SPE Mini Project

Ram Sai Koushik Polisetti

May 2024

Contents

1 Introduction

This report provides a comprehensive overview of the steps involved in building a DevOps pipeline for a command-line calculator program. The focus of the report is on demonstrating the use of various DevOps tools, such as Maven, Git, Docker, Jenkins, and Ansible, to automate Continuous Integration (CI) and Continuous Deployment (CD) processes.

The report begins with an explanation of each tool used in the pipeline, detailing its purpose and key functionalities. Following this, the setup process is described in detail, including the specific commands and configurations employed to build, test, and deploy the application. Screenshots and relevant examples are included to provide further clarity on the process.

Additionally, the GitHub repository and DockerHub images used for the project are provided:

- GitHub Repository: `https://github.com/RamsaiKoushik/Calculator-SPE-Miniproject.git`
- DockerHub: `https://hub.docker.com/r/ramsai987/calculator`

By the end of this report, the entire lifecycle of the project—from code commit to deployment—will be thoroughly illustrated using the chosen DevOps tools and practices.

2 What and Why of DevOps?

2.1 What is DevOps?

DevOps is a combination of "Development" and "Operations" practices aimed at automating and improving the process of software delivery. It bridges the gap between software development (Dev) and IT operations (Ops), enabling faster development cycles, more reliable software releases, and better collaboration between teams.

2.2 Why DevOps?

- **Faster Delivery**: By automating the software development process, DevOps enables quicker deployments and shorter release cycles, allowing organizations to deliver software features, bug fixes, and security updates faster.
- **Reliability and Consistency**: DevOps practices, such as Continuous Integration (CI) and Continuous Deployment (CD), ensure that code is automatically tested and deployed, reducing the chances of bugs making it to production. Automated monitoring helps in quickly identifying and resolving issues, ensuring a more stable environment.
- **Scalability**: With DevOps, infrastructure is often managed using code (Infrastructure as Code, or IaC), allowing organizations to easily scale up or down based on demand. This flexibility is crucial for cloud-based applications and large-scale systems.
- **Improved Collaboration**: DevOps promotes a culture of collaboration between developers, operations, and other stakeholders. This leads to shared responsibilities and better communication across teams, reducing friction in the software delivery process.

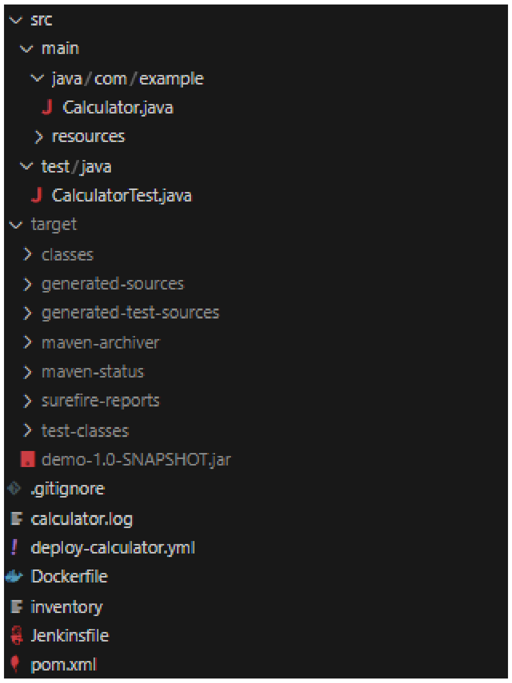
3 Codebase

The calculator program is implemented as a Maven project. **Maven** is used as the build automation tool for compiling, testing, and packaging the code. The project structure follows the standard Maven directory layout, which helps in organizing the code, tests, and dependencies efficiently. We will discuss more about Maven and its role in the pipeline in the *Tools Used* section.

3.1 Project Structure

The following is the structure of the Maven project:

- **src/main/java**: Contains the main application code. The core functionality of the calculator (square root, factorial, natural logarithm, power function) is implemented here.
- **src/test/java**: Holds the unit tests for the scientific calculator program. Tests are written to validate each function in the calculator.
- **pom.xml**: The Maven Project Object Model (POM) file, which lists the dependencies, build configurations, and plugins used in the project.
- **target**: The directory where Maven outputs the compiled classes and packaged application (JAR file).



```
src
  main
    java / com / example
      Calculator.java
    resources
  test / java
    CalculatorTest.java
target
  classes
  generated-sources
  generated-test-sources
  maven-archiver
  maven-status
  surefire-reports
  test-classes
  demo-1.0-SNAPSHOT.jar
.gitignore
calculator.log
deploy-calculator.yml
Dockerfile
inventory
Jenkinsfile
pom.xml
```

Figure 1: Maven Project Structure

3.2 Execution

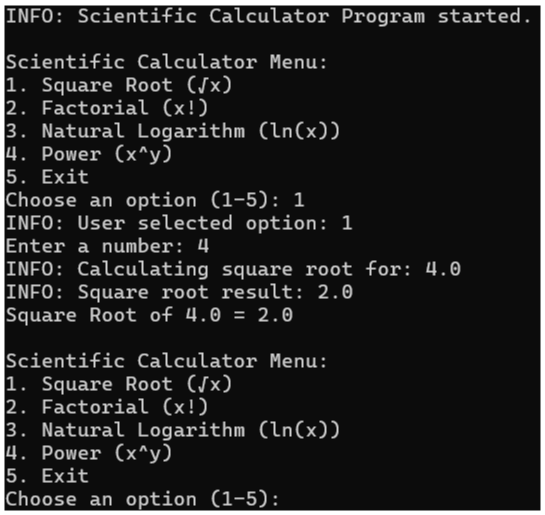
Upon executing the application, users are prompted with a menu-driven interface to select the desired operation. The program does the calculation based on the user's input and displays the result in the console.

4 Tools Used

4.1 Maven

4.1.1 What is Maven?

Maven is a powerful build automation tool primarily used for Java projects. It simplifies the build process, dependency management, and project configuration. Maven uses a Project Object Model (POM) file to manage the project's structure, dependencies, and build lifecycle, allowing developers to automate repetitive tasks and focus on writing code.



```
INFO: Scientific Calculator Program started.

Scientific Calculator Menu:
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^y)
5. Exit
Choose an option (1-5): 1
INFO: User selected option: 1
Enter a number: 4
INFO: Calculating square root for: 4.0
INFO: Square root result: 2.0
Square Root of 4.0 = 2.0

Scientific Calculator Menu:
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^y)
5. Exit
Choose an option (1-5):
```

Figure 2: Execution Output of the Scientific Calculator

4.1.2 Why Use Maven?

Maven standardizes the project's build process, enabling easy collaboration among developers by using the same project structure and build configurations. It manages dependencies efficiently, automatically downloading and resolving required libraries. Moreover, it integrates seamlessly with other DevOps tools for testing, packaging, and deployment, making it an essential tool for automating Java project pipelines.

4.1.3 Key Terminologies

- **POM (Project Object Model):** An XML file ('pom.xml') that contains information about the project and its dependencies, build plugins, and configurations.
- **Dependencies:** External libraries or frameworks required by the project, which Maven downloads and manages automatically.
- **Artifact:** A file, typically a JAR or WAR, produced by Maven as a result of the build process.
- **Repository:** A storage location for artifacts. Maven Central is the default remote repository used to download dependencies.

4.1.4 Common Commands and Their Usage

- **Building and Installing the Project:**
  To clean, compile, test, and install the project into the local repository, use:

  ```
  mvn clean install
  ```

  The 'clean' phase deletes the 'target/' directory from previous builds. The 'install' phase compiles, runs tests, packages the project, and installs the generated artifact into the local Maven repository for use by other projects.

- **Running Tests:**
  To run the unit tests defined in the project, use the command:

  ```
  mvn test
  ```

This compiles the test classes and runs them, ensuring that the code is functioning as expected.

- **Resolving Dependencies:**
  To update the dependencies specified in the ‘pom.xml‘, use:

  ```
  mvn dependency:resolve
  ```

  This command resolves and downloads the necessary dependencies required by the project, ensuring they are available during the build process.

4.2 Git and GitHub

4.2.1 What is Git?

Git is a distributed version control system that allows developers to track changes to files, coordinate work on those files among multiple team members, and manage project versions. It enables developers to maintain a detailed history of their codebase, branch out features, and merge them back into the main project once stable.

4.2.2 What is GitHub?

GitHub is a web-based platform that uses Git for version control and is commonly used for hosting repositories. It allows teams to collaborate on code, perform code reviews, track issues, and integrate with continuous integration (CI) tools like Jenkins. GitHub also supports pull requests, enabling developers to review, discuss, and approve code before merging it into the main branch.

4.2.3 Why Use Git and GitHub?

- **Git:** Git provides efficient collaboration among developers, allowing multiple people to work on the same project simultaneously without overwriting each other’s changes. It ensures the integrity of project history and allows developers to revert to previous versions when needed.
- **GitHub:** GitHub enhances Git by providing a user-friendly interface to manage repositories, review code, and collaborate effectively. It integrates well with DevOps pipelines, making it ideal for continuous integration and deployment (CI/CD).

4.2.4 Key Terminologies

- **Repository (Repo):** A project’s folder that Git is tracking. It stores all files, branches, and the version history.
- **Commit:** A snapshot of changes in the repository. Each commit has a unique ID and contains the changes made along with a message describing the changes.
- **Pull Request (PR):** A request to merge changes from one branch to another. Typically used in GitHub for code review before merging.
- **Clone:** A copy of a remote repository, downloaded to a local machine for development.
- **Fork:** A personal copy of another user’s repository, allowing you to freely experiment with changes without affecting the original repository.
- **Issue Tracking:** A feature in GitHub to manage bugs or tasks within the project, facilitating collaboration and planning.

4.2.5 Common Commands and Their Usage

- **Cloning a Repository:**
  To clone a remote repository from GitHub to your local machine, use:

  ```
  git clone https://github.com/user/repo.git
  ```

  This command creates a copy of the repository on your local machine.

- **Checking Repository Status:**
  To check the status of your repository (files changed, added, or deleted), use:

  ```
  git status
  ```

- **Staging and Committing Changes:**
  First, stage the changes you want to commit:

  ```
  git add <file_name>  # Add a specific file
  git add .            # Add all changes in the current directory
  ```

  After staging the files, commit them with a descriptive message:

  ```
  git commit -m "Added new feature for scientific calculator"
  ```

- **Pushing Changes to Remote Repository:**
  To push your commits to the remote GitHub repository, use:

  ```
  git push origin <branch_name>
  ```

  This command pushes your local branch changes to the corresponding branch on GitHub.

- **Setting up Remote Repository:**
  If your local repository is not yet connected to GitHub, you can set it up by adding a remote:

  ```
  git remote add origin https://github.com/user/repo.git
  ```

  This command links your local repository to the remote GitHub repository. You can then push and pull changes.

- **Pulling Latest Changes from Remote:**
  To update your local repository with the latest changes from the remote repository:

  ```
  git pull origin <branch_name>
  ```

4.3 Docker

4.3.1 What is Docker?

Docker is a platform that enables developers to package applications and their dependencies into lightweight, portable containers. These containers can run consistently across different environments, ensuring that the application behaves the same on a developer's machine, in testing, and in production.

4.3.2 Why Use Docker?

Docker ensures that applications are portable and can run on any system that supports Docker, eliminating the "works on my machine" issue. Containers are isolated from each other and the host system, which improves security and allows for efficient use of system resources. Additionally, Docker simplifies application deployment by bundling everything needed to run the application within the container.

4.3.3 Key Terminologies

- **Container:** A lightweight, standalone, and executable package that includes everything needed to run an application (code, runtime, libraries, environment variables, and system tools).
- **Docker Image:** A read-only template used to create Docker containers. It contains the application code, runtime, libraries, and other dependencies.
- **Dockerfile:** A text file that contains instructions to build a Docker image. It defines the environment in which your application runs.
- **Docker Hub:** A cloud-based registry where Docker images are stored and shared publicly or privately.
- **Volume:** A persistent storage mechanism in Docker, allowing data generated by and used within containers to be saved outside the container lifecycle.
- **Registry:** A place where Docker images are stored, such as Docker Hub or a private registry.
- **Docker Engine:** The core component of Docker that allows you to build, run, and manage containers. It includes the server (the Docker daemon) and the client (the Docker CLI).
- **Docker Client:** The command-line interface (CLI) that allows users to interact with the Docker Engine. It sends commands to the Docker daemon to manage containers, images, networks, and volumes.

4.3.4 Common Commands and Their Usage

- **Building a Docker Image:**
  To build a Docker image from a Dockerfile, use the following command:

  ```
  docker build -t {image_name} .
  ```

  This command builds the Docker image in the current directory (.) and tags it with the name image_name.

- **Running a Docker Container:**
  To create and run a container from an image, use:

  ```
  docker run -d -p <host_port>:<container_port> <image_name>
  ```

  The -d flag runs the container in detached mode, and the -p flag maps the host machine's port to the container's port, allowing access from outside.

- **Listing Running Containers:**
  To view all running Docker containers, use:

  ```
  docker ps
  ```

- **Stopping a Container:**
  To stop a running container, use:

  ```
  docker stop {container_id}
  ```

  The container_id can be obtained using the docker ps command.

- **Pushing an Image to Docker Hub:**
  After building your Docker image, push it to Docker Hub using the following command:

  ```
  docker push <dockerhub_username>/<image_name>
  ```

  Ensure you are logged in to Docker Hub before pushing an image:

  ```
  docker login
  ```

- **Pulling an Image from Docker Hub:**
  To pull a Docker image from Docker Hub to your local machine, use:

  ```
  docker pull <dockerhub_username>/<image_name>
  ```

- **Removing a Docker Container:**
  To remove a stopped container, use:

  ```
  docker rm <container_id>
  ```

- **Removing a Docker Image:**
  To delete an image from your local machine, use:

  ```
  docker rmi <image_id>
  ```

4.4 Jenkins

4.4.1 What is Jenkins?

Jenkins is an open-source automation server that helps automate parts of the software development process related to building, testing, and deploying applications. It provides continuous integration (CI) and continuous delivery (CD) capabilities, allowing developers to frequently integrate code changes into a shared repository.

4.4.2 Why Use Jenkins?

Jenkins simplifies the automation of the build process, enabling teams to detect issues early and often. It supports a wide range of plugins that integrate with various version control systems, build tools, and deployment strategies, making it highly extensible and adaptable to different project requirements.

4.4.3 Key Terminologies

- **Job:** A task or a series of tasks defined in Jenkins that can build or test a project. Each job can be configured to run on specific events, such as code commits.
- **Pipeline:** A series of automated steps defined in Jenkins to perform the build, test, and deploy processes. Pipelines can be defined using a domain-specific language (DSL) called Jenkinsfile.
- **Plugin:** A reusable piece of software that adds a specific functionality to Jenkins. Plugins enable integration with version control systems, build tools, and cloud providers.
- **Build Trigger:** A mechanism to automatically start a build based on specific events, such as code changes in a version control system or scheduled intervals.

4.4.4 Common Commands and Their Usage

Jenkins is primarily managed through its web interface rather than command-line commands, but you can interact with it using the following approaches:

- **Starting Jenkins:**
  To start Jenkins on a local server, navigate to the Jenkins installation directory and use:

  ```
  java -jar jenkins.war
  ```

  This command runs Jenkins on the default port (8080).
- **Installing Plugins:**
  To install a plugin, navigate to `Manage Jenkins` > `Manage Plugins` > `Available` tab in the Jenkins dashboard, search for the desired plugin, and click `Install without restart`.
- **Creating a New Job:**
  To create a new job, click on `New Item` in the Jenkins dashboard, provide a name for the job, select the job type (e.g., Freestyle project or Pipeline), and click `OK` to configure the job.
- **Configuring Build Triggers:**
  Inside a job configuration, you can set build triggers under the `Build Triggers` section. Options include polling the SCM (Source Code Management) or using webhook triggers.
- **Running a Job:**
  To manually run a job, navigate to the job's page and click on `Build Now`. This will trigger the job and start the build process.
- **Accessing Build Logs:**
  After running a job, click on the build number in the job's page to access the build logs and see the output of the build process.
- **Configuring a Pipeline:**
  You can create a Jenkins pipeline by adding a `Jenkinsfile` in the project repository. To configure it in Jenkins, create a new pipeline job and point it to the repository containing the Jenkinsfile.

4.5 Ansible

4.5.1 What is Ansible?

Ansible is an open-source automation tool used for configuration management, application deployment, and task automation. It allows users to define infrastructure as code, making it easier to manage and provision systems across various environments.

4.5.2 Why Use Ansible?

Ansible simplifies the automation of complex IT tasks and helps achieve consistent configurations across multiple systems. Its agentless architecture means it does not require any special software to be installed on target machines, reducing the overhead of management. Ansible uses YAML for its playbooks, making it easy to read and write.

4.5.3 Key Terminologies

- **Playbook:** A YAML file that defines a series of tasks to be executed on one or more target hosts. It specifies the desired state of the system.
- **Inventory:** A file that contains a list of target hosts along with their connection information. It can be static or dynamic.
- **Task:** A single action to be performed on the target hosts, defined within a playbook.
- **Module:** A reusable unit of work that Ansible executes on the target hosts. Ansible has many built-in modules for tasks like package management, file manipulation, and system configuration.
- **Role:** A way to organize playbooks and related files in a structured manner. Roles allow for reusability and sharing of configurations.

4.6 Ngrok and Webhooks

4.6.1 Ngrok

What: Ngrok is a tool that creates secure tunnels to your local machine, providing a temporary public URL to expose a local service to the internet.
Why: Jenkins is running on `localhost`, which is not accessible publicly. GitHub requires a public URL to trigger webhooks, so we use Ngrok to forward requests from a public URL to our local Jenkins instance.
How it works: Ngrok generates a public URL that forwards incoming requests to a specific port on your local machine. In this project, it forwards requests from the generated URL to port 8080 where Jenkins is hosted.

4.6.2 Webhooks

What: Webhooks are automated messages sent from one system to another when certain events happen. In this case, GitHub sends an HTTP POST request to Jenkins when a commit is made.
Why: To automate builds in Jenkins when a new commit is pushed to GitHub, webhooks trigger Jenkins to pull the changes and build the project.
How it works: A webhook is configured in the GitHub repository to send a POST request to the Ngrok public URL whenever a push event occurs. This POST request then triggers Jenkins to initiate the build process.

5 Setup Steps from Scratch

5.1 Installing Required Tools

To set up the development environment, you need to install the following tools:

- **Java Development Kit (JDK)**:

  ```
  sudo apt install openjdk-11-jdk
  ```

- **Maven**:

```
sudo apt install maven
```

- **Git**:

```
sudo apt install git
```

- **Docker**:

```
sudo apt install docker.io
```

- **Jenkins**:

```
sudo apt install jenkins
```

- **Ansible**:

```
sudo apt install ansible
```

- **Ngrok**:

```
sudo apt install ngrok
```

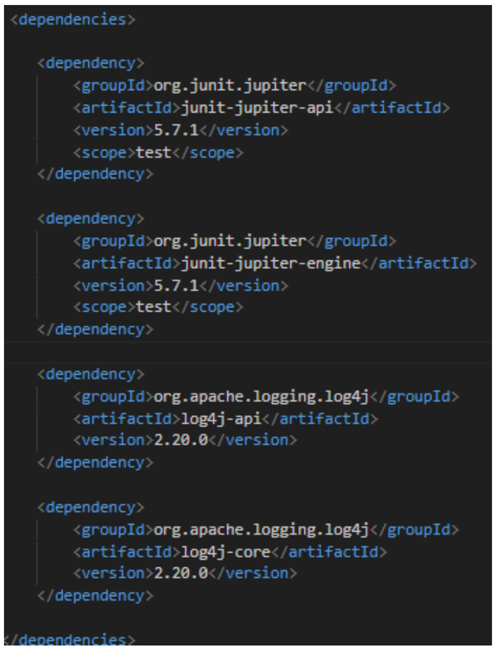
5.2 Installing Jenkins Plugins

To ensure Jenkins can handle our pipeline properly, certain plugins need to be installed. Follow the steps below to install the required plugins:

- Open Jenkins in your browser:

```
http://localhost:8080
```

- Navigate to **Manage Jenkins** → **Manage Plugins**.
- Go to the **Available** tab and search for the following plugins:
  - **Docker Pipeline**: Allows Jenkins to interact with Docker containers during the pipeline.
  - **GitHub Plugin**: For integrating Jenkins with GitHub.
  - **Pipeline**: For setting up and executing Jenkins pipelines.
  - **Ansible**: To enable Jenkins to interact with Ansible during the deployment phase.
- Select the plugins from the list and click **Install without restart**.
- Once the plugins are installed, Jenkins is ready to run the pipeline.



```xml
<dependencies>

  <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.7.1</version>
      <scope>test</scope>
  </dependency>

  <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-engine</artifactId>
      <version>5.7.1</version>
      <scope>test</scope>
  </dependency>

  <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
      <version>2.20.0</version>
  </dependency>

  <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
      <version>2.20.0</version>
  </dependency>

</dependencies>
```

Figure 3: `pom.xml` Dependencies

5.3 Creating the Maven Project

The first step was to create a Maven project using the Maven extension in Visual Studio Code (VS Code). Once the project was created, I followed the typical Maven folder structure, adding code files under the `src/main/java` directory and test files under `src/test/java`.

After adding the necessary code and test files, I configured the project's dependencies by editing the `pom.xml` file. The `pom.xml` defines the project's dependencies, which were essential for managing external libraries and handling test cases. The following figure shows the dependencies used.
Dependencies added:

- **JUnit 5.7.1**: Used for writing and running unit tests. The two dependencies for JUnit (`junit-jupiter-api` and `junit-jupiter-engine`) provide the testing API and the engine for running the tests.
- **Log4j 2.20.0**: Provides the logging functionality to track application behavior during execution. Two dependencies are used: `log4j-api` and `log4j-core` to fully support logging.

5.4 Initializing Git Repository and Setting Remote

Once the Maven project was set up, I initialized a Git repository in the project directory using the following command:

```
git init
```

After initializing the Git repository, a remote GitHub repository was created. The local repository was linked to the remote GitHub repository using the `git remote` command:

```
git remote add origin https://github.com/RamsaiKoushik/Calculator-SPE-Miniproject.git
```

This setup ensures that the local codebase is synchronized with the remote repository on GitHub for version control and collaboration purposes.

5.5 Docker, Jenkins, and Ansible Configuration

After setting up the Maven project and pushing it to GitHub, I wrote the Dockerfile, Jenkinsfile, and Ansible playbook for deployment automation. Below are the respective files and the steps involved:

5.5.1 Dockerfile

The Dockerfile defines the build process for the Docker image of our Maven project. Here is the Dockerfile:

```
FROM openjdk:11

COPY ./target/demo-1.0-SNAPSHOT.jar ./

WORKDIR ./

CMD ["java", "-jar", "demo-1.0-SNAPSHOT.jar"]
```

This Dockerfile uses the official **OpenJDK 11** image as a base, copies the Maven-built JAR file into the container, sets the working directory, and defines the command to run the application.

5.5.2 Jenkinsfile

The Jenkinsfile defines the build and deployment pipeline. It consists of various stages for building the Maven project, creating and pushing the Docker image, cleaning up Docker images, and deploying using Ansible. Below is the Jenkinsfile:

```
pipeline{
    environment{
        DOCKERHUB_CRED = credentials("DockerHub-Credentials-ID")
    }
    agent any
    stages{
        stage("Stage 1 : Maven Build"){
            steps{
                sh 'mvn clean install'
            }
        }

        stage("Stage 2: Build Docker Image"){
            steps{
                sh "docker build -t ramsai987/calculator:latest ."
            }
        }

        stage("Stage 3: Push Docker Image to Dockerhub"){
            steps{
                sh 'echo $DOCKERHUB_CRED_PSW | docker login -u $DOCKERHUB_CRED_USR --password-stdin'
                sh "docker push ramsai987/calculator:latest"
            }
        }

        stage("Stage 4: Clean Unwanted Docker Images"){
            steps{
                sh "docker container prune -f"
                sh "docker image prune -a -f"
            }
        }

        stage('Stage 5: Ansible Deployment') {
            steps {
                ansiblePlaybook colorized: true,
                credentialsId: 'localhost',
                installation: 'Ansible',
                inventory: 'inventory',
                playbook: 'deploy-calculator.yml'
            }
        }
    }
}
```

- **Stage 1:** Runs the Maven build command `mvn clean install`.
- **Stage 2:** Builds a Docker image tagged as `ramsai987/calculator:latest`.
- **Stage 3:** Logs in to DockerHub and pushes the Docker image.
- **Stage 4:** Cleans up any unwanted Docker containers and images.
- **Stage 5:** Deploys the Docker container using Ansible.

5.5.3 Ansible Playbook and Inventory

The Ansible playbook automates the process of deploying the Docker container. Here's the **deploy-calculator.yml** playbook:

```
---
- name: Pull Docker Image of Calculator
  hosts: all

  tasks:
    - name: Start Docker Service
      service:
        name: docker
        state: started

    - name: Pull Image
      shell: docker pull ramsai987/calculator:latest

    - name: Run the container
      shell: docker run -it --name Calculator ramsai987/calculator:latest
```

Explanation of Ansible Playbook:

- **hosts: all:** This specifies that the tasks will be run on all hosts listed in the **inventory** file. In this case, we are deploying locally, so the host is set to `127.0.0.1`.
- **Start Docker Service:** This task ensures that the Docker service is running on the target machine before we attempt to pull images or run containers. The `service` module in Ansible is used to manage system services, ensuring Docker is started.
- **Pull Image:** This task uses the `shell` module to run a command that pulls the latest version of the `ramsai987/calculator` image from DockerHub. This ensures that the most up-to-date image is deployed on the machine.

- **Run the container:** Finally, this task runs the Docker container using the `docker run` command. The container is named `Calculator`, and it is run interactively (`-it`) to allow communication with it. This effectively starts the application inside the container.

The corresponding **inventory** file is simple, as the deployment is being done locally:

```
[localhost]
127.0.0.1 ansible_connection=local ansible_user=ram
```

This tells Ansible to use the local machine (`127.0.0.1`) for deployment, with the user set to `ram`.

5.6 Setting Up DockerHub, Jenkins, Ngrok, and Webhooks

5.6.1 Creating a DockerHub Account and Using Credentials in Jenkins

To push the Docker image created by Jenkins to DockerHub, we first need to create a DockerHub account:

- Go to **docker.com** and sign up for an account if you don't already have one.
- Once your account is created, log in to DockerHub.
- In Jenkins, you need to store your DockerHub credentials securely:
  - Open Jenkins and navigate to **Manage Jenkins** → **Credentials**.
  - Under **Global credentials**, click **Add Credentials**.
  - Choose **Username with password**, and enter your DockerHub username and password.
  - Assign an ID (e.g., `DockerHub-Credentials-ID`) to the credentials for later use in the Jenkinsfile.

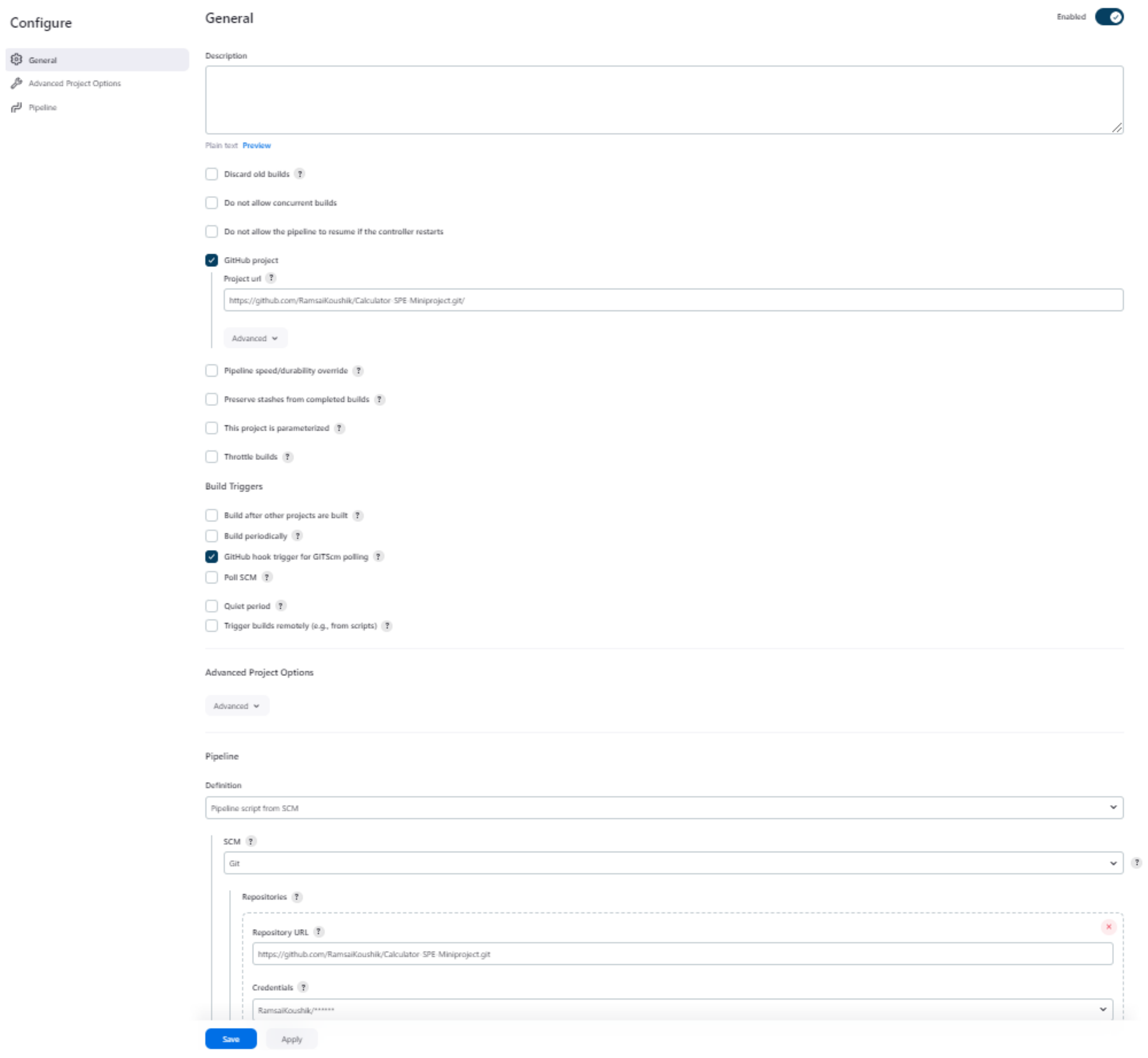
The credentials can now be referenced in the Jenkins pipeline using the ID we defined in the Jenkinsfile:

```
environment{
    DOCKERHUB_CRED = credentials("DockerHub-Credentials-ID")
}
```

5.6.2 Configuring a Jenkins Project for Webhook Triggers

Next, we need to set up Jenkins to automatically trigger builds whenever changes are pushed to the GitHub repository. This is done using GitHub webhooks:

- In Jenkins, create a new pipeline project:
  - Click on **New Item**, select **Pipeline**, and give the project a name.
  - Under the **General** section, check the box for **GitHub Project** and provide the URL of your GitHub repository (e.g., `https://github.com/RamsaiKoushik/Calculator-SPE-Miniproject.git`) in the **Project URL** field.
  - Under the **Pipeline** section, set the pipeline definition to `Pipeline script from SCM`.
  - Choose `Git` as the source, and again provide the URL of your GitHub repository.
  - In the **Branch Specifier**, specify the branch (`*/main`) you want Jenkins to monitor.
- **Add a new pipeline project in Jenkins** as shown in the following image 4:
- Configure the project to trigger builds via webhooks:
  - Under **Build Triggers**, check the option for **GitHub hook trigger for GITScm polling**.


Configure

General

Enabled

Description

Plain text Preview

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url: https://github.com/RamsaiKoushik/Calculator-SPE-Miniproject.git/

- [ ] Pipeline speed/durability override
- [ ] Preserve stashes from completed builds
- [ ] This project is parameterized
- [ ] Throttle builds

Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [x] GitHub hook trigger for GITScm polling
- [ ] Poll SCM
- [ ] Quiet period
- [ ] Trigger builds remotely (e.g., from scripts)

Advanced Project Options

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/RamsaiKoushik/Calculator-SPE-Miniproject.git

Credentials: RamsaiKoushik/******

Figure 4: Creating a new pipeline project in Jenkins

5.6.3 Setting Up Ngrok for Public URL Access

Since Jenkins is hosted locally, GitHub needs a public URL to communicate with the Jenkins server. This is where **ngrok** comes into play. Ngrok provides a public URL that can forward requests to your local Jenkins server.

To set up ngrok:

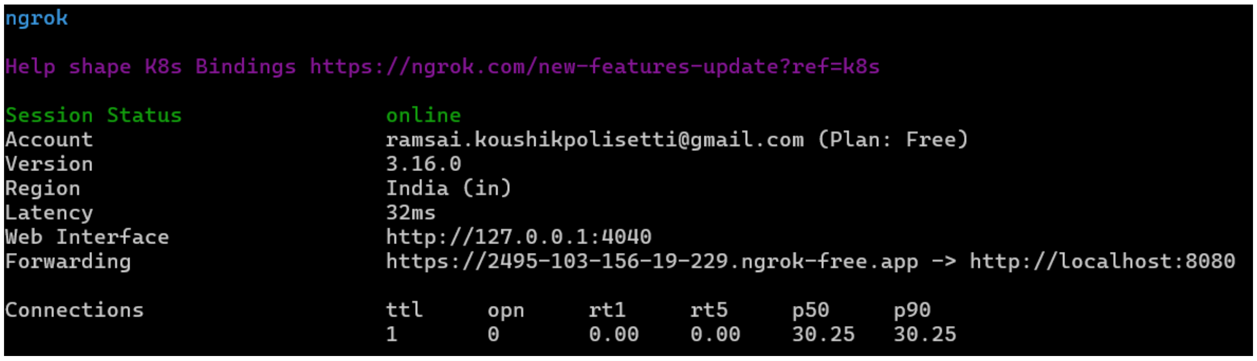
- In the terminal, run the following command to expose your Jenkins server:

  ```
  ngrok http 8080
  ```

  This command will expose port 8080 (the default Jenkins port) to the internet.

- Ngrok will provide a public URL (e.g., `http://1234abcd.ngrok.io`) that you can use in GitHub Webhooks.

- **Use the public URL from ngrok** as shown in the following image 5:



```
ngrok

Help shape K8s Bindings https://ngrok.com/new-features-update?ref=k8s

Session Status                online
Account                       ramsai.koushikpolisetti@gmail.com (Plan: Free)
Version                       3.16.0
Region                        India (in)
Latency                       32ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://2495-103-156-19-229.ngrok-free.app -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.00    0.00    30.25   30.25
```

Figure 5: Setting up ngrok for public access to Jenkins

5.6.4 Setting Up GitHub Webhook

Once ngrok is set up and providing a public URL, you can configure GitHub to send webhook notifications to Jenkins:

- In your GitHub repository, go to **Settings** → **Webhooks**.
- Click **Add webhook**, and paste the public URL provided by ngrok into the **Payload URL** field.
- Append `/github-webhook/` at the end of the URL (e.g., `http://1234abcd.ngrok.io/github-webhook/`).
- Set the **Content type** to `application/json`, and select **Just the push event**.
- This will trigger Jenkins to build the project automatically whenever you push new changes to the repository.
- Here is the screenshot(6) showing how to configure webhooks:

5.6.5 Pipeline Overview

The overall pipeline consists of the following stages:

- **Stage 1: Maven Build** – Compiles and builds the project using Maven.
- **Stage 2: Docker Image Build** – Builds the Docker image of the project.
- **Stage 3: DockerHub Push** – Pushes the Docker image to DockerHub.
- **Stage 4: Docker Cleanup** – Cleans up unused Docker containers and images.
- **Stage 5: Ansible Deployment** – Uses Ansible to deploy the container to the specified target.

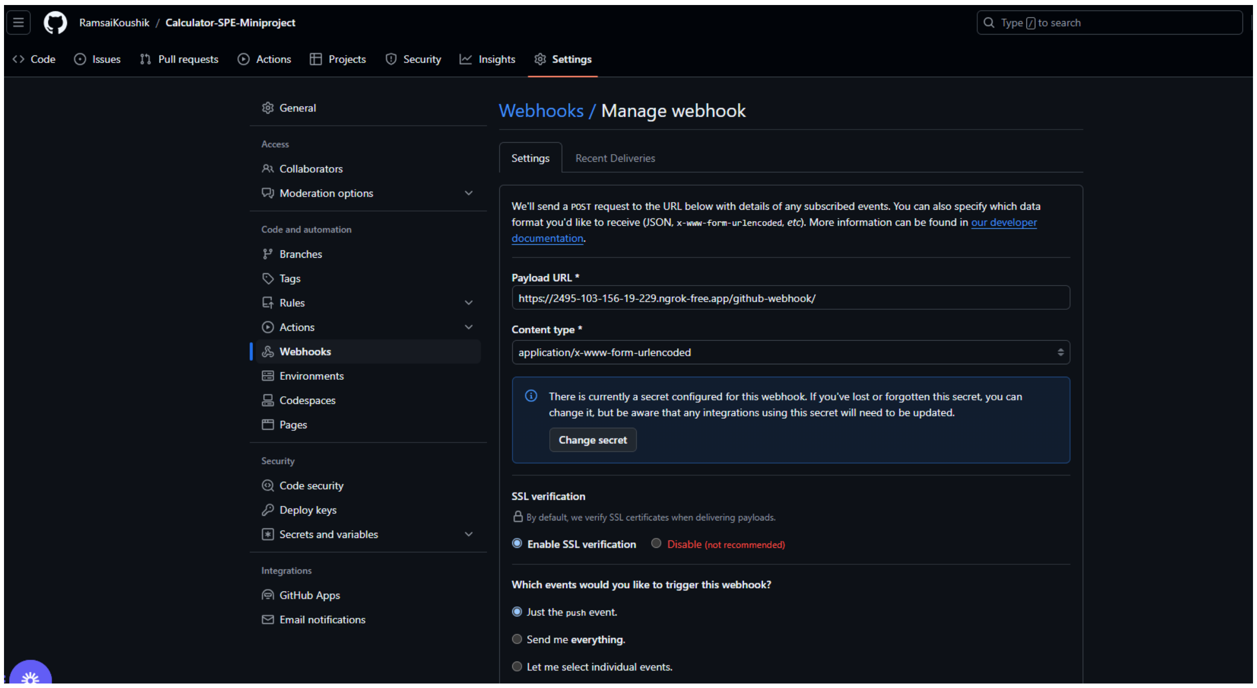
Overall Pipeline Flow: Image 7

5.6.6 Running the Docker Container Manually

After the Jenkins pipeline completes the build and deployment, the Docker container can be started manually on the target machine using the following command:

```
docker start -a -i Calculator
```

This command starts the `Calculator` container, attaching the terminal to its logs (`-a`) and running it interactively (`-i`).



Webhooks / Manage webhook

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *
https://2495-103-156-19-229.ngrok-free.app/github-webhook/

Content type *
application/x-www-form-urlencoded

There is currently a secret configured for this webhook. If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated.

SSL verification
By default, we verify SSL certificates when delivering payloads.
[✓ Enable SSL verification] Disable (not recommended)

Which events would you like to trigger this webhook?
[✓ Just the `push` event.]
Send me **everything.**
Let me select individual events.

Figure 6: Configuring webhooks in github

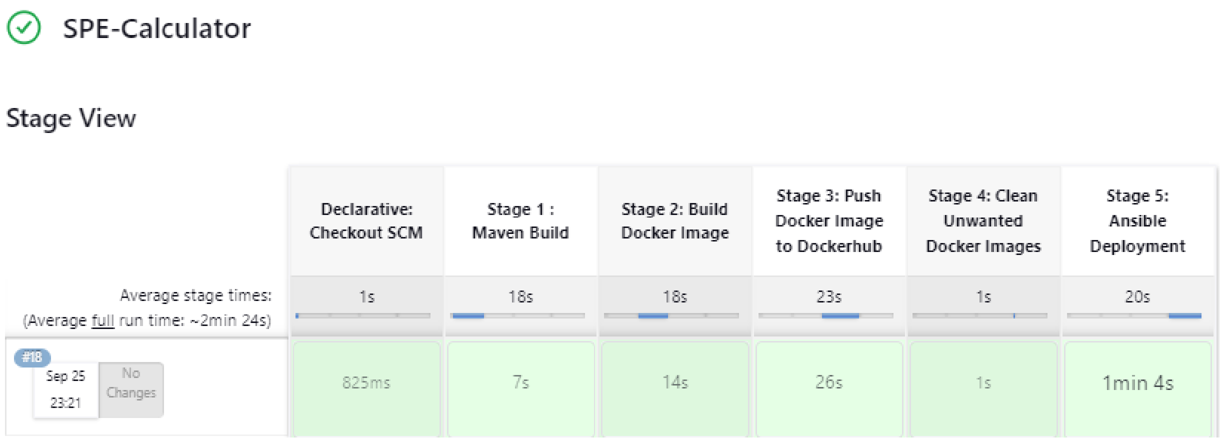
5.7 Running the Docker Container

Once the build and deployment process is complete, the Docker container can be manually started using the following command:

```
docker start -a -i Calculator
```

This command attaches the terminal to the running container, displaying the logs interactively. Here, the `-a` flag attaches to the container, and `-i` keeps it interactive, allowing you to see real-time logs and interact with the container's process.

Here is a screenshot of the execution (8):

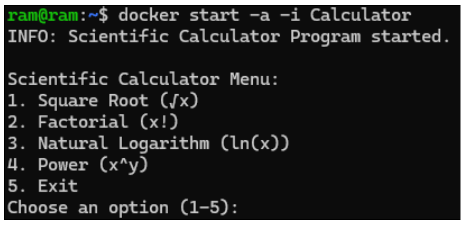


SPE-Calculator

Stage View

	Declarative: Checkout SCM	Stage 1 : Maven Build	Stage 2: Build Docker Image	Stage 3: Push Docker Image to Dockerhub	Stage 4: Clean Unwanted Docker Images	Stage 5: Ansible Deployment
Average stage times: (Average full run time: ~2min 24s)	1s	18s	18s	23s	1s	20s
#18 Sep 25 23:21 No Changes	825ms	7s	14s	26s	1s	1min 4s

Figure 7: Pipeline Overview in Jenkins



```
ram@ram:~$ docker start -a -i Calculator
INFO: Scientific Calculator Program started.

Scientific Calculator Menu:
1. Square Root (√x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power (x^y)
5. Exit
Choose an option (1-5):
```

Figure 8: Execution of Docker Container in Terminal