025

1 Links

GitHub repository: `https://github.com/BKAPADIA04/calculator-spe-mini-project`

Docker Hub image: `https://hub.docker.com/repository/docker/bkapadia04/calculator-spe-mini-project/general`

2 Introduction

2.1 What is DevOps

DevOps is a modern evolution in software development practices that builds upon earlier methodologies such as **Agile** and the now largely outdated **Waterfall model**. These earlier paradigms contributed significantly to improving the efficiency and effectiveness of software engineering processes.

The Waterfall model followed a strictly linear, sequential approach, whereas Agile introduced shorter and faster development cycles, enabling teams to progress through the programming stages more efficiently. However, even Agile, despite fostering stronger collaboration between developers and clients, still maintains some separation between different phases of development.

DevOps advances these ideas by viewing software development as a **continuous and integrated process** rather than a set of distinct stages. It removes boundaries between teams—whether they belong to development, operations, or testing—promoting a highly collaborative environment. In this paradigm, development is not limited to coding; instead, it involves creating a constant feedback loop where features are developed, deployed, monitored, and improved in real time with input from all stakeholders.

The primary advantage of DevOps lies in its seamless integration between development and operations, which enables continuous delivery, faster feature deployment, and instant feedback from users. Simultaneously, the testing process runs in parallel, allowing for quick identification and resolution of bugs.

Ultimately, DevOps fosters a **cohesive and cooperative workflow** where every team understands and values each other's roles. This synergy enhances productivity, reduces inefficiencies, and eliminates the blame culture that often arises when teams work in isolation.

2.2 How to Implement DevOps

1. **Integrated Development Environment (IDE):** A robust IDE supports version control systems like Git and build tools such as Maven. Developers write code in the IDE, and using Maven, tasks such as building and testing the application can be automated. Git integration allows multiple developers to work simultaneously without conflicts.

2. **Maven:** Maven serves as a build automation tool that manages compiling, packaging, and testing the project. It ensures consistent behavior across different environments, provided the required dependencies are available.

3. **Git (Version Control System):** Git helps developers manage and track changes to code, enabling them to revert updates if necessary and work on separate features without affecting the stable version. Git operates locally, unlike GitHub, which is primarily for remote collaboration.

4. **GitHub:** GitHub acts as a central repository where multiple developers can merge their changes. It simplifies collaboration and ensures that different project components are integrated properly. Alternatives like GitLab or self-hosted Git servers can be used for projects requiring stricter security.

5. **Jenkins:** Jenkins automates the build process whenever code changes are pushed to the repository. Integration with GitHub via webhooks allows Jenkins to trigger builds immediately, reducing the need for manual intervention.

6. **Jenkins Pipeline:** The CI/CD pipeline in Jenkins automates and monitors the entire workflow—from building and testing to deployment. Pipelines allow tasks to be chained based on success or failure of prior steps, streamlining operations.

7. **Docker:** Docker packages applications into lightweight containers that can be deployed consistently across multiple environments. Containers are more resource-efficient than virtual machines because they do not include a full operating system, making deployment faster and simpler.

8. **Docker Hub:** Docker Hub serves as a repository for container images, similar to how GitHub stores code. Developers can push container images to Docker Hub, making them accessible for testing or deployment from any location.

9. **Ansible:** Ansible ensures that deployment environments are consistent and reproducible. By providing Infrastructure as Code (IaC), Ansible eliminates the "it works on my machine" problem and guarantees that each deployment uses the same configuration.

3 Requirement Analysis

3.1 Functional Requirements of the Project

The Scientific Calculator program is designed to perform a set of mathematical operations based on user input through a menu-driven interface. The key functional requirements of the project are described below:

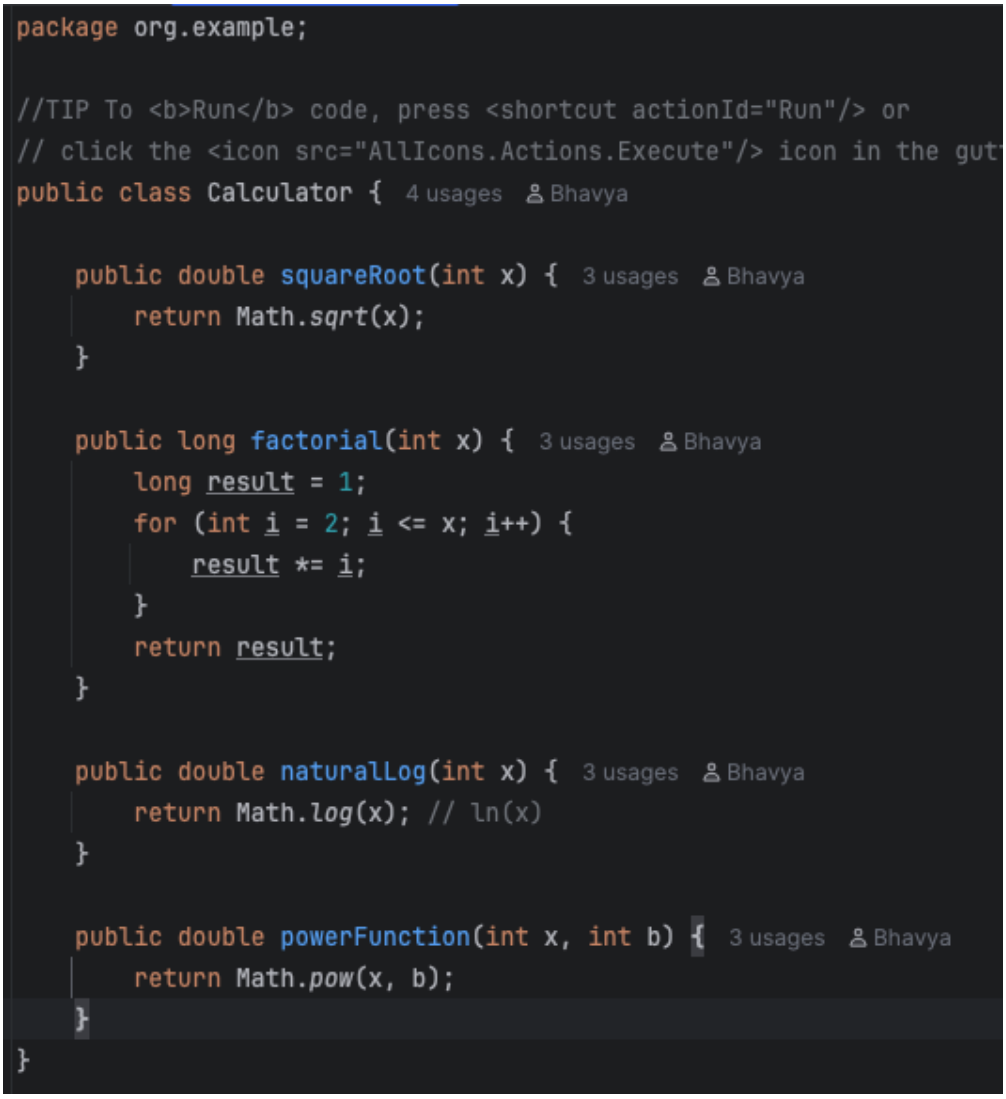
1. **Square Root Function ($\sqrt{x}$)** The calculator shall compute the square root of a given non-negative number x.
   - Input: A real number $x \geq 0$
   - Output: $\sqrt{x}$, the positive square root of x
2. **Factorial Function ($x!$)** The calculator shall determine the factorial of a given non-negative integer x.
   - Input: An integer $x \geq 0$
   - Output: $x! = x \times (x-1) \times (x-2) \times \ldots \times 1$
3. **Natural Logarithm Function ($\ln(x)$)** The calculator shall evaluate the natural logarithm (base e) of a positive number x.
   - Input: A real number $x > 0$
   - Output: $\ln(x)$, such that $e^{\ln(x)} = x$
4. **Power Function (x^b)** The calculator shall compute the result of raising a base number x to a given exponent b.
   - Input: Two real numbers x (base) and b (exponent)
   - Output: x^b

The program would display a user-friendly menu allowing the user to select any of the above operations, input the required values, and view the computed result.

3.2 Non-Functional Requirements of the Project

The non-functional requirements define the quality attributes, operational standards, and development practices that ensure the Scientific Calculator project is maintainable, reliable, and user-friendly.

- **User Experience Assistance:** The application should provide clear, informative guidance and error messages to help users operate the calculator correctly and handle invalid inputs gracefully.
- **Code Quality Assurance:** The code must be clean, well-structured, and thoroughly commented to enhance readability and maintainability for developers and future maintainers.



```java
package org.example;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gut
public class Calculator {  4 usages  Bhavya

    public double squareRoot(int x) {  3 usages  Bhavya
        return Math.sqrt(x);
    }

    public long factorial(int x) {  3 usages  Bhavya
        long result = 1;
        for (int i = 2; i <= x; i++) {
            result *= i;
        }
        return result;
    }

    public double naturalLog(int x) {  3 usages  Bhavya
        return Math.log(x); // ln(x)
    }

    public double powerFunction(int x, int b) {  3 usages  Bhavya
        return Math.pow(x, b);
    }
}
```

Figure 1: Scientific Calculator System Overview

- **Version Control:** A version control system such as **Git** and a repository platform like **GitHub** should be used to manage the project code, enabling proper collaboration and history tracking.
- **Build Automation:** Tools like **Maven** (or equivalents) should be used to automate compilation, testing, and building processes, ensuring consistent and reproducible builds across environments.
- **Continuous Integration/Continuous Deployment (CI/CD):** A CI/CD tool such as **Jenkins** should automate building, testing, and deployment pipelines, ensuring that code changes are continuously integrated and delivered.
- **Containerization:** The application should be packaged into containers using tools like **Docker** and stored in a remote repository (e.g., Docker Hub) for easy distribution

```java
package org.example;
import java.util.Scanner;

public class CalculatorInterface {  Bhavya

    public static void main(String[] args) {  Bhavya
        Calculator calculator = new Calculator();
        Scanner sc = new Scanner(System.in);

        boolean entry = true;

        while (entry) {
            System.out.println("=== Calculator Application Menu ===");
            System.out.println("1. Square Root");
            System.out.println("2. Factorial");
            System.out.println("3. Natural Logarithm");
            System.out.println("4. Power");
            System.out.println("5. Exit");
            System.out.print("Select an option: ");
            int choice = sc.nextInt();

            if (choice == 1) {
                System.out.print("Enter a non-negative number: ");
                int num = sc.nextInt();
                if (num < 0) {
                    System.out.println("Error: Square root is not defined for negative numbers.");
                } else {
                    System.out.println("√" + num + " = " + calculator.squareRoot(num));
                }
            } else if (choice == 2) {
                System.out.print("Enter a non-negative integer: ");
                int num = sc.nextInt();
                if (num < 0) {
                    System.out.println("Error: Factorial is not defined for negative numbers.");
                } else {
                    System.out.println(num + "! = " + calculator.factorial(num));
                }
```

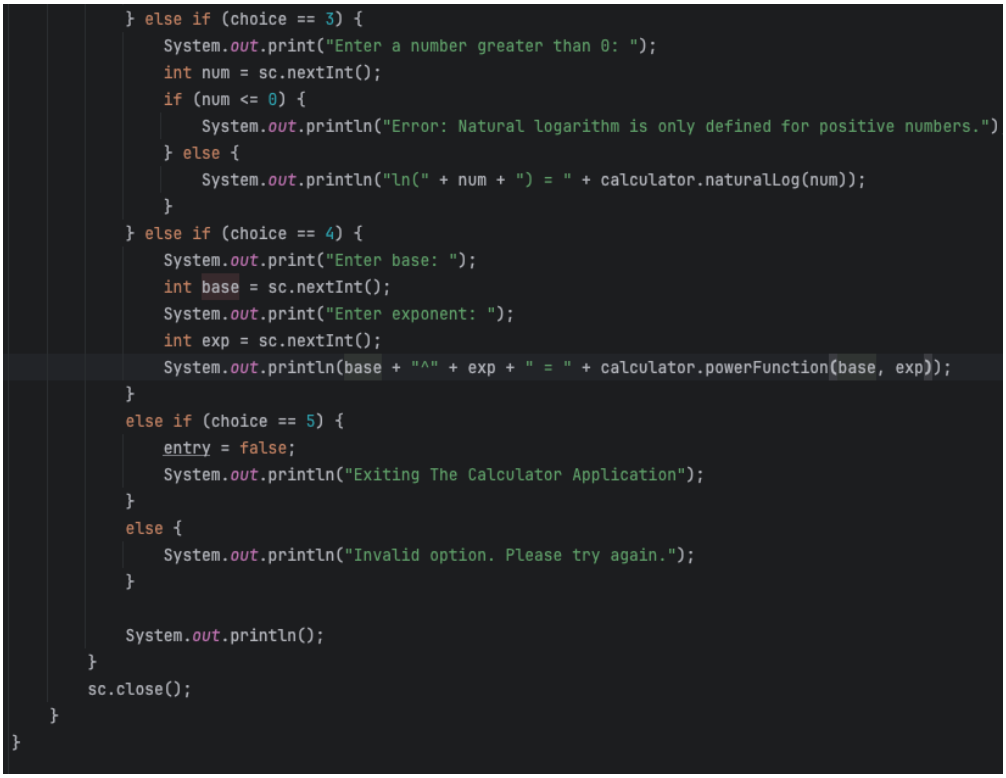
Figure 2: Scientific Calculator System

and deployment.

- **Environment Management:** Configuration management tools such as **Ansible** should be used to deploy containers consistently, ensuring repeatable and properly configured environments.

4 Tools Used

For the development and deployment of the Scientific Calculator project, a set of industry-standard tools were employed to ensure efficient version control, testing, building, integration, containerization, and deployment. The code is written in **Java**. **GitHub** was used for source code management and collaboration. Unit testing was performed using **JUnit** to validate code functionality. **Maven** handled build automation, ensuring consistent compilation and packaging of the project. **Jenkins** was utilized for continuous integration, enabling automated builds and tests. The application was containerized using **Docker**, and



```
            } else if (choice == 3) {
                System.out.print("Enter a number greater than 0: ");
                int num = sc.nextInt();
                if (num <= 0) {
                    System.out.println("Error: Natural logarithm is only defined for positive numbers.")
                } else {
                    System.out.println("ln(" + num + ") = " + calculator.naturalLog(num));
                }
            } else if (choice == 4) {
                System.out.print("Enter base: ");
                int base = sc.nextInt();
                System.out.print("Enter exponent: ");
                int exp = sc.nextInt();
                System.out.println(base + "^" + exp + " = " + calculator.powerFunction(base, exp));
            }
            else if (choice == 5) {
                entry = false;
                System.out.println("Exiting The Calculator Application");
            }
            else {
                System.out.println("Invalid option. Please try again.");
            }

            System.out.println();
        }
        sc.close();
    }
}
```

Figure 3: Scientific Calculator System

the resulting images were stored on **Docker Hub** for easy distribution. Finally, **Ansible** facilitated configuration management and deployment, allowing the Docker containers to be deployed reliably on local machines or remote servers. The steps below are for macOS:

4.1 Java

To install Java (OpenJDK 17) on macOS and verify the installation, use the following commands:

Listing 1: Java Installation Commands

```
# Step 1: Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    install/HEAD/install.sh)"

# Step 2: Install OpenJDK 17 using Homebrew
brew install openjdk@17

# Step 3: Add Java to PATH
sudo ln -sfn /usr/local/opt/openjdk@17/libexec/openjdk.jdk /Library/Java/
    JavaVirtualMachines/openjdk-17.jdk
echo 'export PATH="/usr/local/opt/openjdk@17/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

# Step 4: Verify Java installation
```

```
java --version
```

The expected output should be:

Listing 2: Expected Output

```
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)
```

4.2 Maven

To install Maven on macOS and verify the installation, use the following commands:

Listing 3: Maven Installation Commands

```
# Step 1: Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
    install/HEAD/install.sh)"

# Step 2: Install Maven using Homebrew
brew install maven

# Step 3: Verify Maven installation
mvn -version
```

The expected output should be similar to:

Listing 4: Expected Output

```
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: /opt/homebrew/Cellar/maven/3.9.9/libexec
Java version: 17.0.14, vendor: Eclipse Adoptium, runtime: /Library/Java/
    JavaVirtualMachines/temurin-17.jdk/Contents/Home
Default locale: en_IN, platform encoding: UTF-8
OS name: "mac os x", version: "15.6.1", arch: "aarch64", family: "mac"
```

4.3 Git and GitHub

Definition: Git is a distributed version control system used to track changes in source code, enabling collaboration among multiple developers. GitHub is a cloud-based hosting platform for Git repositories that provides tools for version control, collaboration, and CI/CD integration.

Listing 5: Git Commands

```
# Step 1: Install Git using Homebrew
brew install git

# Step 2: Verify Git installation
git --version
```

The expected output should be similar to:

Listing 6: Expected Output

```
git version 2.39.5 (Apple Git-154)
```

Step 3: Configure Git with your username and email.

Listing 7: Git Configuration

```
git config --global user.name "Bhavya Kapadia"
git config --global user.email "your_email@example.com"
```

Step 4: Create a new project directory and initialize it as a Git repository.

Listing 8: Repository Initialization

```
mkdir scientific-calculator
cd scientific-calculator
git init
```

Step 5: Add your project files and make the first commit.

Listing 9: Initial Commit

```
git add .
git commit -m "Initial commit"
```

Step 6: Create a new repository on GitHub manually or using GitHub CLI, then link your local repository to the remote GitHub repository.

Listing 10: Connect to GitHub Repository

```
git remote add origin https://github.com/your-username/scientific-
    calculator.git
git branch -M main
git push -u origin main
```

Step 7: Verify that the remote repository is connected properly.

Listing 11: Verify Remote Repository

```
git remote -v
```

4.4 Docker

Docker is a containerization platform that packages applications and their dependencies into lightweight, portable containers, allowing them to run consistently across different environments, with the Docker Daemon managing container execution and Docker Desktop providing a user-friendly interface. To install Docker on macOS and verify the installation, follow these commands:

Listing 12: Docker Commands

```
# Step 1: Install Docker Desktop using Homebrew
brew install --cask docker

# Step 2: Start Docker Desktop
open /Applications/Docker.app
```

```
# Step 3: Verify Docker installation
docker --version

# Note: Docker Desktop includes the Docker Daemon which runs in the
    background,
# allowing containers to be executed. Ensure the Docker Desktop app is
    running
# before executing Docker commands.
```

The expected output should be similar to:

Listing 13: Expected Output

```
Docker version 28.4.0, build d8eb465f86
```

4.5 Jenkins

Jenkins is an open-source automation server that facilitates continuous integration and continuous deployment (CI/CD) by automating the build, test, and deployment processes. It helps ensure faster development cycles, improved code quality, and seamless delivery through pipeline-based automation. Jenkins integrates easily with tools like Git, Maven, and Docker to create a fully automated DevOps workflow. Setup NGROK with Jenkins and Github. To install Jenkins on macOS, follow these steps:

Listing 14: Bash Commands

```
# Step 1: Download Jenkins WAR file
mkdir -p ~/jenkins
cd ~/jenkins
curl -LO https://get.jenkins.io/war-stable/latest/jenkins.war

# Step 2: Run Jenkins in user mode
java -jar jenkins.war

# Step 3: Alternative: Start Jenkins from Homebrew installation at port
    8080
java -jar /opt/homebrew/Cellar/jenkins-lts/2.516.3/libexec/jenkins.war --
    httpPort=8080

# Note: By default, Jenkins runs on port 8080.
# You can access it in your browser at http://localhost:8080
# The first time, Jenkins will ask for an initial admin password,
# which can be found in the terminal output or in the file:
# ~/jenkins/secrets/initialAdminPassword
```

The expected output should show something like:

Listing 15: Expected Output

```
Running from: /Users/bhavyakapadia/jenkins/jenkins.war
Jenkins home directory: /Users/bhavyakapadia/.jenkins
Jenkins is fully up and running
INFO: Jenkins started successfully and is listening on port 8080
```

4.6 Ansible

Ansible is an open-source automation tool used for configuration management, application deployment, and orchestration. It enables consistent and repeatable deployments by automating system setup and managing infrastructure through simple, human-readable YAML playbooks. Ansible helps streamline DevOps workflows by reducing manual intervention and ensuring environment consistency across servers. Installing Ansible on macOS:

Step 1: Update Homebrew to ensure you have the latest package list.

```
brew update
```

Step 2: Install Ansible using Homebrew.

```
brew install ansible
```

Step 3: Verify the installation.

```
ansible --version
```

Example Output:

```
ansible [core 2.19.2]
  config file = None
  configured module search path = ['/Users/bhavyakapadia/.ansible/plugins/
     modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /opt/homebrew/Cellar/ansible/12.0.0/
     libexec/lib/python3.13/site-packages/ansible
  ansible collection location = /Users/bhavyakapadia/.ansible/collections
     :/usr/share/ansible/collections
  executable location = /opt/homebrew/bin/ansible
  python version = 3.13.7 (main, Aug 14 2025, 11:12:11) [Clang 17.0.0 (
     clang-1700.0.13.3)] (/opt/homebrew/Cellar/ansible/12.0.0/libexec/bin/
     python)
  jinja version = 3.1.6
  pyyaml version = 6.0.2 (with libyaml v0.2.5)
```

5 Credentials Integration

Before initiating the project setup and deployment, it is crucial to have all the required credentials prepared in advance. Ensuring these credentials are available will facilitate a seamless configuration process and enable effective integration of the various tools used in the CI/CD pipeline through Jenkins.

5.1 GitHub Credentials

- **GitHub Username:** This is required to identify your account when accessing repositories.
- **Personal Access Token (PAT):** A secure token generated from GitHub that allows authentication for private repositories. It replaces the need for a password when performing operations via the Git command line or GitHub API.

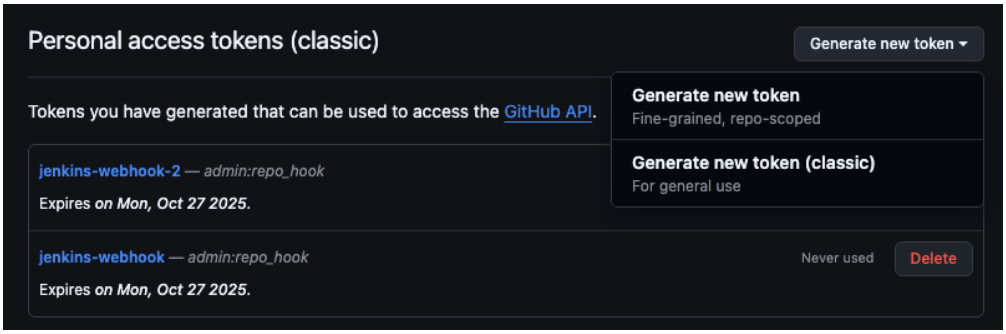


Figure 4: GitHub Credentials Overview

5.2 DockerHub Credentials

- **DockerHub Username:** Required to identify your account when pushing or pulling images from private repositories.
- **Password:** The password associated with your Docker Hub account is needed to authenticate these operations.

Managing Credentials in Jenkins

To securely manage credentials in Jenkins without hardcoding them in the Jenkinsfile, Jenkins can store and use the necessary credentials. By setting up credentials in Jenkins, sensitive information can be accessed using credential IDs, reducing the risk of exposing sensitive data in GitHub repositories or other environments.
Navigate to `Dashboard > Manage Jenkins > Credentials > System` and click on `Add Credentials`.

- **GitHub Credentials:**
  - **Credential ID:** 72a53cbb-0db4-4722-9eeb-b38d0ca90704
  - **Description:** GitHub Username & Personal Access Token
  - **Usage:** Utilized to access and clone private GitHub repositories within the Jenkins pipeline.
- **DockerHub Credentials:**
  - **Credential ID:** dockerhub-credentials
  - **Description:** DockerHub username & password
  - **Usage:** Utilized to push and pull Docker images from private Docker Hub repositories during the CI/CD pipeline execution.

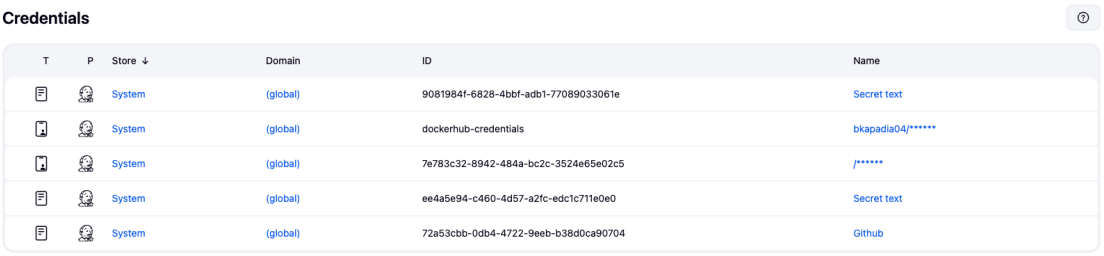


Figure 5: Jenkins Credentials Overview

6 Dockerfile for the Scientific Calculator Application

Definition: This Dockerfile defines a multi-stage build for the Scientific Calculator application. The first stage builds the application using Maven, and the second stage runs the application using OpenJDK 17.

Listing 16: Dockerfile Commands

```
# Stage 1: Build the application
FROM maven:3.8.5-openjdk-17 AS build

# Set the working directory inside the container
WORKDIR /app

# Copy the pom.xml and download the dependencies
COPY pom.xml .

# Copy the entire project source code
COPY src ./src

# Package the application
RUN mvn clean install -DskipTests

# Stage 2: Run the application
FROM openjdk:17-jdk-slim AS run

# Set the working directory inside the container
WORKDIR /app

# Copy the jar file from the build stage
COPY --from=build /app/target/*.jar ./app.jar

# Command to run the application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

6.1 Explanation of Dockerfile Commands

- **FROM maven:3.8.5-openjdk-17 AS build** Specifies the base image for the build stage. It includes Maven 3.8.5 and OpenJDK 17. The stage is named `build`.

- **WORKDIR /app** Sets the working directory inside the container to `/app`. All subsequent commands will run from this directory.
- **COPY pom.xml .** Copies the `pom.xml` file from the host machine to the container. This allows Maven to download project dependencies first.
- **COPY src ./src** Copies the entire source code from the host `src` folder to the container's `src` directory.
- **RUN mvn clean install -DskipTests** Runs Maven to compile the code and package it into a JAR file, skipping tests to speed up the build.
- **FROM openjdk:17-jdk-slim AS run** Specifies the base image for the run stage. Uses a lightweight OpenJDK 17 image. The stage is named `run`.
- **WORKDIR /app** Sets the working directory inside the runtime container to `/app`.
- **COPY –from=build /app/target/*.jar ./app.jar** Copies the compiled JAR file from the build stage into the runtime container and renames it `app.jar`.
- **ENTRYPOINT ["java", "-jar", "app.jar"]** Defines the command to run when the container starts. Executes the JAR file using Java.

7 Jenkins Pipeline for CI/CD of Scientific Calculator

Definition: This Jenkins pipeline automates the full CI/CD workflow for the Scientific Calculator project. It includes stages for code checkout, testing, Docker build and push, Python virtual environment setup, Ansible deployment, and email notifications.

Listing 17: Jenkins Pipeline Script

```
pipeline {
    agent any

    environment {
        IMAGE_NAME = "bkapadia04/calculator-spe-mini-project"
        IMAGE_TAG  = "latest"
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/BKAPADIA04/
                    calculator-spe-mini-project.git'
            }
        }

        stage('Run tests') {
            steps {
                sh 'mvn test'
            }
```

```
        }

        stage('Check Docker') {
            steps {
                echo 'Checking Docker version...'
                sh 'docker --version'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh "docker build -t ${IMAGE_NAME}:${IMAGE_TAG} ."
            }
        }

        stage('Push to Docker Hub') {
            steps {
                withCredentials([usernamePassword(credentialsId: '
                    dockerhub-credentials',
                                                  usernameVariable: '
                                                     DOCKER_USER',
                                                  passwordVariable: '
                                                     DOCKER_PASS')]) {
                    sh '''
                        echo "$DOCKER_PASS" | docker login -u "
                           $DOCKER_USER" --password-stdin
                        docker push ${IMAGE_NAME}:${IMAGE_TAG}
                    '''
                }
            }
        }

        stage('Setup Python Virtualenv') {
            steps {
                sh '''
                    python3 -m venv ~/ansible-venv
                    source ~/ansible-venv/bin/activate
                    pip install --upgrade pip
                    pip install ansible requests docker
                '''
            }
        }

        stage('Ansible Deployment') {
            steps {
                sh '''
                    source ~/ansible-venv/bin/activate
                    ansible-playbook -i inventory.ini deploy.yml
                    deactivate
                '''
            }
        }
    }
```

```
    post {
        success {
            mail to: 'bkapadia04@gmail.com',
                 subject: "[SUCCESS] Jenkins Pipeline Succeeded: ${
                    currentBuild.fullDisplayName}",
                 body: "<html>...Pipeline Succeeded HTML...</html>",
                 mimeType: 'text/html'
        }

        failure {
            mail to: 'bkapadia04@gmail.com',
                 subject: "[FAILURE] Jenkins Pipeline Failed: ${
                    currentBuild.fullDisplayName}",
                 body: "<html>...Pipeline Failed HTML...</html>",
                 mimeType: 'text/html'
        }
    }
}
```

Detailed Explanation of Jenkins Pipeline

This Jenkins pipeline automates the complete CI/CD workflow for the Scientific Calculator project. It defines environment variables for Docker image name and tag, performs source code checkout, testing, Docker build and push, sets up a Python virtual environment for Ansible deployment, and sends email notifications based on build status. The pipeline is designed to run on any available agent (`agent any`), making it flexible across different Jenkins nodes.

Environment Variables The `environment` block defines two important variables: `IMAGE_NAME` and `IMAGE_TAG`. These are used throughout the pipeline to standardize the Docker image naming convention. By defining these variables at the top, any stage can reference them, reducing hardcoding and improving maintainability.

Checkout Stage The `Checkout` stage fetches the latest source code from the GitHub repository. It uses the `git` step to clone the `main` branch of the repository. This ensures that the pipeline always works with the latest code committed to the central repository.

Run Tests Stage In the `Run tests` stage, the pipeline executes `mvn test` to run all the unit tests defined in the project. This step ensures that the code is functional and does not break existing features. By running tests early, errors are detected before building the Docker image, saving time and resources.

Check Docker Stage The `Check Docker` stage verifies the Docker installation on the Jenkins agent by running `docker --version`. This is a simple preflight check to ensure that the agent has Docker installed and available, preventing failures in subsequent Docker-related stages. This basically checks if the Docker daemon is running.

```java
package org.example;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class MainTest {  Bhavya

    private final Calculator calculator = new Calculator();  8 usages

    @Test  Bhavya
    void testSquareRoot() {
        assertEquals( expected: 5.0, calculator.squareRoot( x: 25));
        assertEquals( expected: 0.0, calculator.squareRoot( x: 0));
    }

    @Test  Bhavya
    void testFactorial() {
        assertEquals( expected: 120, calculator.factorial( x: 5));
        assertEquals( expected: 3628800, calculator.factorial( x: 10));
    }

    @Test  Bhavya
    void testNaturalLog() {
        assertEquals( expected: 2.302585092994046, calculator.naturalLog( x: 10));
        assertEquals( expected: 1.9459101490553132, calculator.naturalLog( x: 7));
    }

    @Test  Bhavya
    void testPowerFunction() {
        assertEquals( expected: 8.0, calculator.powerFunction( x: 2,  b: 3));
        assertEquals( expected: 0.25, calculator.powerFunction( x: 2,  b: -2));
    }


}
```

Figure 6: Tests

Build Docker Image Stage The `Build Docker Image` stage builds a Docker image of the application using the command `docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .`. It tags the image using the `IMAGE_NAME` and `IMAGE_TAG` environment variables. This ensures a consistent image name for deployment and distribution across environments.

Push to Docker Hub Stage In the `Push to Docker Hub` stage, the pipeline authenticates with Docker Hub using stored Jenkins credentials (`dockerhub-credentials`). It logs in securely using `echo "$DOCKER_PASS" | docker login -u "$DOCKER_USER" --password-stdin` and then pushes the Docker image using `docker push ${IMAGE_NAME}:${IMAGE_TAG}`. This allows the image to be available for deployment in any environment.

Setup Python Virtualenv Stage This stage sets up a Python virtual environment on the Jenkins agent (`~/ansible-venv`). The pipeline activates the environment, upgrades `pip`, and installs required Python packages including `ansible`, `requests`, and `docker`. This isolates dependencies for running Ansible playbooks without affecting the system Python.

Ansible Deployment Stage The `Ansible Deployment` stage runs the Ansible playbook on target hosts using the command `ansible-playbook -i inventory.ini deploy.yml`.

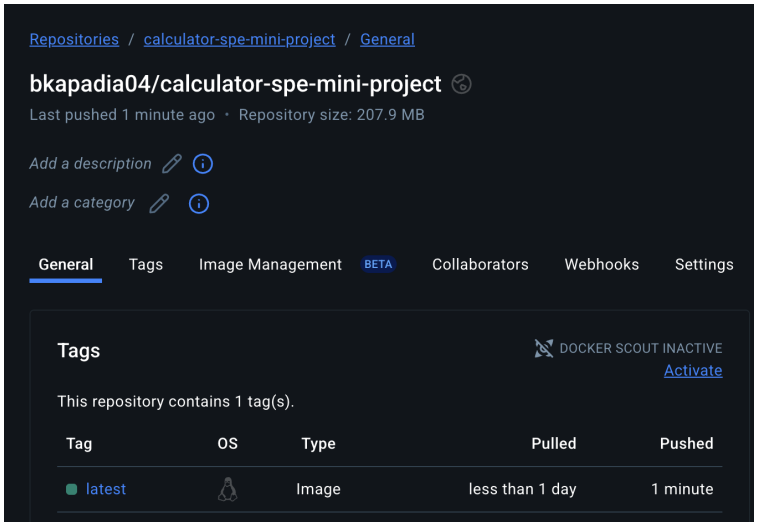


Figure 7: DockerHub

This deploys the Docker container as defined in the inventory and playbook. The Python virtual environment is activated before running the command and deactivated afterward.

Post Actions (Notifications) The `post` block handles notifications after the pipeline completes.

- **Success:** Sends an HTML-formatted email to notify that the pipeline succeeded, including job name, build number, and a link to the Jenkins build.

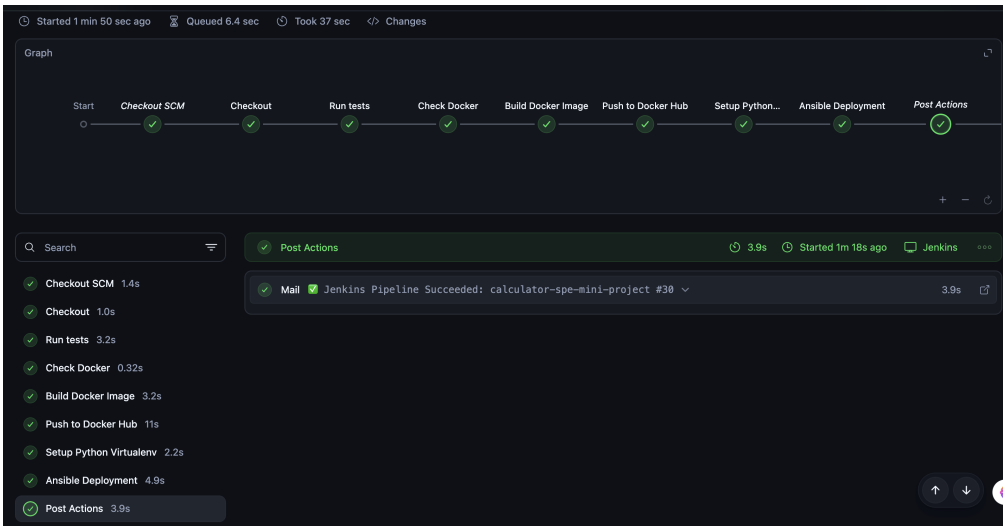


Figure 8: Jenkins Pipeline

- **Failure:** Sends a similar email if the pipeline fails, indicating the failure status and providing the build details link.

Figure 9: Email Success

Figure 10: Email Fail

This ensures stakeholders are informed immediately of build results, improving transparency and reducing manual monitoring.

Setting up Ansible for Deployment

To allow Ansible to connect to your servers securely without entering a password each time, you need to set up SSH keys and test the connection.

Step 1: Generate SSH Keys

Generate SSH keys on your local machine using the command:

Listing 18: Generate SSH Keys

```
ssh-keygen -t rsa
```

This will prompt you for a file name and passphrase. Press `Enter` to accept the defaults.

Step 2: Copy SSH Keys to Remote/Local Server

Copy your public key to the remote or local server using the command:

Listing 19: Copy SSH Key to Server

```
ssh-copy-id bhavyakapadia@172.16.139.46
```

Step 3: Testing the Setup

Verify that Ansible can connect to the specified hosts using the ping module:

Listing 20: Test Ansible Connection

```
ansible myhosts -i inventory.ini -m ping
```

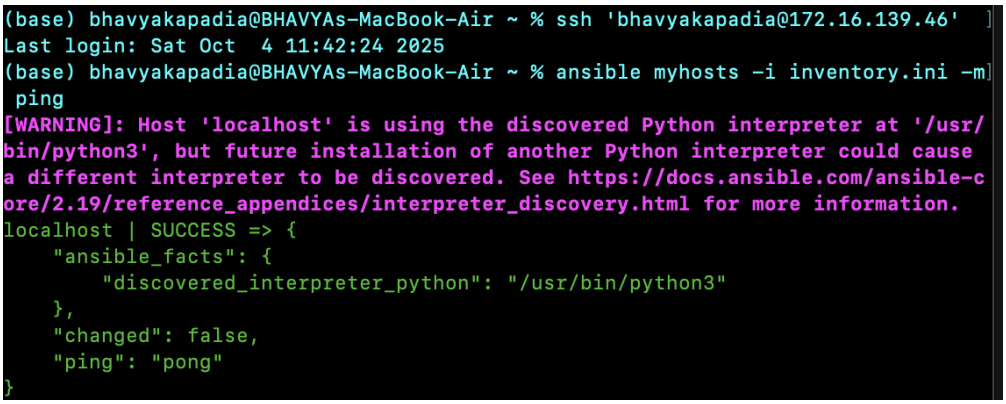


Figure 11: Ansible SSH Key

Ansible Inventory for Local Deployment

To configure Ansible for local deployment, we define a group called `local` in the inventory file. This allows Ansible to run tasks directly on the local machine without SSH.

Listing 21: inventory.ini

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

Explanation:

- `[local]`: Defines the group name for local deployment.
- `localhost`: Refers to the local machine.
- `ansible_connection=local`: Instructs Ansible to run tasks directly on the local machine instead of connecting via SSH.

Ansible Playbook: deploy.yml

This playbook automates pulling and running the Calculator CLI Docker image on the local machine. It ensures Docker is running, removes any existing container, and runs the latest image in detached mode.

Listing 22: Ansible Playbook deploy.yml

```
---
- name: Pull and run my calculator CLI Docker image
  hosts: localhost
  gather_facts: false
  vars:
    # Use Python from the virtual environment
    ansible_python_interpreter: /Users/bhavyakapadia/ansible-venv/bin/
       python3

  tasks:

    - name: Check if Docker is running
      ansible.builtin.command: docker info
      register: docker_status
      ignore_errors: yes

    - name: Fail if Docker is not running
      ansible.builtin.fail:
        msg: "Docker is not running. Please start Docker Desktop before
           running this playbook."
      when: docker_status.rc != 0

    - name: Pull my Docker image
      community.docker.docker_image:
        name: bkapadia04/calculator-spe-mini-project:latest
        source: pull

    - name: Remove existing container if present
      community.docker.docker_container:
        name: calculator-cli
        state: absent
        force_kill: true

    - name: Run my container in detached mode
      community.docker.docker_container:
        name: calculator-cli
        image: bkapadia04/calculator-spe-mini-project:latest
        state: started
        restart: yes
        interactive: true
        detach: yes
```

Explanation of Key Tasks

- **Check if Docker is running:** Runs `docker info` and registers the result in `docker_status`. This ensures the playbook only continues if Docker is available.
- **Fail if Docker is not running:** Uses `ansible.builtin.fail` to stop execution with a clear message if Docker is not running.

- **Pull Docker image:** The `community.docker.docker_image` module pulls the latest image `bkapadia04/calculator-spe-mini-project:latest` from Docker Hub.
- **Remove existing container:** Ensures that any container named `calculator-cli` is removed before starting a new one, avoiding conflicts.
- **Run container in detached mode:** The `community.docker.docker_container` module starts the container in detached mode (`detach: yes`), restarts it automatically if stopped (`restart: yes`), and allows interaction (`interactive: true`).

Running the Calculator CLI Container

After successfully completing the Jenkins pipeline and deploying via Ansible, the Docker image and container will be available on your system. To run the Calculator CLI inside the running container, use the following command:

Listing 23: Run Calculator CLI in Docker Container

```
docker exec -it calculator-cli java -jar app.jar
```

Explanation:

- `docker exec -it calculator-cli`: Executes a command inside the running container named `calculator-cli` interactively.
- `java -jar app.jar`: Runs the Calculator CLI Java application packaged as a JAR inside the container.

This command allows you to directly interact with the Calculator CLI, confirming that the Docker container is running correctly and that the application is functional.

```
(base) bhavyakapadia@BHAVYAs-MacBook-Air ~ % docker pull bkapadia04/calculator-spe-mini-project:latest
latest: Pulling from bkapadia04/calculator-spe-mini-project
Digest: sha256:b9eead030e7d7b0311baa5669fe6f8e7a0f8a9a818f339e274015a6cbb45e377
Status: Image is up to date for bkapadia04/calculator-spe-mini-project:latest
docker.io/bkapadia04/calculator-spe-mini-project:latest
(base) bhavyakapadia@BHAVYAs-MacBook-Air ~ % docker run -it bkapadia04/calculator-spe-mini-project:latest
=== Calculator Application Menu ===
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Select an option: 1
Enter a non-negative number: 4
√4 = 2.0

=== Calculator Application Menu ===
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Select an option: 2
Enter a non-negative integer: 4
4! = 24

=== Calculator Application Menu ===
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Select an option: 3
Enter a number greater than 0: 10
ln(10) = 2.302585092994046

=== Calculator Application Menu ===
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Select an option: 4
Enter base: 2
Enter exponent: 4
2^4 = 16.0

=== Calculator Application Menu ===
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
5. Exit
Select an option: 5
Exiting The Calculator Application
```

Figure 12: Scientific Calculator System Terminal