

SPE Mini Project Report

Ananthakrishna K
Roll Number: IMT2022086

[GitHub Repository](#)
[DockerHub Repository](#)

October 10, 2025

Contents

1	Introduction	3
1.1	What and Why of DevOps?	3
1.2	How to Implement DevOps	3
2	Requirement Analysis	3
2.1	Functional Requirements	3
2.2	Non-Functional Requirements	4
3	Tools Used & Setup	4
3.1	Java & Maven	4
3.2	Git & GitHub	4
3.3	Docker	5
3.4	Jenkins	5
3.5	Ansible	5
3.6	Ngrok	5
4	Project Workflow & Setup	6
4.1	Creating the Project and Implementing Code	6
4.2	Building and Testing Locally with Maven	6
4.3	Setting Up Version Control with Git and GitHub	9
4.3.1	Initializing the Local Repository	9
4.3.2	Creating a Remote GitHub Repository	9
4.3.3	Pushing Code to GitHub	9
4.4	Containerizing the Application with Docker	9
4.4.1	Writing the Dockerfile	9
4.4.2	Building and Pushing the Docker Image	10
4.5	Setting up the CI/CD Pipeline in Jenkins	10
4.5.1	Managing Credentials in Jenkins	10
4.5.2	Creating the Jenkins Pipeline Job	10
4.5.3	Writing the Jenkinsfile	12
4.6	Automating Deployment with Ansible	15
4.6.1	Creating the Ansible Inventory	15
4.6.2	Writing the Ansible Playbook	15
4.7	Enabling Automatic Builds with GitHub Webhooks & Ngrok	16
4.7.1	Exposing Jenkins with Ngrok	16
4.7.2	Configuring the GitHub Webhook	17
5	Conclusion	17

1 Introduction

1.1 What and Why of DevOps?

DevOps represents an evolution in software development and IT operations, building upon methodologies such as Agile and the Waterfall model. It is a cultural and professional movement that emphasizes collaboration and communication between software developers (Dev) and IT operations (Ops) professionals while automating the process of software delivery and infrastructure changes. The primary goal of DevOps is to shorten the systems development life cycle and provide continuous delivery with high software quality.

By breaking down the traditional silos between development and operations teams, DevOps treats the entire delivery lifecycle as a single, integrated process. This fosters a culture of shared responsibility, where teams are collectively accountable for the software from conception through to production and monitoring.

The adoption of DevOps practices yields several key benefits:

- **Increased Velocity:** Automation of the build, test, and deployment processes enables rapid and frequent releases, reducing the time-to-market for new features and updates.
- **Improved Reliability:** The emphasis on continuous integration and continuous testing ensures that code is validated at each stage, leading to higher quality releases and more stable operating environments.
- **Enhanced Collaboration:** By unifying development and operations, DevOps improves communication, eliminates bottlenecks, and fosters a more productive and efficient work environment.
- **Integrated Security:** The practice of DevSecOps, which integrates security into the CI/CD pipeline, ensures that security is a shared responsibility and is addressed throughout the entire application lifecycle.

1.2 How to Implement DevOps

The effective implementation of a DevOps methodology is facilitated by a well-integrated toolchain that automates and streamlines the software delivery pipeline. Each tool in the chain serves a specific purpose, working in concert to create a seamless workflow from source code to production deployment. Key components of this toolchain include:

- **Source Control Management (Git & GitHub):** Manages and tracks revisions to the codebase. Git provides distributed version control, while GitHub serves as a centralized, collaborative platform for hosting repositories.
- **Build Automation (Maven):** Automates the compilation of source code, management of project dependencies, and packaging of the application into a distributable format.
- **Continuous Integration/Continuous Deployment (Jenkins):** Acts as the central automation server. It orchestrates the CI/CD pipeline by integrating with other tools to automatically build, test, and deploy the application upon code changes.
- **Containerization (Docker):** Packages the application and its dependencies into a standardized, portable unit known as a container. This ensures consistent application behavior across all environments, from development to production.
- **Configuration Management (Ansible):** Automates the provisioning and configuration of the deployment environment. It ensures that infrastructure is consistent, repeatable, and managed as code.

2 Requirement Analysis

2.1 Functional Requirements

The project is to develop a command-line scientific calculator that performs the following mathematical operations, presented to the user via a menu:

- **Square Root:** Calculates the square root of a given number x , i.e., \sqrt{x} .
- **Factorial:** Computes the factorial of a non-negative integer x , i.e., $x! = x \times (x - 1) \times \dots \times 1$.
- **Natural Logarithm:** Finds the natural logarithm (base e) of a given positive number x , i.e., $\ln(x)$.
- **Power Function:** Raises a number x to the power of b , i.e., x^b .

The program must include error handling for invalid inputs, such as negative numbers for square root or logarithm, and non-integer values for the factorial function.

2.2 Non-Functional Requirements

- **Code Quality:** The code must be clean, well-commented, and easily maintainable.
- **Version Control:** The project's codebase must be managed using Git and hosted on a public GitHub repository.
- **Build Automation:** The build and testing process must be automated using Maven.
- **CI/CD:** A Jenkins pipeline must be implemented to automate the integration, testing, and deployment of the application.
- **Containerization:** The application must be packaged into a Docker container and the image pushed to a public Docker Hub repository.
- **Environment Management:** Ansible must be used to automate the deployment of the Docker container in a consistent environment.
- **Documentation:** A detailed report documenting the entire project workflow, tools, and setup must be provided.

3 Tools Used & Setup

All setup commands are intended for an Ubuntu-based Linux environment.

3.1 Java & Maven

Java is the programming language used for the application, and Maven is the build automation tool used to manage project dependencies and the build lifecycle.

Installation:

```
1 # Update package list
2 sudo apt update
3 # Install OpenJDK 21 and Maven
4 sudo apt install openjdk-21-jdk maven -y
```

Verification:

```
1 java -version
2 mvn -version
```

3.2 Git & GitHub

Git is a distributed version control system for tracking code changes locally. GitHub is a cloud-based platform for hosting Git repositories.

Installation:

```
1 sudo apt install git -y
```

Verification:

```
1 git --version
```

3.3 Docker

Docker is a platform for creating and running applications in containers, ensuring consistency across environments.

Installation:

```
1 # Add Docker's official GPG key and repository
2 sudo apt-get update
3 sudo apt-get install ca-certificates curl
4 sudo install -m 0755 -d /etc/apt/keyrings
5 sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker
6 .asc
7 sudo chmod a+r /etc/apt/keyrings/docker.asc
8 echo \
9     "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https
10    ://download.docker.com/linux/ubuntu \
11    $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
12    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
13 sudo apt-get update
14 # Install Docker packages
15 sudo apt-get install docker-ce docker-ce-cli containerd.io -y
16 # Add user to the docker group to run docker without sudo
17 sudo usermod -aG docker $USER
18 newgrp docker
```

Verification:

```
1 docker ps
```

3.4 Jenkins

Jenkins is an open-source automation server for implementing CI/CD pipelines.

Installation:

```
1 # Add Jenkins repository key and source
2 sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
3     https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
4 echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
5     https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
6     /etc/apt/sources.list.d/jenkins.list > /dev/null
7 sudo apt-get update
8 # Install Jenkins
9 sudo apt-get install jenkins -y
```

Setup: Start Jenkins with

```
1 sudo systemctl start jenkins
```

then navigate to <http://localhost:8080> to complete the initial setup.

3.5 Ansible

Ansible is an automation tool used for configuration management and application deployment.

Installation:

```
1 sudo apt update
2 sudo apt install software-properties-common -y
3 sudo add-apt-repository --yes --update ppa:ansible/ansible
4 sudo apt install ansible -y
```

Verification:

```
1 ansible --version
```

3.6 Ngrok

Ngrok is a tool that creates a secure tunnel to expose a local server to the public internet. It's essential for allowing GitHub webhooks to communicate with a Jenkins instance running on a local machine.

Installation:

```
ak@fedora:~$ java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (Red_Hat-21.0.8.0.9-1) (build 21.0.8+9)
OpenJDK 64-Bit Server VM (Red_Hat-21.0.8.0.9-1) (build 21.0.8+9, mixed mode, sharing)
ak@fedora:~$ mvn --version
Apache Maven 3.9.9 (Red Hat 3.9.9-14)
Maven home: /usr/share/maven
Java version: 21.0.8, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-21-openjdk
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "6.15.9-201.fc42.x86_64", arch: "amd64", family: "unix"
ak@fedora:~$ git --version
git version 2.51.0
ak@fedora:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ak@fedora:~$ ansible --version
ansible [core 2.18.6]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ak/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.13/site-packages/ansible
  ansible collection location = /home/ak/.ansible/collections:usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.13.7 (main, Aug 14 2025, 00:00:00) [GCC 15.2.1 20250808 (Red Hat 15.2.1-1)] (/usr/bin/python3)
  jinja version = 3.1.6
  libyaml = True
ak@fedora:~$
```

Figure 1: Verify all the installations.

```
1 # Add Ngrok's GPG key and repository
2 curl -s https://ngrok-agent.s3.amazonaws.com/ngrok.asc | \
3   sudo tee /etc/apt/trusted.gpg.d/ngrok.asc >/dev/null
4 echo "deb https://ngrok-agent.s3.amazonaws.com buster main" | \
5   sudo tee /etc/apt/sources.list.d/ngrok.list
6 sudo apt update
7 # Install Ngrok
8 sudo apt install ngrok -y
```

Setup: You need to add your authtoken from the Ngrok dashboard:

```
1 ngrok config add-authtoken <YOUR_TOKEN>
```

4 Project Workflow & Setup

This section details the step-by-step process of developing, containerizing, and deploying the application.

4.1 Creating the Project and Implementing Code

A standard Maven project was created. The core application logic was implemented in `Calculator.java`, providing a command-line interface for the scientific calculator functions, and unit tests were written in `CalculatorTest.java`.

4.2 Building and Testing Locally with Maven

Maven commands were used to compile, test, and package the application locally to ensure it works as expected before integrating it into the pipeline.

```
1 # Clean the project (removes previous build artifacts)
2 mvn clean
3 # Compile and run all unit tests
4 mvn test
5 # Package the application into an executable JAR file
6 mvn package
7 # Run the packaged application
8 java -jar target/calculator-1.0-SNAPSHOT.jar
```

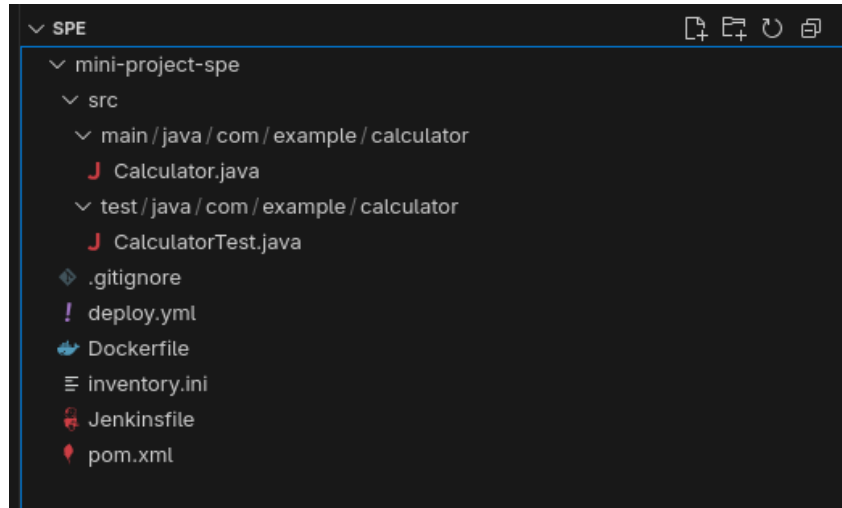


Figure 2: Folder Structure used in the project.

```

k@fedora:~/spe/mini-project-spe$ mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example.calculator:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ calculator ---
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.3.4/maven-shared-utils-3.3.4.jar
[INFO] Deleted from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.3.4/maven-shared-utils-3.3.4.jar (153 kB at 33 kB/s)
[INFO] Deleting /home/ak/spe/mini-project-spe/target
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 5.470 s
[INFO] Finished at: 2025-10-04T22:48:08+05:30
[INFO]

```

Figure 3: Expected result for build commands.

```

k@fedora:~/spe/mini-project-spe$ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example.calculator:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO]
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/ak/spe/mini-project-spe/src/main/resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ calculator ---
[INFO] Recompiling the module because of changed source code.
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 1.8] to target/classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[WARNING] source value 8 is obsolete and will be removed in a future release
[WARNING] target value 8 is obsolete and will be removed in a future release
[WARNING] To suppress warnings about obsolete options, use -Xlint:-options.
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ calculator ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/ak/spe/mini-project-spe/src/test/resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ calculator ---
[INFO] Recompiling the module because of changed dependency.
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file with javac [debug target 1.8] to target/test-classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[WARNING] source value 8 is obsolete and will be removed in a future release
[WARNING] target value 8 is obsolete and will be removed in a future release
[WARNING] To suppress warnings about obsolete options, use -Xlint:-options.
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junit4.JUnit4Provider
[INFO]
[INFO] T E S T S
[INFO]
[INFO] Running com.example.calculator.CalculatorTest
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.137 s -- in com.example.calculator.CalculatorTest
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.580 s
[INFO] Finished at: 2025-10-04T22:48:21+05:30
[INFO]

```

Figure 4: Expected result for build commands.

```

ak@fedora:~/spe/mini-project-spe$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example.calculator:calculator >-----
[INFO] Building calculator 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ calculator ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/ak/spe/mini-project-spe/src/main/resources
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date.
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ calculator ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/ak/spe/mini-project-spe/src/test/resources
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ calculator ---
[INFO] Nothing to compile - all classes are up to date.
[INFO]
[INFO] --- surefire:3.2.5:test (default-test) @ calculator ---
[INFO] Using auto detected provider org.apache.maven.surefire.junit4.JUnit4Provider
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.example.calculator.CalculatorTest
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.127 s -- in com.example.calculator.CalculatorTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] --- jar:3.4.1:jar (default-jar) @ calculator ---
[INFO] Building jar: /home/ak/spe/mini-project-spe/target/calculator-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- shade:3.2.4:shade (default) @ calculator ---
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /home/ak/spe/mini-project-spe/target/calculator-1.0-SNAPSHOT.jar with /home/ak/spe/mini-project-spe/target/calculator-1.0-SNAPSHOT-shaded.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.580 s
[INFO] Finished at: 2025-10-04T22:48:34+05:30
[INFO]
[INFO] -----
ak@fedora:~/spe/mini-project-spe$

```

Figure 5: Expected result for build commands.

```

ak@fedora:~/spe/mini-project-spe$ java -jar target/calculator-1.0-SNAPSHOT.jar
Press any key and Enter to start the calculator:

-----Calculator-----
--- Calculator Menu ---
1. Square Root (√x)
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power Function (x^b)
5. Exit
Enter your choice: 1
Enter a number: 34
Result: 5.830951894845301

--- Calculator Menu ---
1. Square Root (√x)
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power Function (x^b)
5. Exit
Enter your choice: 2
Enter an integer: 7
Result: 5040.0

--- Calculator Menu ---
1. Square Root (√x)
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power Function (x^b)
5. Exit
Enter your choice: 3
Enter a number: 5
Result: 1.6094379124341003

--- Calculator Menu ---
1. Square Root (√x)
2. Factorial (!x)
3. Natural Logarithm (ln(x))
4. Power Function (x^b)
5. Exit
Enter your choice: 5
Exiting
ak@fedora:~/spe/mini-project-spe$

```

Figure 6: The Calculator Application

4.3 Setting Up Version Control with Git and GitHub

4.3.1 Initializing the Local Repository

In the project root, a Git repository was initialized, and a '.gitignore' file was created to exclude build artifacts and IDE-specific files from version control.

```
1 # Initialize Git
2 git init
3 # Stage all files for the first commit
4 git add .
5 # Commit the files
6 git commit -m "Initial commit"
```

4.3.2 Creating a Remote GitHub Repository

A new public repository was created on GitHub to host the project's source code remotely.

4.3.3 Pushing Code to GitHub

The local repository was linked to the remote GitHub repository, and the initial commit was pushed.

```
1 # Add the remote repository URL
2 git remote add origin https://github.com/Ananthakrishna-K-13/mini-project-spe.git
3 # Push the main branch to the remote repository
4 git push -u origin main
```

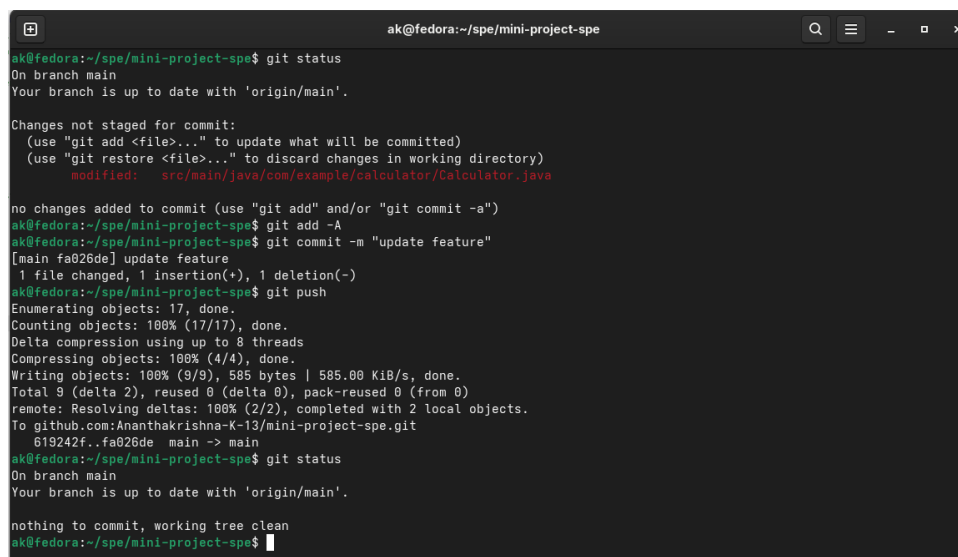
A terminal window titled 'ak@fedora:~/spe/mini-project-spe' showing the execution of Git commands. The user runs 'git status', which shows they are on the 'main' branch and up to date with 'origin/main'. They then run 'git add -A' to stage all changes, followed by 'git commit -m "update feature"', which creates a new commit '619242f..fa026de'. Finally, they run 'git push', which pushes the commit to the remote repository 'github.com:Ananthakrishna-K-13/mini-project-spe.git'. The output shows the progress of pushing, including enumerating objects, counting, compressing, and writing objects. After the push, 'git status' shows the working tree is clean and up to date with 'origin/main'.

Figure 7: Git commands to push updates.

4.4 Containerizing the Application with Docker

4.4.1 Writing the Dockerfile

A Dockerfile was created to package the application into a lightweight and portable container. This version uses a single-stage build process, starting with a minimal Java 8 runtime environment, copying the compiled JAR file, and setting the command to run the application on startup.

```
1 FROM openjdk:8-jre-alpine
2
3 WORKDIR /app
4
5 COPY target/calculator-1.0-SNAPSHOT.jar /app/application.jar
6
7 CMD ["java", "-jar", "/app/application.jar"]
```

Explanation of the Dockerfile:

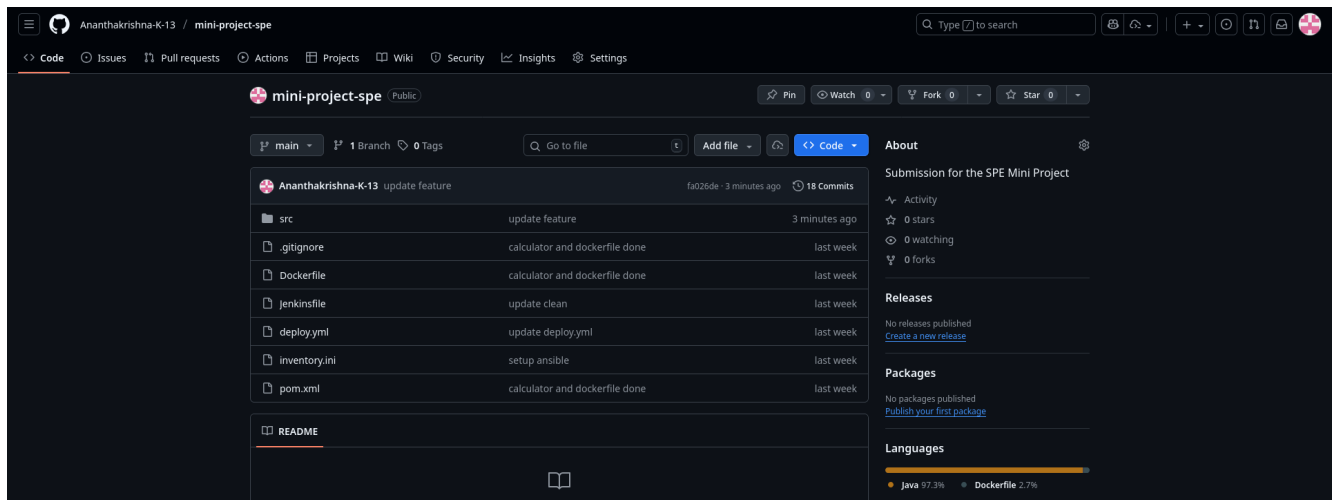


Figure 8: Github Repository page after push.

- **FROM openjdk:8-jre-alpine:** This line sets the base image for our container. It uses a minimal Alpine Linux image that includes the Java 8 Runtime Environment (JRE), which is all that's needed to run the compiled Java application. This keeps the final image size small.
- **WORKDIR /app:** This command sets the working directory inside the container to `/app`. Subsequent commands like `COPY` and `CMD` will be executed from this directory.
- **COPY target/calculator-1.0-SNAPSHOT.jar /app/application.jar:** This copies the compiled JAR file from the `target` directory of the host machine (where Maven places it) into the container's `/app` directory and renames it to `application.jar`.
- **CMD ["java", "-jar", "/app/application.jar"]:** This specifies the default command to run when the container starts. It executes the Java application packaged in the JAR file.

4.4.2 Building and Pushing the Docker Image

The Docker image was built from the `Dockerfile` and pushed to a public Docker Hub repository.

```
1 # Build the Docker image with a tag
2 docker build -t ananthak22/calculator:latest .
3 # Log in to Docker Hub
4 docker login -u ananthak22
5 # Push the image to Docker Hub
6 docker push ananthak22/calculator:latest
```

4.5 Setting up the CI/CD Pipeline in Jenkins

4.5.1 Managing Credentials in Jenkins

To allow Jenkins to interact securely with Docker Hub, credentials were added to the Jenkins credential manager.

- **Path:** Dashboard > Manage Jenkins > Credentials > System > Global credentials.
- **Credential Type:** Username with password.
- **ID:** `dockerhub-credentials` (this ID is referenced in the `Jenkinsfile`).

4.5.2 Creating the Jenkins Pipeline Job

A new Pipeline job was created in Jenkins and configured to use the 'Jenkinsfile' from the project's GitHub repository.

```
ak@fedora:~/sps/mini-project-sps$ docker build -t ananthak22/calculator:latest .
[*] Building 8.3s (9/9) FINISHED
--> [internal] load build definition from Dockerfile
--> [internal] load metadata for docker.io/library/openjdk:8-jre-alpine
--> [auth] library/openjdk:pull token for registry-1.docker.io
--> [internal] load .dockerignore
--> transferring context: 2B
--> [1/3] FROM docker.io/library/openjdk:8-jre-alpine@sha256:f3c2b105b878e129cbe738f2805f47395c8ea0bca5e44051c302936c3193
--> [internal] load build context
--> transferring context: 4.50B
--> CACHED [2/3] WORKDIR /app
--> [3/3] COPY target/calculator-1.0-SNAPSHOT.jar /app/application.jar
--> exporting to image
--> exporting layers
--> writing image sha256:f450b05c0e472157477c0b4a47386abe0aef930b1475281ba02c5900
--> pushing to docker.io/ananthak22/calculator:latest
ak@fedora:~/sps/mini-project-sps$ docker login -u ananthak22

Info - A Personal Access Token (PAT) can be used instead.
To create a PAT, visit https://docs.docker.com/settings

Password:

WARNING! Your credentials are stored unencrypted in '/home/ak/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
ak@fedora:~/sps/mini-project-sps$ docker push ananthak22/calculator:latest
The push refers to repository [docker.io/ananthak22/calculator]
62ad4f3284af: Pushing [=====] 6.650kB
7a24c0bae4cb: Layer already exists
edd0158bd126: Layer already exists
62ad4f3284af: Pushed
60b57f3050a0: Layer already exists
f1b59323fe4b5: Layer already exists
latest: digest: sha256:35d9852ea2f9899cda9ba67bae4ceb1ba3fd13cda92f28e47fe5c6f8e66da14 size: 1361
ak@fedora:~/sps/mini-project-sps$
```

Figure 9: Building and pushing the image to Dockerhub.

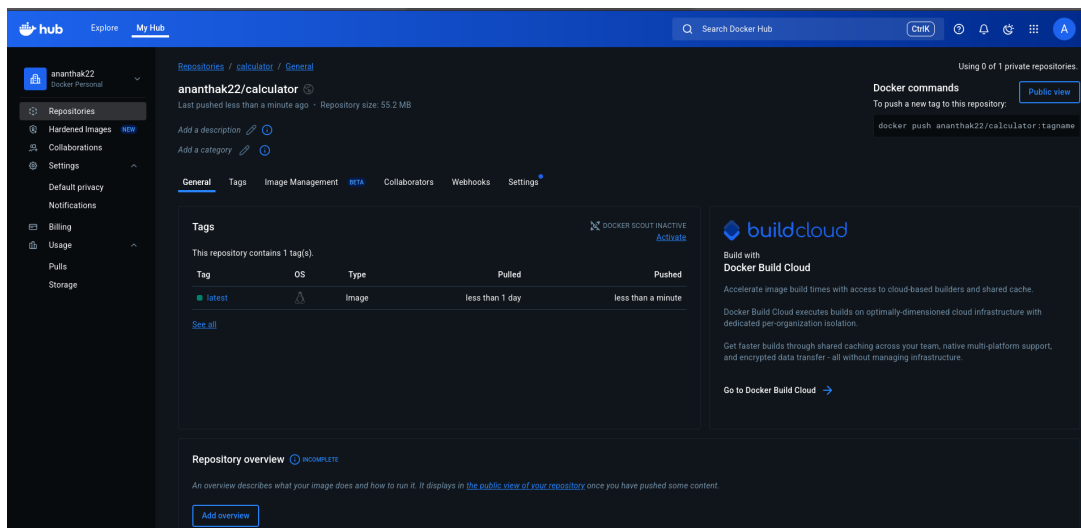


Figure 10: Dockerhub page of the Repository.

Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted)

New credentials

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username:

Blank username; did you mean to use secret text credentials instead?
☐ Treat username as secret

Password:

ID:

Create

Figure 11: Setting up Dockerhub credentials.

- **Type:** Pipeline.
- **Definition:** Pipeline script from SCM.
- **SCM:** Git.
- **Repository URL:** <https://github.com/Ananthakrishna-K-13/mini-project-spe>.
- **Script Path:** Jenkinsfile.

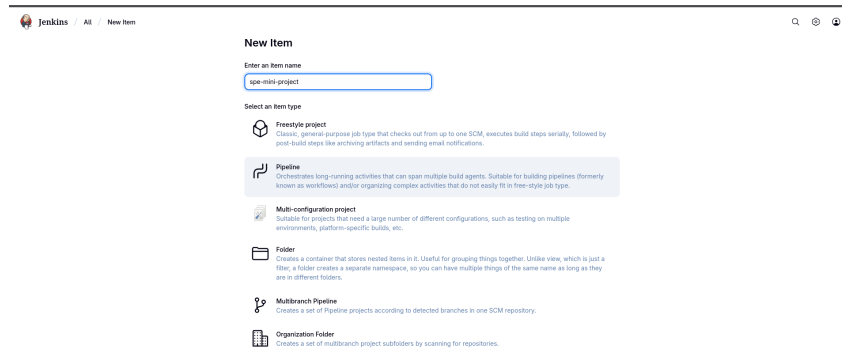


Figure 12: Setting up a Pipeline job in Jenkins.

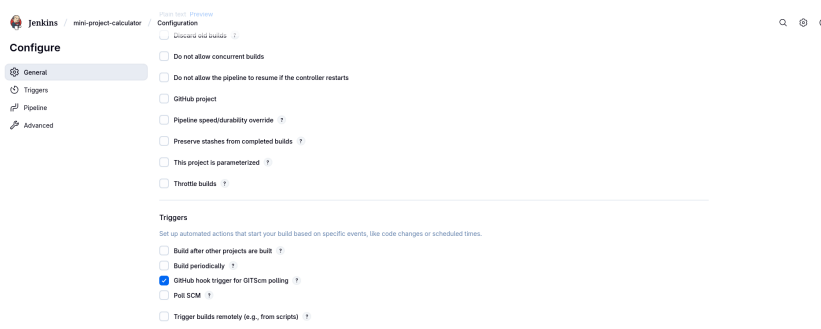


Figure 13: Setting up a Pipeline job in Jenkins.

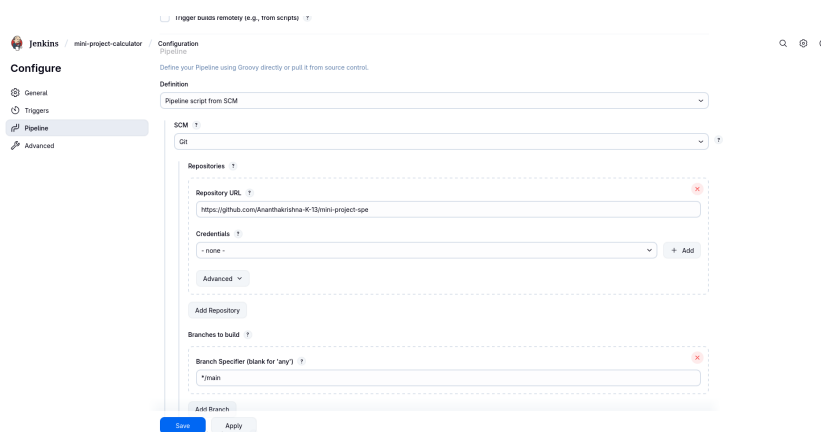


Figure 14: Setting up a Pipeline job in Jenkins.

4.5.3 Writing the Jenkinsfile

The **Jenkinsfile** defines the entire CI/CD process. This pipeline is structured with stages for building, testing, containerizing, and deploying the application. Key features include the use of an **environment**

block to manage variables, the `withCredentials` block for secure Docker Hub login, passing the image name dynamically to Ansible using `--extra-vars`, and a post block to clean up the workspace after each run.

```

1 pipeline {
2     agent any
3
4     environment {
5         DOCKERHUB_USERNAME = 'ananthak22'
6         IMAGE_NAME = "${env.DOCKERHUB_USERNAME}/calculator"
7         IMAGE_TAG = "latest"
8         CONTAINER_NAME = 'calculator_container'
9         EMAIL_RECIPIENTS = 'ananthakk26@gmail.com'
10    }
11
12    stages {
13        // Build and Test the Java Application
14        stage('Build & Test') {
15            steps {
16                script {
17                    echo ' Building and testing the application using Maven '
18                    sh 'mvn clean package'
19                }
20            }
21        }
22
23        // Build the Docker Image
24        stage('Build Docker Image') {
25            steps {
26                script {
27                    echo " Building Docker image: ${IMAGE_NAME}:${IMAGE_TAG} "
28                    sh "docker build -t ${IMAGE_NAME}:${IMAGE_TAG} ."
29                }
30            }
31        }
32
33        // Push Docker Image to Docker Hub
34        stage('Push to Docker Hub') {
35            steps {
36                script {
37                    echo " Pushing Docker image to Docker Hub "
38                    withCredentials([usernamePassword(credentialsId: 'dockerhub-
credentials', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_PASS')]) {
39                        // Log in to Docker Hub using the credentials
40                        sh "docker login -u ${DOCKER_USER} -p ${DOCKER_PASS}"
41                        // Push the image with its tag
42                        sh "docker push ${IMAGE_NAME}:${IMAGE_TAG}"
43                    }
44                }
45            }
46        }
47
48        //deploy using ansible
49        stage('Deploy with Ansible') {
50            steps {
51                script {
52                    echo "Deploying application using Ansible"
53                    sh "ansible-playbook -i inventory.ini deploy.yml --extra-vars '
image_name=${IMAGE_NAME}:${IMAGE_TAG}'"
54                }
55            }
56        }
57    }
58
59    post {
60        success {
61            script {
62                echo 'Pipeline was successful. Sending notification...'
63                emailxmt (
64                    subject: "SUCCESS: Pipeline '${env.JOB_NAME}' - Build #${env.
BUILD_NUMBER}",
65                    body: ""<p>The pipeline <b>${env.JOB_NAME}</b> build #${env.
BUILD_NUMBER} completed successfully.</p>

```

```

66         <p>Check the build output here: <a href="${env.BUILD_URL}">
        ${env.BUILD_URL}</a></p>""",
67         to: "${env.EMAIL_RECIPIENTS}",
68         mimeType: 'text/html'
69     )
70 }
71 }
72
73 failure {
74     script {
75         echo 'Pipeline failed. Sending notification...'
76         emailext (
77             subject: "FAILED: Pipeline '${env.JOB_NAME}' - Build #${env.
BUILD_NUMBER}",
78             body: ""<p>The pipeline <b>${env.JOB_NAME}</b> build #${env.
BUILD_NUMBER} failed.</p>
79                 <p>Check the build console output for errors: <a href="${
env.BUILD_URL}">${env.BUILD_URL}</a></p>""",
80             to: "${env.EMAIL_RECIPIENTS}",
81             mimeType: 'text/html'
82         )
83     }
84 }
85
86 always {
87     echo 'Cleaning up...'
88     sh "docker rm -f ${CONTAINER_NAME} || true"
89     sh "docker rmi -f ${IMAGE_NAME}:${IMAGE_TAG} || true" // Added -f to force
removal
90     cleanWs()
91 }
92 }
93 }

```

Explanation of the Jenkinsfile:

- **pipeline { ... }**: This is the main block that defines the entire CI/CD workflow.
- **agent any**: Specifies that the pipeline can run on any available Jenkins agent.
- **environment { ... }**: Defines global environment variables that are used throughout the pipeline, such as the Docker Hub username, image name, and container name. This makes the pipeline easy to configure and maintain.
- **stages { ... }**: This block contains the sequence of stages that make up the pipeline.
 - **Build & Test**: This stage uses Maven to compile the Java code, run unit tests, and package the application into a JAR file (`mvn clean package`).
 - **Build Docker Image**: It executes the `docker build` command to create a Docker image from the `Dockerfile`, tagging it with the name and tag defined in the environment variables.
 - **Push to Docker Hub**: This stage securely pushes the built image to Docker Hub. The `withCredentials` block retrieves the stored Docker Hub credentials and uses them to log in before pushing the image.
 - **Deploy with Ansible**: This final stage runs the Ansible playbook to deploy the application. It executes `ansible-playbook`, passing the dynamic image name to the playbook using the `--extra-vars` flag.
- **post { ... }**: This block defines actions that are run after the pipeline completes.
 - **success** and **failure** blocks send email notifications indicating the outcome of the build.
 - The **always** block executes regardless of the pipeline's status. It performs cleanup tasks, such as removing the container and the Docker image from the Jenkins agent and clearing the workspace (`cleanWs()`) to ensure a clean environment for the next build.



Figure 15: The complete pipeline in stage view.

4.6 Automating Deployment with Ansible

4.6.1 Creating the Ansible Inventory

The inventory file (`inventory.ini`) tells Ansible which hosts to manage. For this project, it's the local machine where Jenkins is running.

```
1 [local]
2 localhost ansible_connection=local
```

Explanation of the Ansible Inventory: This file defines the hosts that Ansible will manage.

- `[local]`: This is a group name for a set of hosts.
- `localhost ansible_connection=local`: This line specifies the host to be managed. `localhost` refers to the machine Ansible is running on. `ansible_connection=local` tells Ansible to execute the playbook directly on the local machine instead of trying to connect to it via SSH, which is ideal for a single-machine setup.

4.6.2 Writing the Ansible Playbook

The playbook (`deploy.yml`) automates the deployment. It uses privilege escalation (`become:yes`) to interact with the Docker daemon. The playbook is designed to be dynamic, accepting the `image_name` as a variable passed from the Jenkins pipeline. Its tasks are to pull the specified image, stop any existing container with the same name, and start a new container.

```
1 ---
2 - name: Deploy Docker Container
3   hosts: all
4   become: yes
5
6   vars:
7     image_name: "ananthak22/calculator:latest"
8     container_name: "calculator_container"
9
10  tasks:
11    - name: 1. Pull the latest Docker image from Docker Hub
12      community.docker.docker_image:
13        name: "{{ image_name }}"
14        source: pull
15
16    - name: 2. Stop and remove any existing container of the same name
17      community.docker.docker_container:
18        name: "{{ container_name }}"
19        state: absent
20
21    - name: 3. Start the new container from the pulled image
22      community.docker.docker_container:
23        name: "{{ container_name }}"
24        image: "{{ image_name }}"
25        state: started
```

```

26     interactive: yes
27     tty: yes
28     detach: no

```

Explanation of the Ansible Playbook: This playbook defines the automated steps for deploying the Docker container.

- **hosts:** `all`: This specifies that the playbook should run on all hosts defined in the inventory file.
- **become:** `yes`: This instructs Ansible to use privilege escalation (e.g., ‘sudo’) to execute the tasks, which is necessary for interacting with the Docker daemon.
- **vars:** This section defines default variables. The `image_name` variable is overridden by the value passed from the Jenkins pipeline, making the playbook reusable.
- **tasks:** This section lists the actions to be performed.
 - **Task 1: Pull the latest Docker image:** It uses the `docker_image` module to pull the specified image from Docker Hub. This ensures the latest version is always used for deployment.
 - **Task 2: Stop and remove any existing container:** It uses the `docker_container` module with `state: absent` to ensure that any container with the same name is stopped and removed before a new one is launched. This prevents conflicts.
 - **Task 3: Start the new container:** It uses the `docker_container` module again to start a new container from the pulled image. The options `interactive: yes` and `tty: yes` allow the command-line application to run correctly.

```

< TASK [1. Pull the latest Docker image from Docker Hub] >
-----
      \   ^__^
      \  (oo)\_______
         (__)\       )\/\
            ||----w |
            ||     ||

ok: [localhost]

/ TASK [2. Stop and remove any existing container of the same \
\ name] /
-----
      \   ^__^
      \  (oo)\_______
         (__)\       )\/\
            ||----w |
            ||     ||

ok: [localhost]

< TASK [3. Start the new container from the pulled image] >
-----
      \   ^__^
      \  (oo)\_______
         (__)\       )\/\
            ||----w |
            ||     ||

changed: [localhost]

< PLAY RECAP >
-----
      \   ^__^
      \  (oo)\_______
         (__)\       )\/\
            ||----w |
            ||     ||

localhost                : ok=4   changed=1   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0

```

Figure 16: Ansible tasks executed by Jenkins on localhost.

4.7 Enabling Automatic Builds with GitHub Webhooks & Ngrok

4.7.1 Exposing Jenkins with Ngrok

To make the local Jenkins server reachable from the public internet, Ngrok was used to create a secure tunnel.

```

1 # Start ngrok to forward traffic to Jenkins on port 8080
2 ngrok http 8080

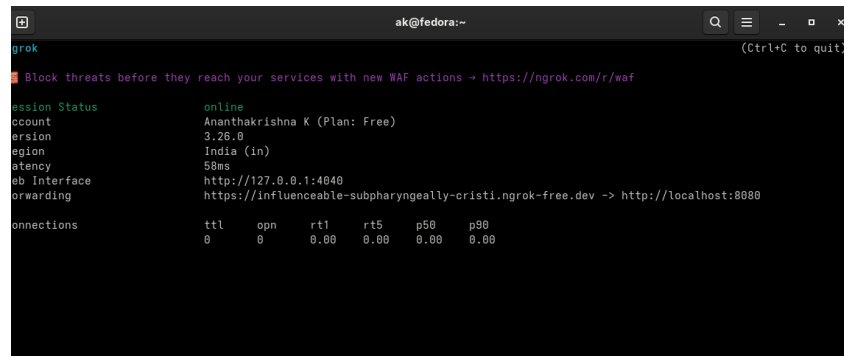
```

This command provides a public URL.

4.7.2 Configuring the GitHub Webhook

The public URL from Ngrok was used to create a webhook in the GitHub repository, which triggers the Jenkins pipeline on every push to the 'main' branch.

- **Path:** GitHub Repo > Settings > Webhooks > Add webhook.
- **Payload URL:** The Ngrok forwarding URL, followed by /github-webhook/.
- **Content type:** application/json.
- **Trigger:** Just the push event.



```
ak@fedora:~  
ngrok  
Block threats before they reach your services with new WAF actions → https://ngrok.com/r/waf  
Session Status      online  
Account             Ananthakrishna K (Plan: Free)  
Version             3.26.0  
Region              India (in)  
Latency             58ms  
Web Interface       http://127.0.0.1:4040  
Forwarding           https://influenceable-subpharyngeally-cristi.ngrok-free.dev → http://localhost:8080  
  
Connections  
t1l    opn    rt1    rt5    p50    p90  
0       0       0.00   0.00   0.00   0.00
```

Figure 17: The exposed Jenkins URL using ngrok.

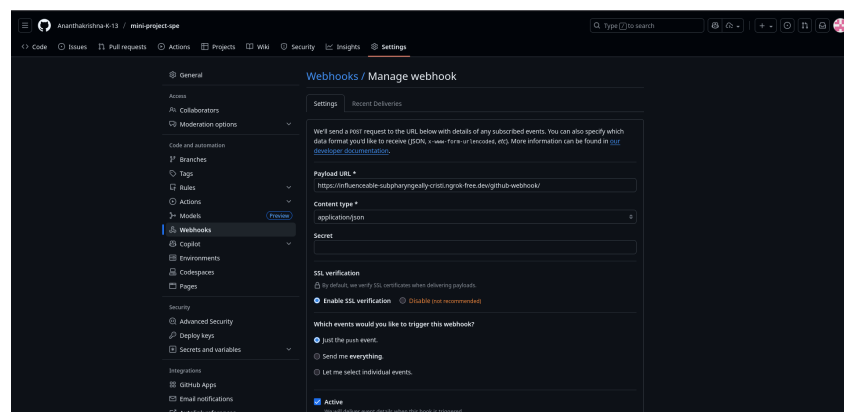


Figure 18: Setting up ngrok webhook in Github .

5 Conclusion

This project successfully demonstrates a complete, end-to-end DevOps lifecycle for a simple Java application. By integrating key tools like Git, Maven, Jenkins, Docker, and Ansible, we have created a fully automated CI/CD pipeline.

Key achievements include:

- **Automation:** The entire process from code commit to deployment is automated, reducing manual effort and potential for human error.
- **Consistency:** Docker and Ansible ensure that the application and its environment are consistent, reproducible, and portable.
- **Collaboration:** Git and GitHub provided a robust platform for version control and collaborative development.

- **Efficiency:** The CI/CD pipeline allows for rapid and reliable delivery of new features and fixes.

This project serves as a practical example of how DevOps principles and tools can be applied to streamline software development and delivery, ultimately leading to higher quality software and faster time-to-market.