International Institute of Information Technology, Bangalore

Software Production Engineering (CSE-816)

Mini Project Report

Scientific Calculator

Submitted by

Abhay Bhadouriya (MT2024003)

Under the guidance of **Prof. B. Thangaraju**

Table of Contents

DevOps (Development & Operations)

1.1 What do you mean by DevOps?

DevOps refers to a collection of methods, tools, and a cultural mindset aimed at streamlining and unifying the workflows of software development and IT operations teams. It focuses on empowering team members, fostering communication and cooperation across groups, and leveraging automation technologies.

1.2 How Does DevOps Work in world of IT?

A DevOps team consists of developers and IT operations staff who work together throughout the entire product lifecycle to enhance both the speed and quality of software releases. This approach represents a fresh working style and a cultural transformation that profoundly impacts teams and their organizations.

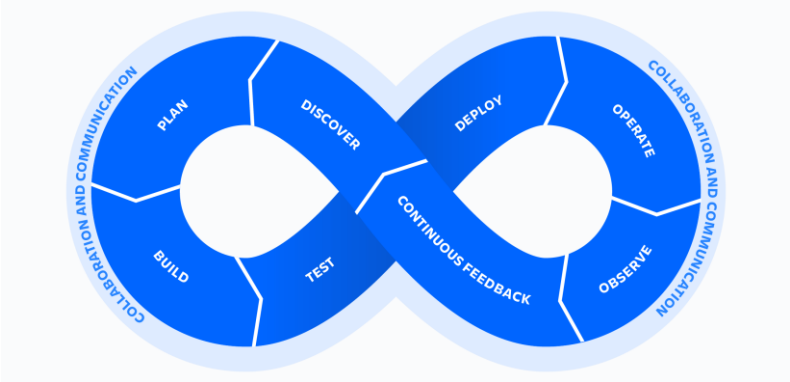
In the DevOps framework, development and operations units no longer operate in isolation. In some cases, these groups combine into one cohesive team where engineers handle all phases of the application process—from coding and testing to deployment and maintenance—bringing a diverse set of skills to the table.

DevOps teams employ tools to streamline and speed up processes, boosting dependability. A DevOps toolchain supports teams in addressing key DevOps principles such as continuous integration, continuous delivery, automation, and teamwork.

1.3 DevOps Lifecycle

Due to the ongoing nature of DevOps, professionals employ the infinity loop to illustrate the connections between the stages of the DevOps lifecycle. Though it may seem to progress in a linear fashion, the loop represents the necessity for ongoing teamwork and continuous enhancement across all phases.

- Discover
- Plan
- Build
- Test
- Deploy
- Operate
- Observe
- Continuous Feedback



1.4 Benefits of using DevOps and why we should use it

- Here's a rewritten version of your text, avoiding plagiarism while preserving the core ideas:
-
- **Velocity:** Operate at a fast pace to accelerate innovation for customers, respond effectively to shifting market demands, and enhance efficiency in achieving business goals. The DevOps approach empowers development and operations teams to deliver these outcomes. For instance, practices like microservices and continuous delivery enable teams to take charge of services and roll out updates more swiftly.
- **Swift Releases:** Boost the frequency and speed of deployments to drive innovation and refine your product rapidly. The sooner you introduce new features or resolve issues, the quicker you can meet customer expectations and gain a competitive edge. Techniques such as continuous integration and continuous delivery streamline the software release process, automating everything from creation to deployment.
- **Scalability:** Manage your infrastructure and development workflows on a large scale. Consistency and automation allow you to handle intricate or evolving systems with greater efficiency and lower risk. For example, infrastructure as code enables repeatable, streamlined management of development, testing, and production environments.
- **Enhanced Teamwork:** Foster stronger teams within a DevOps culture that prioritizes ownership and responsibility. By working closely together, developers and operations staff share tasks and integrate their processes, minimizing inefficiencies and saving time. This collaboration eliminates delays, such as lengthy handoffs, and encourages coding that aligns with the runtime environment.
- **Dependability:** Maintain the quality of application updates and infrastructure adjustments to deliver consistently at a faster pace while ensuring a seamless user experience. Practices like continuous integration and continuous delivery allow you to verify that changes are both functional and secure. Real-time monitoring and logging keep you informed about performance trends.
- **Protection:** Achieve speed without compromising oversight or regulatory adherence. A DevOps framework can incorporate security through automated compliance measures, precise controls, and configuration management tools. For instance, leveraging infrastructure as code and policy as code helps you establish and monitor compliance efficiently across large-scale operations.

Tools & Technologies used in this mini Project

- **Coding Language:** Java
- **Version Management System:** Git
- **Container Technology:** Docker
- **CI/CD Solution:** Jenkins
- **Testing Platform:** JUnit 5
- **Deployment Software:** Ansible

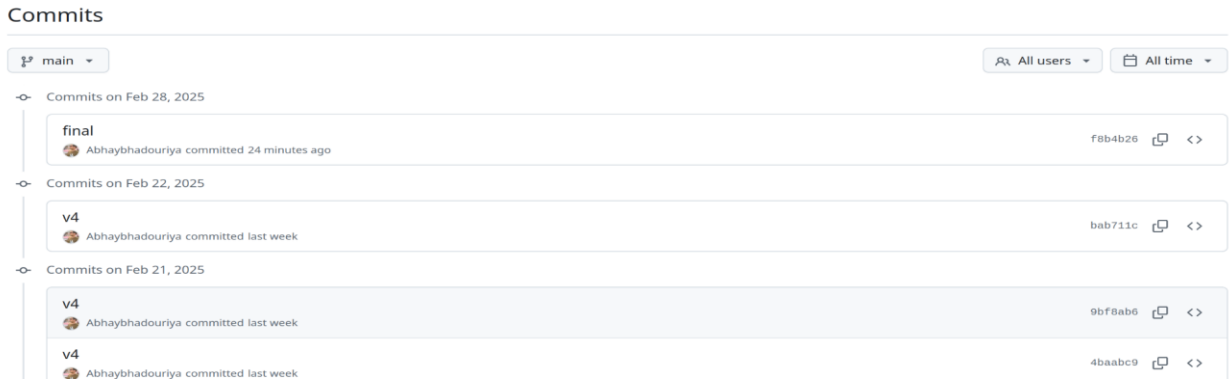
Software Development Life Cycle (SDLC)

3.1 Source Code Management

Source control involves the process of monitoring and organizing modifications to code. Source control management (SCM) tools maintain a detailed record of code evolution and assist in resolving conflicts that arise when integrating changes from various contributors.

For this project, I utilized Git as the SCM system. GitHub served as the online platform for hosting the repository.
The program was developed in a step-by-step manner. Below are some key milestones from the development process, with additional details available at https://github.com/Abhaybhadouriya/miniProject_SPE/commits/main/.

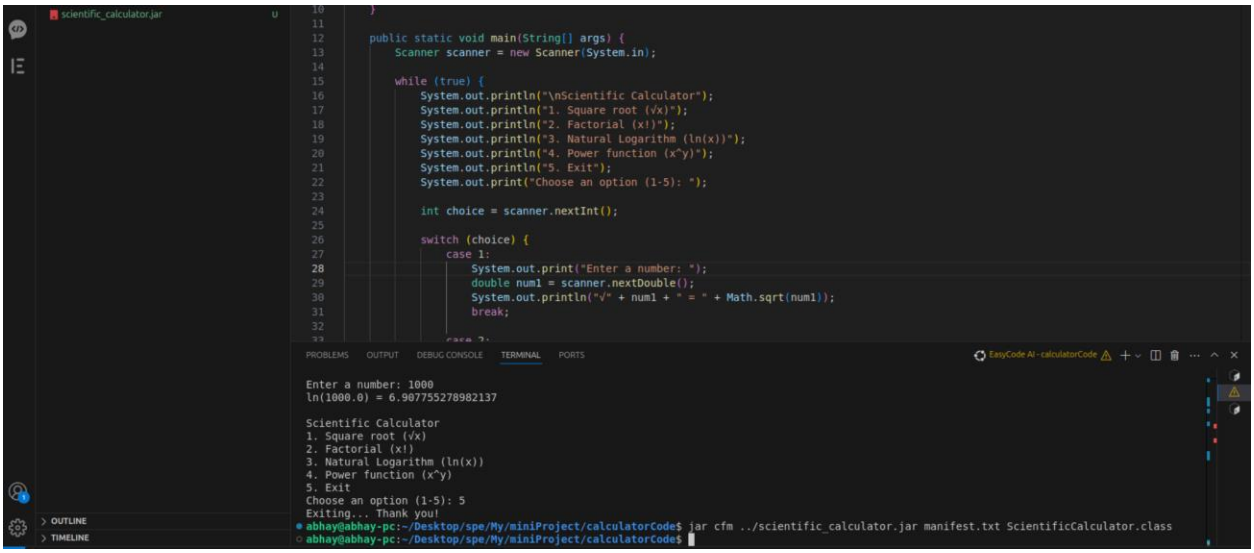
1. Initially, I implemented the addition operation.
2. Created a Dockerfile and Jenkinsfile to support the addition functionality.
3. After confirming everything worked smoothly, I added subtraction, multiplication, and division operations.
4. Next, I incorporated square root, factorial, natural logarithm, and power functions.
5. Included unit test cases to validate the code.
6. Developed an Ansible playbook for deployment.



Github Link: https://github.com/Abhaybhadouriya/miniProject_SPE/commits/main/

3.2 Build

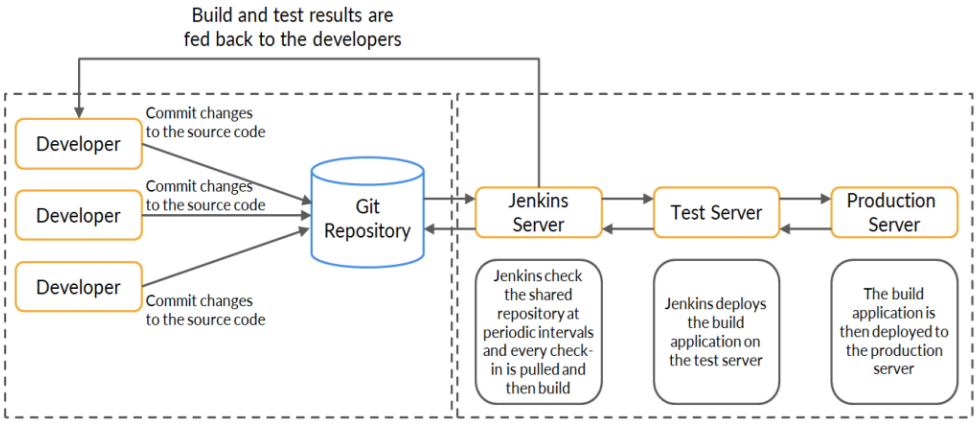
In the calculator program, Creating the Jar file



3.3 what is Continuous Integration (CI) and where we are using?

Continuous Integration (CI) involves merging newly written code into the existing codebase as soon as it's developed. This process also encompasses automatically building the project and executing test cases, as outlined earlier. For this purpose, I employed Jenkins as the CI tool.

Jenkins is a widely used, open-source automation server available at no cost. It supports the automation of various software development activities, including building, testing, and delivering or deploying applications. Jenkins can be set up through native system packages, run within Docker, or operated independently on any machine.



Jenkins Architecture

3.3.1 Jenkins Installation

- ### Jenkins Setup Instructions

- - Install OpenJDK 17:
- `sudo apt install -y openjdk-17-jdk`

- - Import Jenkins GPG Key:
- `curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`

- - Add Jenkins Repository:
- `echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/" | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`

- - Refresh Package List and Install Jenkins:
- `sudo apt update && sudo apt install -y jenkins`

- - Launch Jenkins Service:
- `sudo systemctl start jenkins.service`

- - Access Jenkins in a Browser:
- Open `http://localhost:8080`

- - Obtain the Initial Admin Password:
- `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

- - Proceed with the Setup Wizard:
- Enter the retrieved password.

- - Add Recommended Plugins:
- Install the suggested plugins during setup.

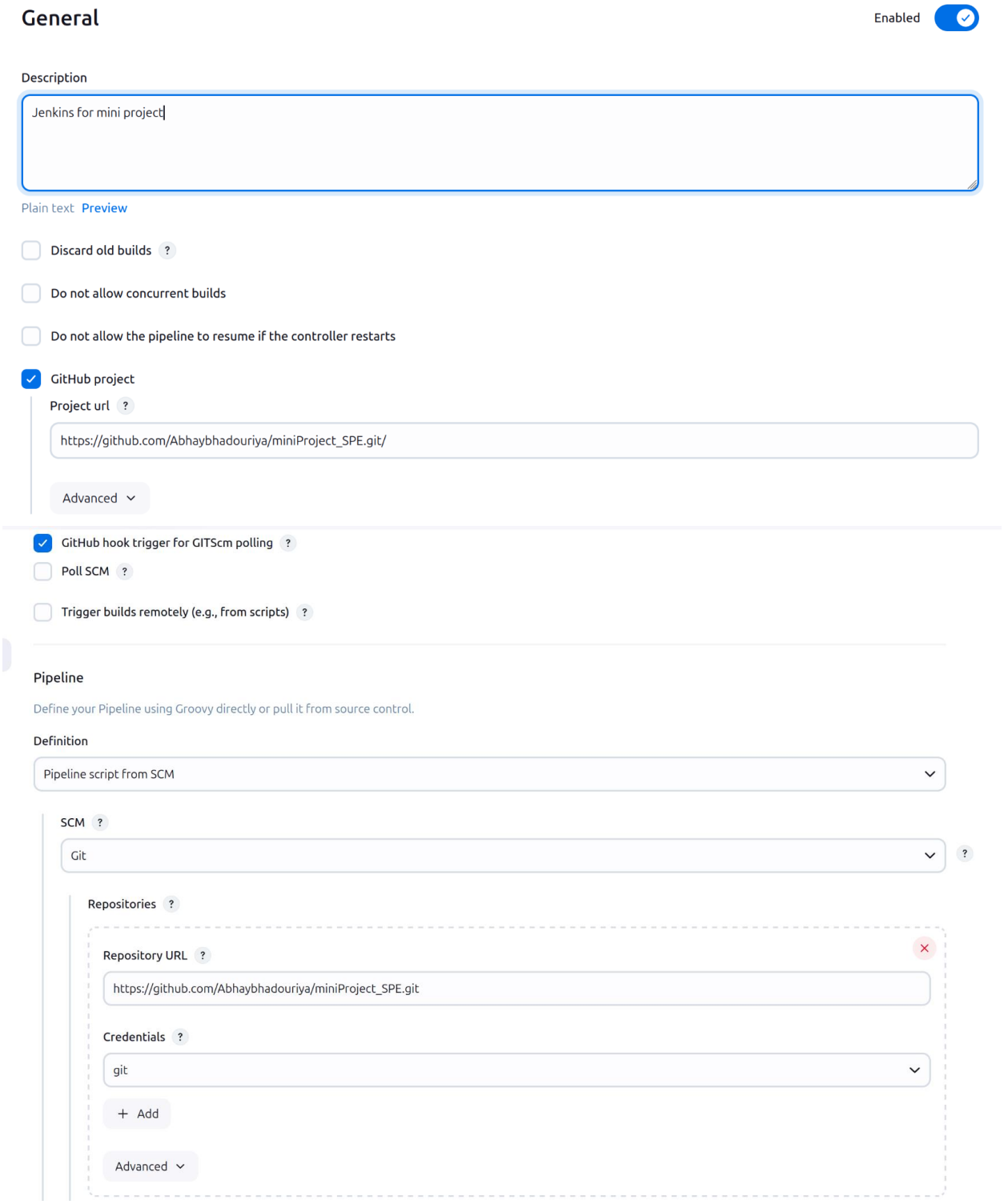
- - Finalize Configuration:
- Complete the setup process to reach the Jenkins dashboard.

3.3.2 Jenkins Pipeline

A Jenkins pipeline provides a visual overview of the stages in a CI/CD workflow. It connects various Jenkins jobs and lets you track their status during execution.

To set up a Jenkins pipeline, use these steps:

1. On the Jenkins main page, find the project list and click the ‘+’ button above it.
2. Input a name for the pipeline view, an optional description, and the first job to include.



The Jenkins Pipeline includes several stages such as Checkout, Build Docker Image, Push Docker Image, and Run Ansible Playbook, followed by a post-action step to send an email notifying the pipeline's status.

Abhaybhadouriya final

Code | Blame | 84 lines (78 loc) · 2.52 KB

```
pipeline {
    agent any
    triggers {
      githubPush()
    }
    environment {
        DOCKER_IMAGE_NAME = 'calculator'
        GITHUB_REPO_URL = 'https://github.com/Abhaybhadouriya/miniProject_SPE.git'
    }

    stages {
        stage('Checkout') {
            steps {
                script {
                     checkout([
                    // Checkout the code from the GitHub repository
                        $class: 'GitSCM',
                        branches: [[name: '*/main']],
                        userRemoteConfigs: [[
                            url: "${GITHUB_REPO_URL}",
                            credentialsId: 'git'
                        ]]
                     ])
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    // Build Docker image
                    docker.build("${DOCKER_IMAGE_NAME}", '.')
                }
            }
        }

        stage('Push Docker Images')    {
            steps {
                script{
                    docker.withRegistry('', 'DockerHubCred') {
```

Jenkins File

We established a GitHub project in Jenkins, providing the URL of our GitHub repository and indicating the Jenkinsfile path within the configuration.

System credentials were configured for Docker Hub and the local machine to support the Ansible playbook. Since the project runs locally and retrieves the image from Docker Hub, we defined these system settings in Jenkins credentials, including one specifically for the Docker Hub repository.

To set Docker Hub credentials we have to specify the username of Docker hub from where we have to pull the image.

Docker Hub Credential

3.4 Continuous Delivery (CD)

In software engineering, a deliverable is a finalized artifact prepared for client handover, representing the culmination of the Software Development Life Cycle (SDLC).

Continuous Delivery (CD) involves producing deliverables immediately following code modifications. This practice depends on an established Continuous Integration (CI) pipeline, making CI a necessary foundation for CD.

In this case, the deliverable is a Docker image, which encapsulates all components needed to execute the project, such as the operating system, OpenJDK, and Tomcat server.

For building the CD pipeline, I utilized Docker and Jenkins as the primary tools.

3.4.1 Docker Installation

To install Docker, follow the given steps:

To set up Docker, follow these steps:

1. `sudo apt-get update`

2. `sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common`

3. `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`

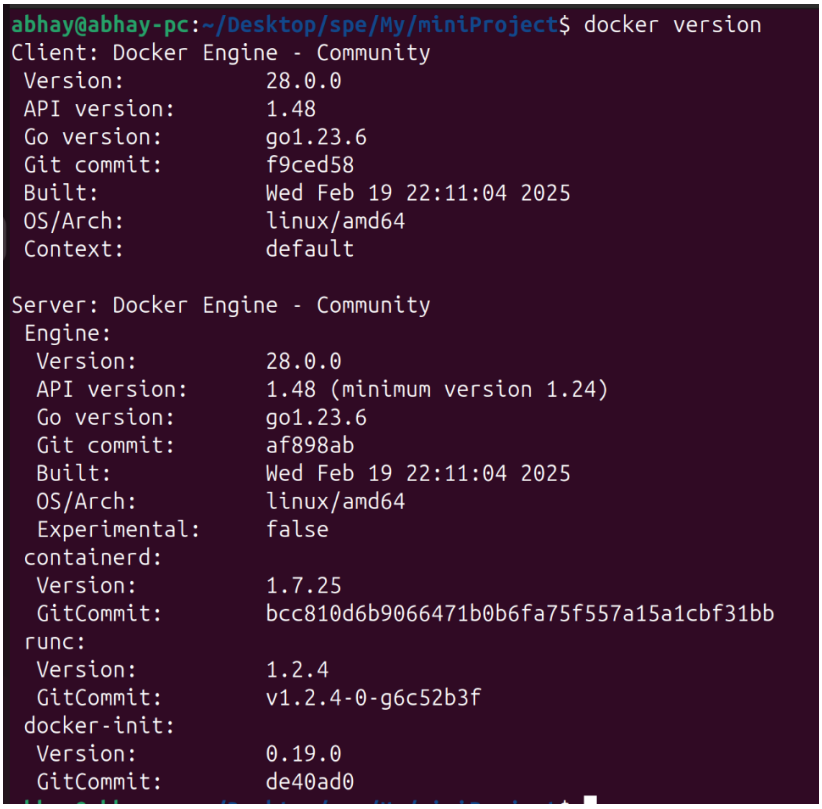
4. `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`

5. `sudo apt update`

6. `sudo apt install docker-ce docker-ce-cli containerd.io`

7. `sudo groupadd docker`

8. `sudo usermod -aG docker <user_name>`



3.4.2 Building and Publish Docker Image

A Dockerfile is a text file that includes a set of instructions a user can execute via the command line to construct a Docker image.

For this project, I began by creating a Dockerfile, which I then placed in the root directory of the project.

Abhaybhadouriya v4

Code | Blame | 17 lines (12 loc) · 478 Bytes

```
1   # Use the official Ubuntu base image
2   FROM ubuntu:latest
3
4   # Set the working directory
5   WORKDIR /app
6
7   # Install Java (required to run the JAR file)
8   RUN apt-get update && apt-get install -y openjdk-21-jdk && rm -rf /var/lib/apt/lists/*
9
10  # Copy the JAR file to the container
11  COPY scientific_calculator.jar /app/scientific_calculator.jar
12
13  # Expose any required ports (if applicable)
14  EXPOSE 8080
15
16  # Set the command to run the JAR file
17  CMD ["java", "-jar", "scientific_calculator.jar"]
```

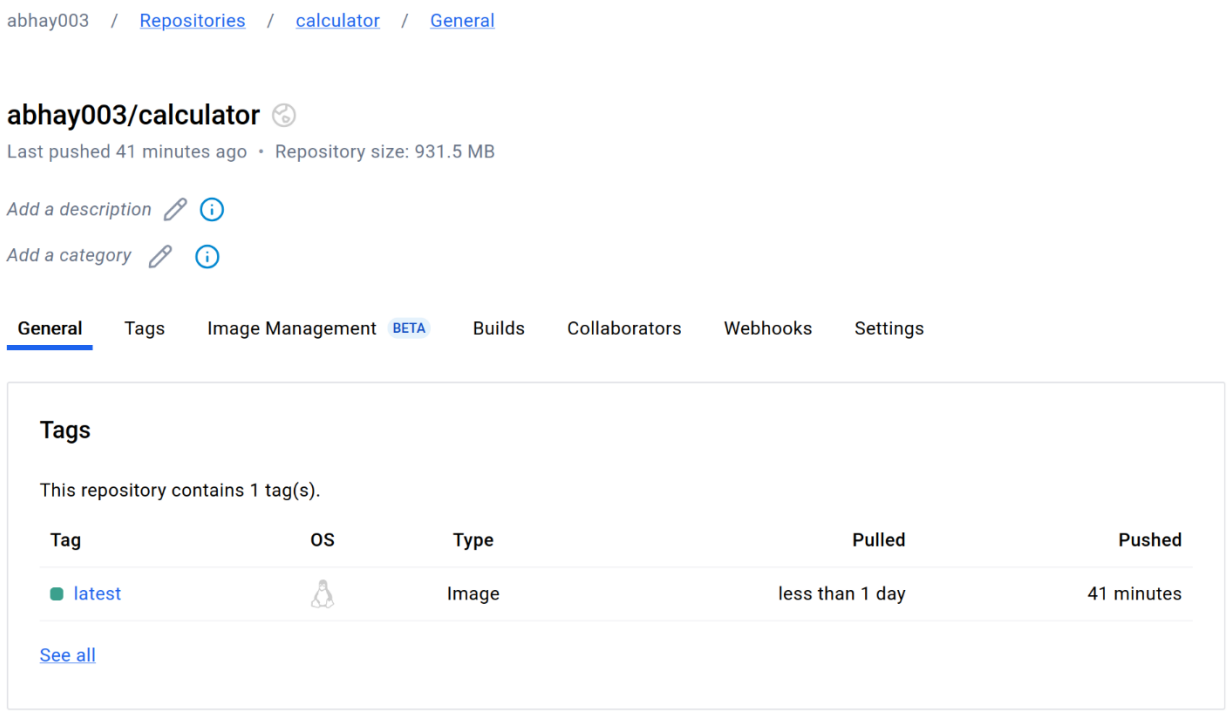
Docker file: https://github.com/Abhaybhadouriya/miniProject_SPE/blob/main/Dockerfile

Build Docker Image: *docker build -t abhay003/calculator .*

Push Docker Image to Docker hub:

docker login

docker push abhay003/calculator



3.5 Continuous Deployment (CDep)

Continuous deployment is a software release approach where any code commit that successfully passes automated testing is immediately deployed to the production environment, making updates visible to users in real time.

In my case, after generating the deliverable (an image), it is published to Docker Hub. For continuous deployment, I utilized Ansible.

Ansible is a tool for automating continuous deployment. It consists of a control node, which manages the deployment process, and multiple managed nodes where the deliverables are deployed.

Ansible needs to be installed only on the control node. It connects to the managed nodes via SSH. The deployment process is defined using a .yml file, known as a playbook, which contains the set of commands to be executed on the managed nodes. Additionally, an inventory file is used to specify details about the managed nodes.

To run the playbook, use the following command:

ansible-playbook <playbook> -i <inventory>

inventory file: https://github.com/Abhaybhadouriya/miniProject_SPE/blob/main/inventory

The Ansible inventory file specifies the hosts and groups of hosts where commands, modules, and tasks in a playbook are executed. Its format may vary depending on the Ansible environment and plugins used.

Ansible playbooks are used to define and execute configuration management, deployment, and orchestration tasks.

To run the application's Docker image, Python pip and Docker must be installed on the host. The playbook pulls the required Docker image from Docker Hub and launches a container on the specified hosts.

miniProject_SPE / deploy.yml

Abhaybhadouriya v4

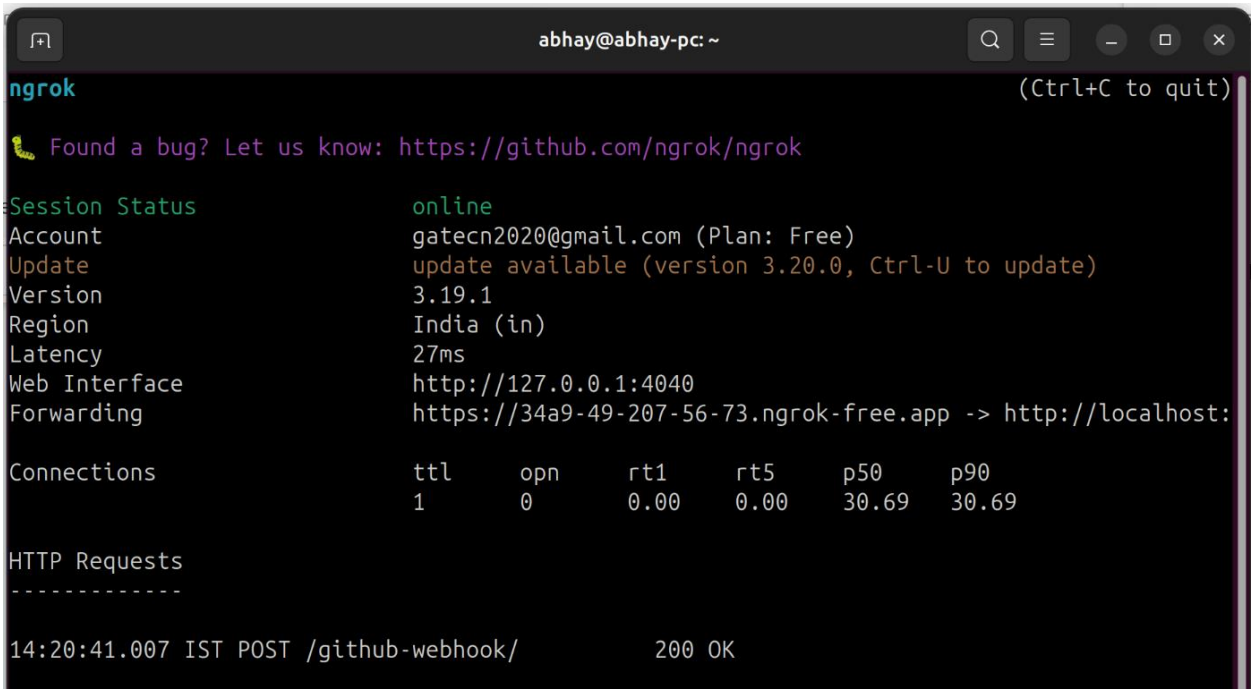
Code | Blame | 36 lines (30 loc) · 833 Bytes

```
---
- name: Pull Docker Image from Docker Hub
  hosts: localhost
  remote_user: abhay
  become: false
  tasks:

    - name: Install Python Docker SDK
      apt:
        name: python3-docker
        state: present
      become: false


    - name: Pull Docker Image
      docker_image:
        name: "abhay003/calculator:latest"
        source: pull
      register: docker_pull_result

    - name: Display Docker Pull Result
      debug:
        var: docker_pull_result

    - name: Start Docker service
      service:
        name: docker
        state: started
    - name: Stop and remove the existing container if it exists
      shell: |
        docker rm -f calci|| true
      ignore_errors: true

    - name: Run Docker container inside container
      shell: >
        docker run -it -d --name calci abhay003/calculator
```

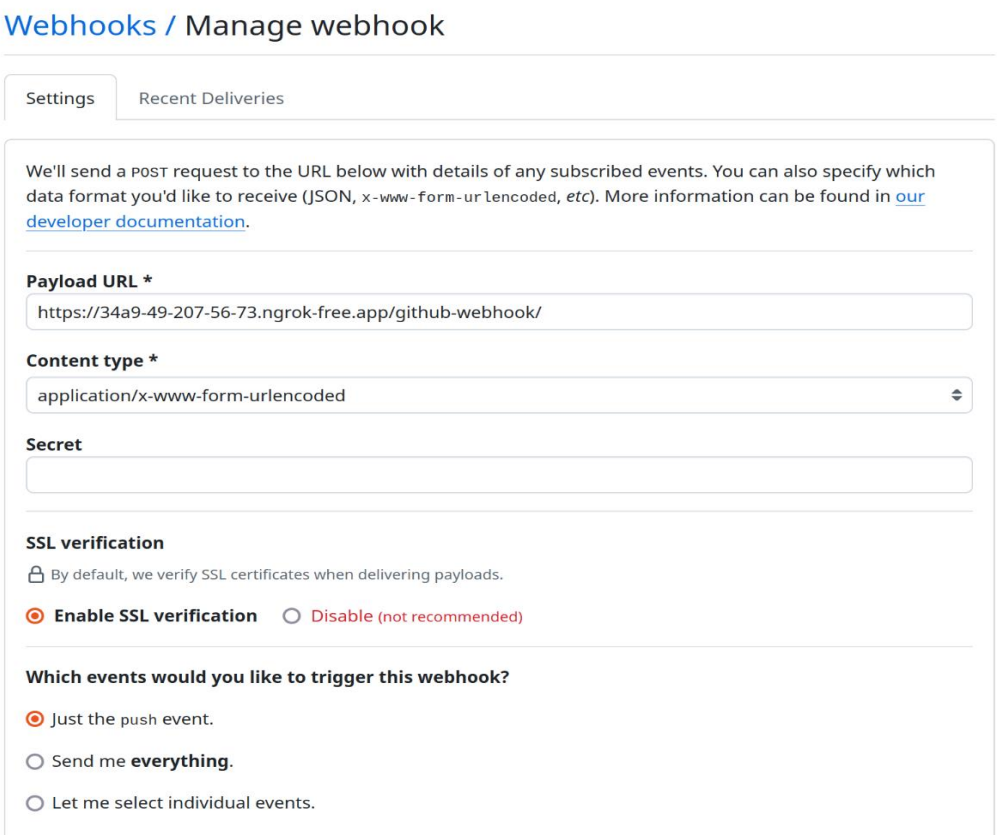
playbook.yml file: https://github.com/Abhaybhadouriya/miniProject_SPE/blob/main/deploy.yml

3.5.1 Webhook Configure

Start NGROk in cmd type-> ngrok http 8080



Configure Github by copying the ngrok forwording url



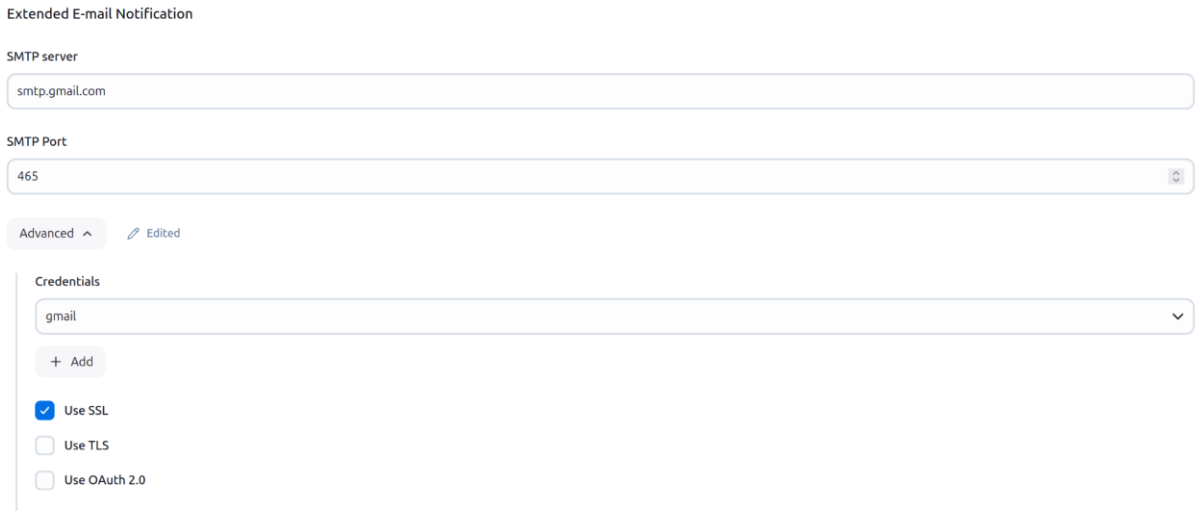
Project Run and Source Code

To run the project as pipeline in Jenkins we have to click build now to see each stages of pipeline and show the output of each stages.

Stage View

Configure Email Notification

Add these field in Extended Email Notification



Now add these creds in Email Notification

E-mail Notification

SMTP server

smtp.gmail.com

Default user e-mail suffix ?

Advanced Edited

Use SMTP Authentication ?

User Name

sbhadouriya39@gmail.com

Password

Concealed Change Password

Use SSL ?

Use TLS

SMTP Port ?

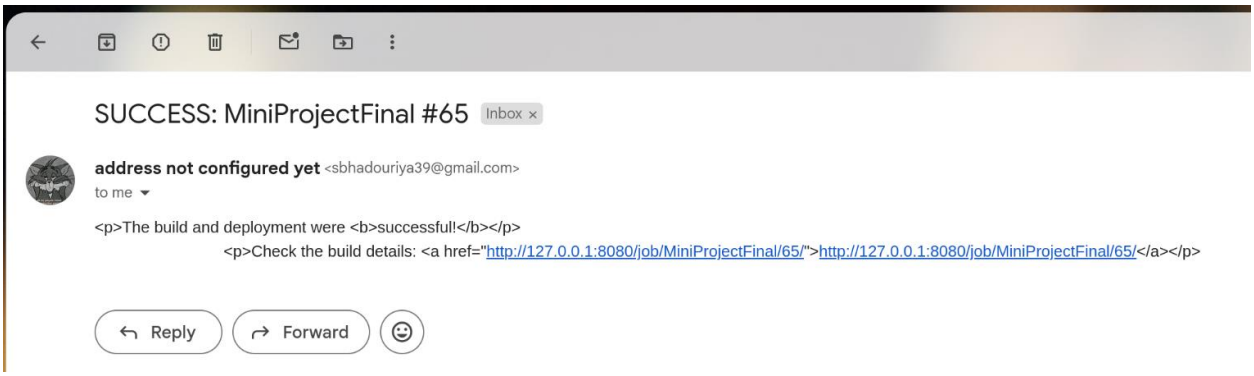
465

Reply-To Address

sbhadouriya39@gmail.com

And Add these Stages in Jenkinsfile to notify user

```
62          post {
63                 success {
64                     emailext(
65                         to: 'sbhadouriya39@gmail.com',
66                         subject: "SUCCESS: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
67                         body: """<p>The build and deployment were <b>successful!</b></p>
68                                 <p>Check the build details: <a href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>"""
69                     )
70                 }
71                 failure {
72                     emailext(
73                         to: 'sbhadouriya39@gmail.com',
74                         subject: "FAILURE: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
75                         body: """<p>The build or deployment <b>failed!</b></p>
76                                 <p>Check the build details: <a href="${env.BUILD_URL}">${env.BUILD_URL}</a></p>"""
77                     )
78                 }
79                 always {
80                     cleanWs()
81                 }
82             }
83
```



Email Notification Received

```
abhay@abhay-pc:~/Desktop/spe/My/miniProject$ docker exec -it calci java \-cp \scientific_calculator.jar \ScientificCalculator

Scientific Calculator
1. Square root (?x)
2. Factorial (x!)
3. Natural Logarithm (ln(x))
4. Power function (x^y)
5. Exit
Choose an option (1-5): 2
Enter an integer: 5
5! = 120

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

Container deployed

Source Code

Abhaybhadouriya v2

Code | Blame | 67 lines (56 loc) · 2.41 KB

```java
import java.util.Scanner;

public class ScientificCalculator {

    public static long factorial(int n) {
        if (n < 0) throw new IllegalArgumentException("Factorial is not defined for negative numbers.");
        long fact = 1;
        for (int i = 1; i <= n; i++) fact *= i;
        return fact;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\nScientific Calculator");
            System.out.println("1. Square root (√x)");
            System.out.println("2. Factorial (x!)");
            System.out.println("3. Natural Logarithm (ln(x))");
            System.out.println("4. Power function (x^y)");
            System.out.println("5. Exit");
            System.out.print("Choose an option (1-5): ");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter a number: ");
                    double num1 = scanner.nextDouble();
                    System.out.println("√" + num1 + " = " + Math.sqrt(num1));
                    break;

                case 2:
                    System.out.print("Enter an integer: ");
                    int num2 = scanner.nextInt();
                    System.out.println(num2 + "! = " + factorial(num2));
                    break;
```

```
                case 3:
                    System.out.print("Enter a number: ");
                    double num3 = scanner.nextDouble();
                    if (num3 <= 0) {
                        System.out.println("ln(x) is only defined for x > 0.");
                    } else {
                        System.out.println("ln(" + num3 + ") = " + Math.log(num3));
                    }
                    break;

                case 4:
                    System.out.print("Enter base (x): ");
                    double base = scanner.nextDouble();
                    System.out.print("Enter exponent (y): ");
                    double exp = scanner.nextDouble();
                    System.out.println(base + "^" + exp + " = " + Math.pow(base, exp));
                    break;

                case 5:
                    System.out.println("Exiting... Thank you!");
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid choice! Please select a valid option.");
            }
        }
    }
}
```

Project GitHub Link: https://github.com/Abhaybhadouriya/miniProject_SPE/

Docker Hub Image Link:
https://hub.docker.com/repository/docker/abhay003/calculator/general

Conclusion

In this project, I automated the entire Software Development Life Cycle (SDLC) using a DevOps toolchain. This streamlines the workflow for both development and operations teams by enabling seamless code changes while minimizing the risk of errors in production. The toolchain facilitates rapid building, testing, and deployment of new software versions, enhancing efficiency and reliability.

By integrating various DevOps tools, I established a continuous integration and continuous deployment (CI/CD) pipeline that ensures faster delivery of features with improved code quality. Automated testing at each stage helps detect and resolve issues early, reducing downtime and deployment failures. Additionally, monitoring and logging mechanisms provide real-time insights into system performance, enabling proactive issue resolution.

This automation not only accelerates the software development process but also enhances collaboration between teams, ensuring smoother and more reliable releases.

Reference

https://git-scm.com/doc

https://maven.apache.org/guides/index.html

https://junit.org/junit5/docs/current/user-guide/

https://www.jenkins.io/doc/

https://docs.docker.com/

https://docs.ansible.com/ansible/latest/getting_started/index.html