CS816 - Software Production Engineering

Scientific Calculator
MINI PROJECT - Report

Name: N V Sai Likhith

Adm No: IMT2020118

Introduction

This document reports the Scientific Calculator Project which was developed using DevOps Engineering Methodologies. The project includes source control management, CI/CD pipeline and Automatic Build feature.

Link to GitHub

GitHub Repo: https://github.com/Likhith-2914/Scientific-Calculator

Docker Hub: https://hub.docker.com/u/nvsailikhith

Project Details

As part of the project, I developed a low-level Scientific Calculator that carries out basic arithmetic operations. All the operations are carried out in the command line interface.

Tools Used

As mentioned this project has leveraged the principles of DevOps Engineering like SCM, CI/CD to achieve the final results.

Application Development:

To develop the source code of the application **Java Programming** Paradigm was used.

Testing

The application is tested using **Junit.**

Source Code Management:

SCM keeps track of the previous versions/commits of the program and enables us to rollback to any stable version in case of mishap. Added advantage is it helps in compiling the code developed by multiple developers. For this particular project, I have used **Git** as my tool for SCM.

CI/CD:

Once a stable version of any part of the code is committed to the SCM(GitHub), the application needs to be re-built with the anticipated changes. An automated version of this task is mandatory as manual build cannot be done after every update to the code.

Jenkins is a tool that helps in continuous and automatic building of the commits made to the source code. I used the same.

Containerization

Our application is meant to work in any environment. For that, we containerize our application into **Docker** images and push into **Docker Hub.** We can use these images in any system to run our application.

Deployment

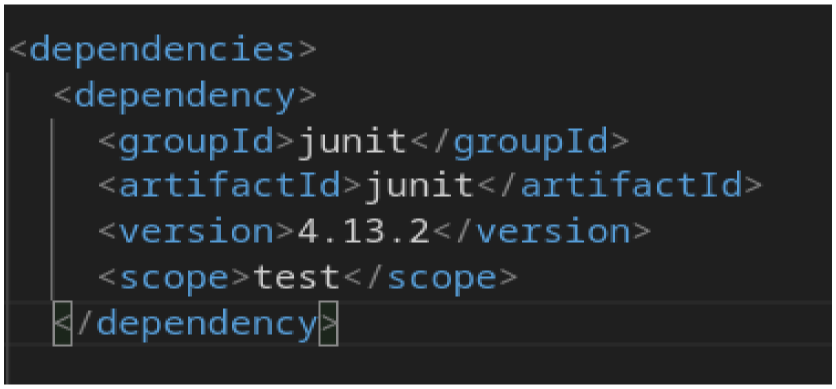
This is the final stage of the project where we deploy our application using **Ansible.** Ansible pulls the docker image and deploys our application following the command specified in the Dockerfile.

Detailed Overview

In this section, I will present a detailed overview of the steps involved in achieving the final application.

Source Code

I started by creating a bare minimum java project that takes an integer input and prints it to the console output. I have used the VS Code IDE to create the Java project. As I move forward in developing the application, there are a set of dependencies that are required to run the code. These are included in the pom.xml file.


```xml
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
```

Link with GitHub

After creating the java project, I have brought the SCM into the scene. I added all my files into git and made my initial commit. Then I pushed the code into my GitHub Repository to keep track of the commits. The commands for the same are:

Add to the git: `git add <filename>`

Commit the changes: `git commit -m ‘Initial Commit’`

Link with the GitHub Repo: `git remote add url <link>`

Push the changes/code: `git push -u origin main`

Connecting with Jenkins

After connecting with the SCM, Jenkins is used for the continuous integration and build. I have created a pipeline project in Jenkins and written a pipeline script that automates the app deployment at various stages. Each stage will be explained in the following section.

Before that, I connected GitHub with Jenkins using **ngrok** and **Github Webhook.**

ngrok is a cross-platform that allows us to expose our local server to the internet by hosting it on their sub-domain. We invoke ngrok by the command `ngrok http 8080` where 8080 specifies the localhost port number. This will generate a link that can be used to generate a **webhook** of our github account. This hook triggers the Jenkins server whenever there is a commit and eventually the build happens without human intervention.

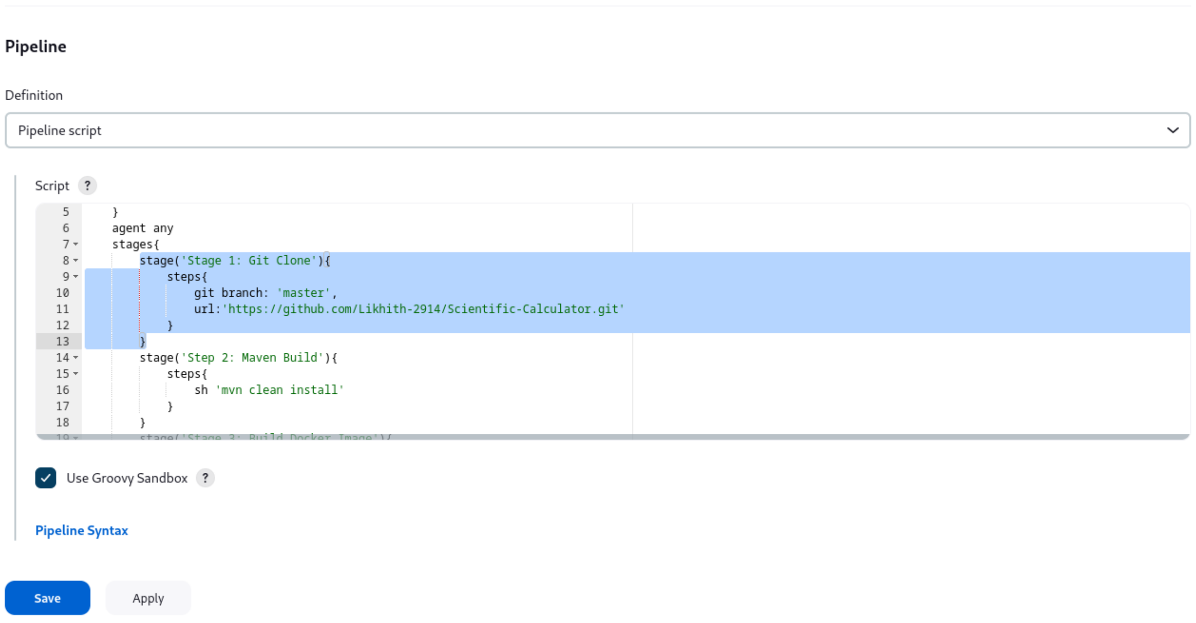
At this stage, the github and Jenkins are connected in a way that immediately after a commit made to the code, the build happens automatically.

Pipeline Stages

In this section, I will explain the activities performed at each stage of the Jenkins that builds and deploys our project.

Stage I: Git Clone

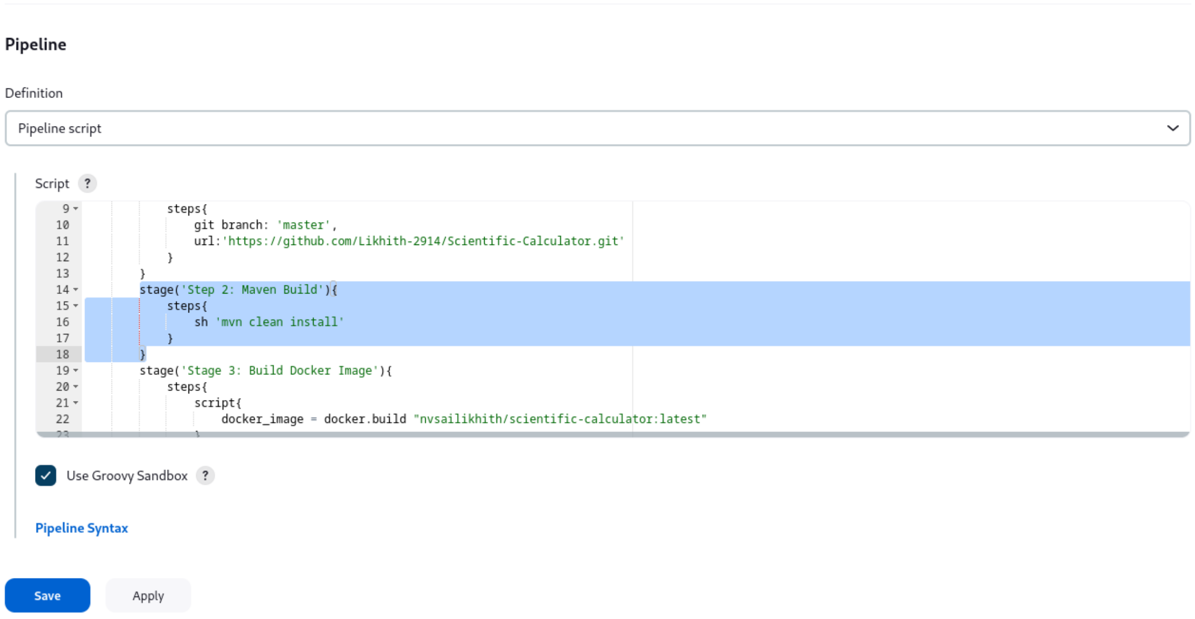
In this stage, I cloned my github repository which means I have copied all my folders to the jenkins workspace (`/var/lib/jenkins/workspace/`).



```
    }
    agent any
    stages{
        stage('Stage 1: Git Clone'){
            steps{
                git branch: 'master',
                url:'https://github.com/Likhith-2914/Scientific-Calculator.git'
            }
        }
        stage('Step 2: Maven Build'){
            steps{
                sh 'mvn clean install'
            }
        }
        stage('Stage 3: Build Docker Image'){
```

Stage II: Maven Build

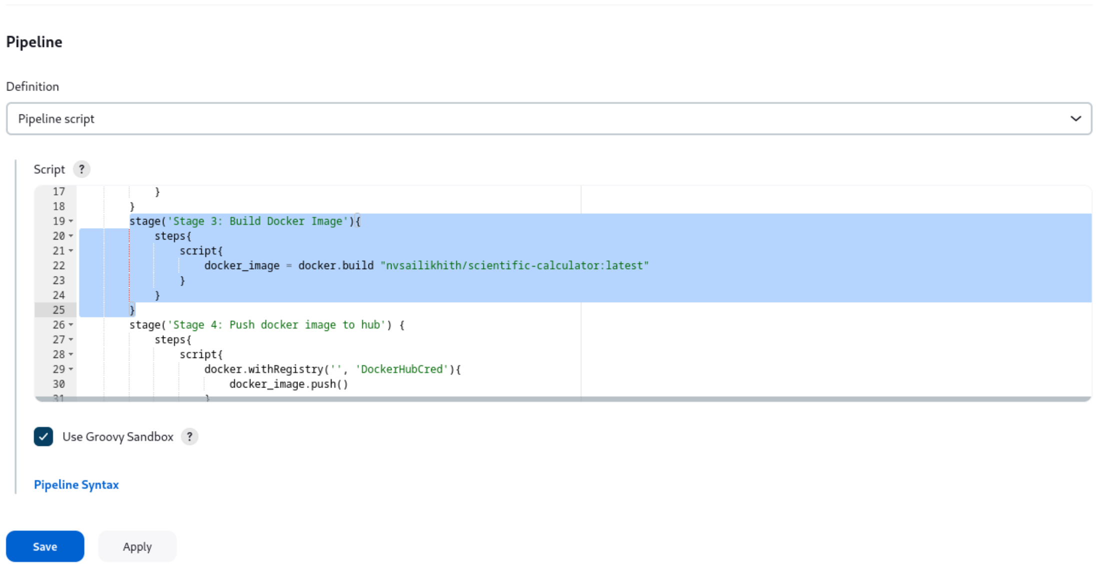
Maven is a Java tool that helps in building a Java project. It creates a jar file for all the dependencies required for the running of the Java program from the `pom.xml` file. This helps us in a flexible way of dealing with dependencies as we can add a dependency whenever we feel necessary and need not worry about the compilation of code.



```
            steps{
                git branch: 'master',
                url:'https://github.com/Likhith-2914/Scientific-Calculator.git'
            }
        }
        stage('Step 2: Maven Build'){
            steps{
                sh 'mvn clean install'
            }
        }
        stage('Stage 3: Build Docker Image'){
            steps{
                script{
                    docker_image = docker.build "nvsailikhith/scientific-calculator:latest"
```

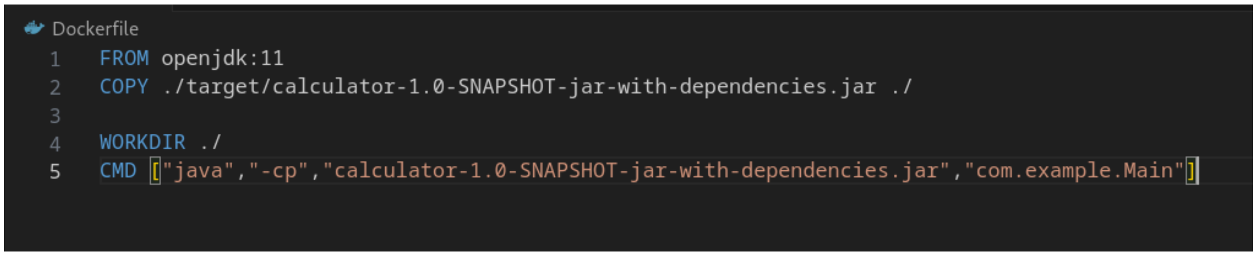
Stage III: Build Docker Image

At this stage, we build a containerised Docker Image that makes our application executable in any environment irrespective of the Operating System and other hardware/software criterion.



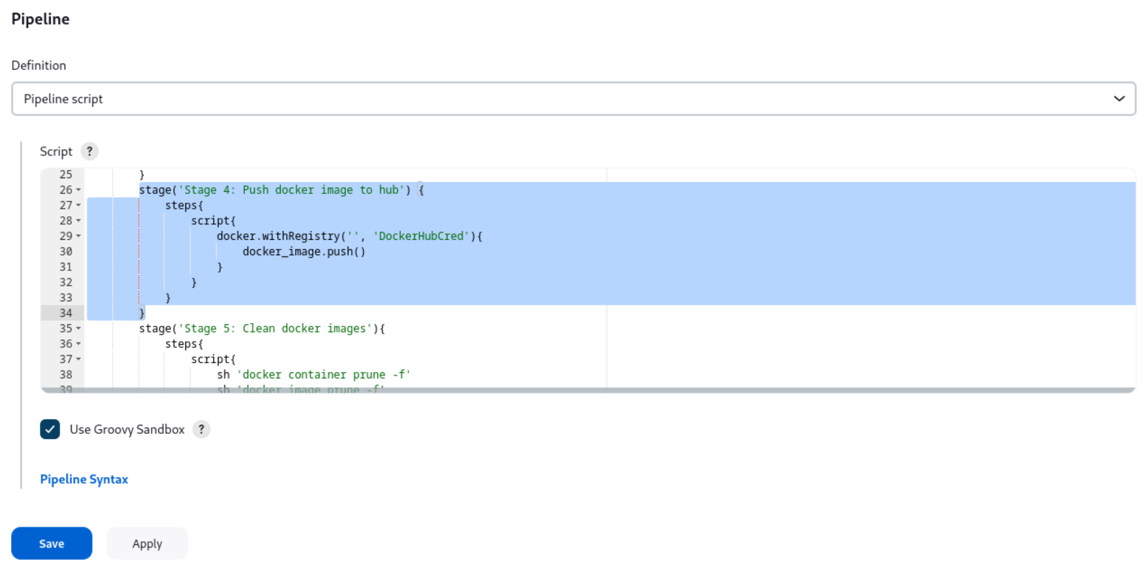
```
            }
        }
        stage('Stage 3: Build Docker Image'){
            steps{
                script{
                    docker_image = docker.build "nvsailikhith/scientific-calculator:latest"
                }
            }
        }
        stage('Stage 4: Push docker image to hub') {
            steps{
                script{
                    docker.withRegistry('', 'DockerHubCred'){
                        docker_image.push()
```

This image, along with the source code and dependencies, contains the commands needed to be executed to run our code for the deployment specified in the **Dockerfile.**


Dockerfile

```
FROM openjdk:11
COPY ./target/calculator-1.0-SNAPSHOT-jar-with-dependencies.jar ./

WORKDIR ./
CMD ["java","-cp","calculator-1.0-SNAPSHOT-jar-with-dependencies.jar","com.example.Main"]
```

Stage IV: Push the Docker Image


Pipeline

Definition

Pipeline script

Script

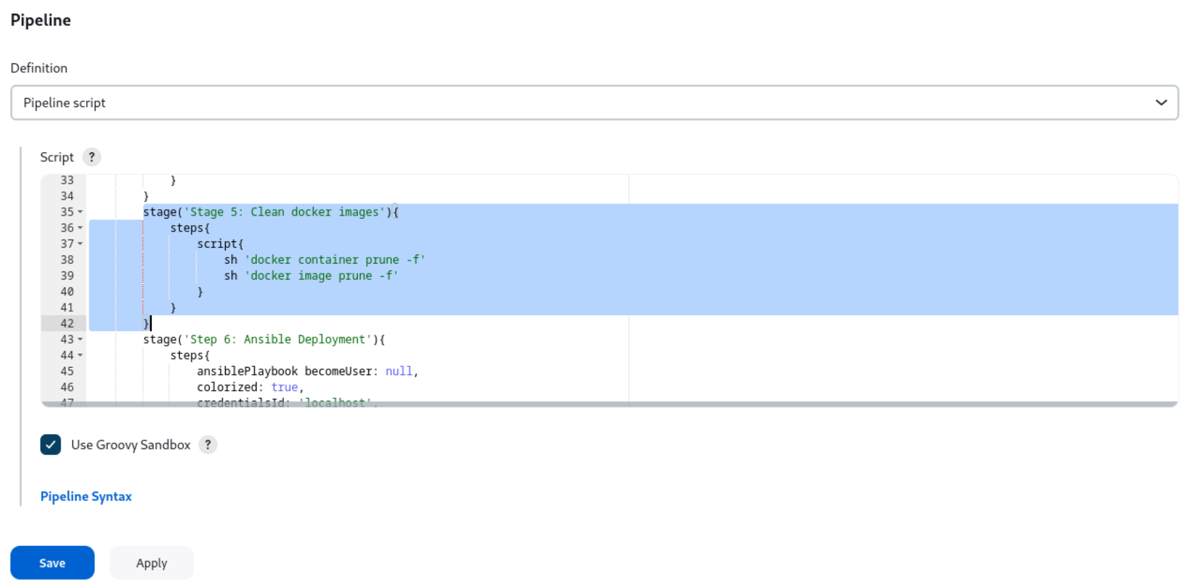
```
    }
    stage('Stage 4: Push docker image to hub') {
        steps{
            script{
                docker.withRegistry('', 'DockerHubCred'){
                    docker_image.push()
                }
            }
        }
    }
    stage('Stage 5: Clean docker images'){
        steps{
            script{
                sh 'docker container prune -f'
                sh 'docker image prune -f'
```

[✓] Use Groovy Sandbox

At this stage, we push our docker image into Docker Hub which can later be pulled to deploy the application.

Stage V: Clean Docker Images

At this stage, the docker image will be cleared locally as it has been pushed into the Docker Hub to reduce the redundancy.


Pipeline

Definition

Pipeline script

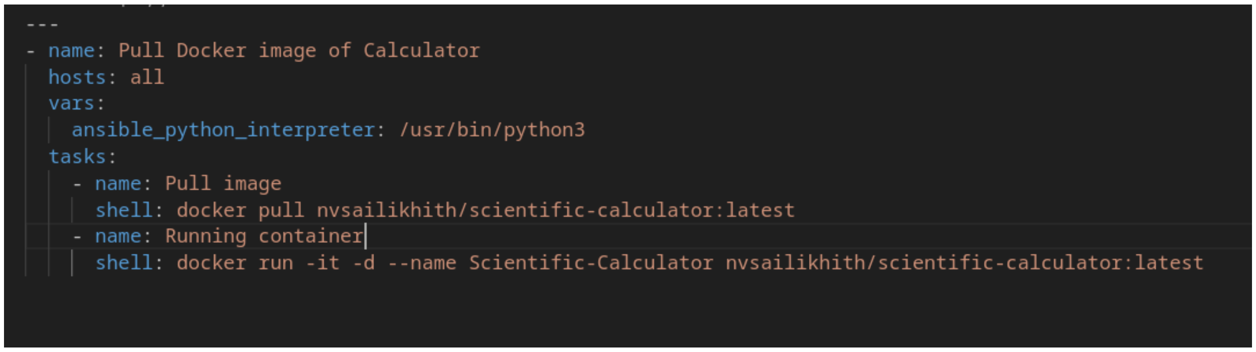
Script

```
            }
        }
        stage('Stage 5: Clean docker images'){
            steps{
                script{
                    sh 'docker container prune -f'
                    sh 'docker image prune -f'
                }
            }
        }
        stage('Step 6: Ansible Deployment'){
            steps{
                ansiblePlaybook becomeUser: null,
                colorized: true,
                credentialsId: 'localhost',
```

☑ Use Groovy Sandbox

Stage VI: Ansible Deployment

This is the final stage where we deploy our application after the successful passage from all the stages. Ansible performs the actions specified in a `yaml` file `deploy.yaml`.

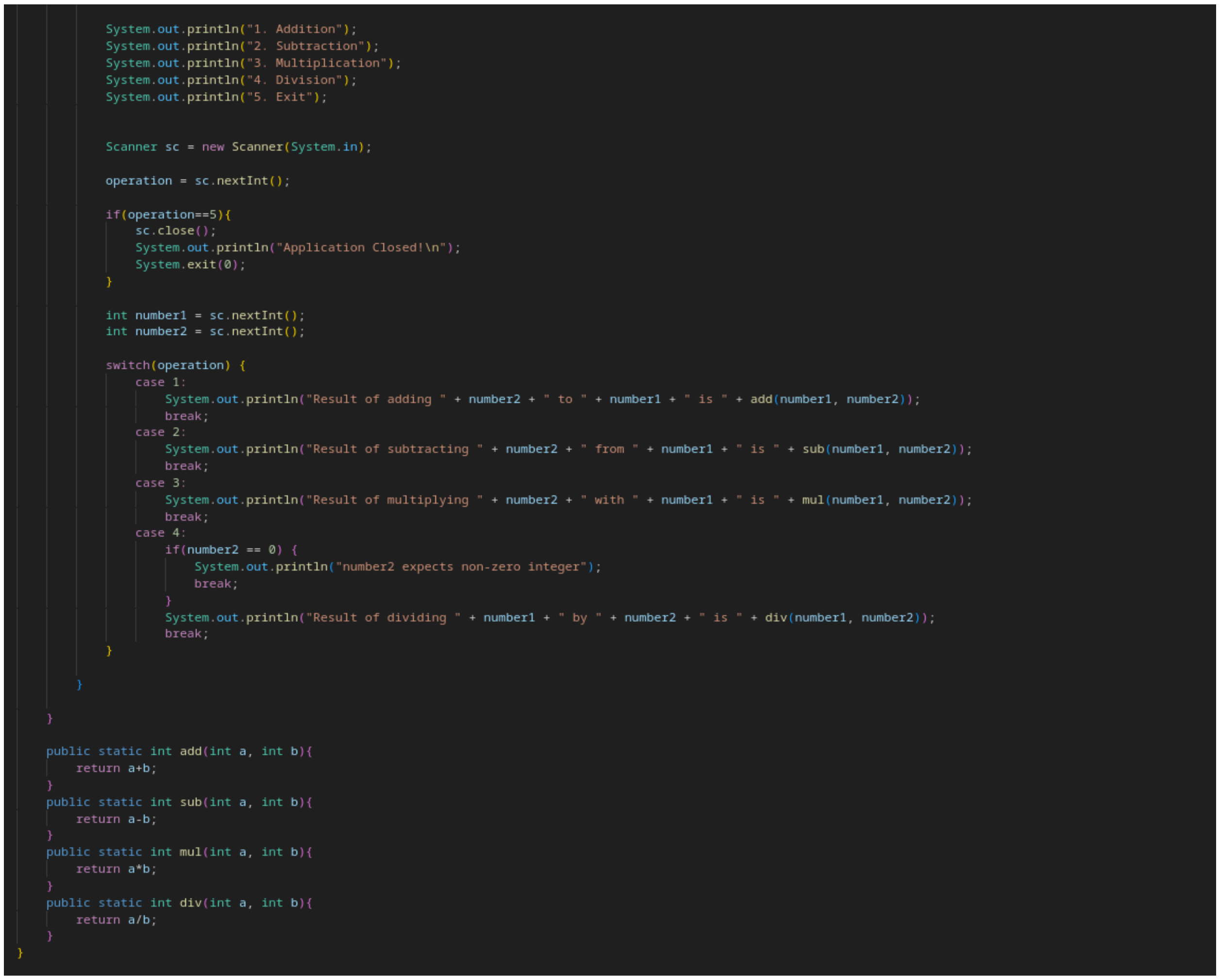

```yaml
---
- name: Pull Docker image of Calculator
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3
  tasks:
    - name: Pull image
      shell: docker pull nvsailikhith/scientific-calculator:latest
    - name: Running container
      shell: docker run -it -d --name Scientific-Calculator nvsailikhith/scientific-calculator:latest
```

With reference to the above screenshot, two tasks are performed by the ansible. Firstly, it pulls the docker image container from the docker hub. Then it runs the container by giving it a specified name and id that can be used for later purposes (like manual stop).

After successful deployment, the container keeps running until we exit from the application manually.

Code for Calculator

Once the setup was done, I started working on developing the calculator. I have implemented basic arithmetic operations of Addition, Subtraction, Multiplication and Division.

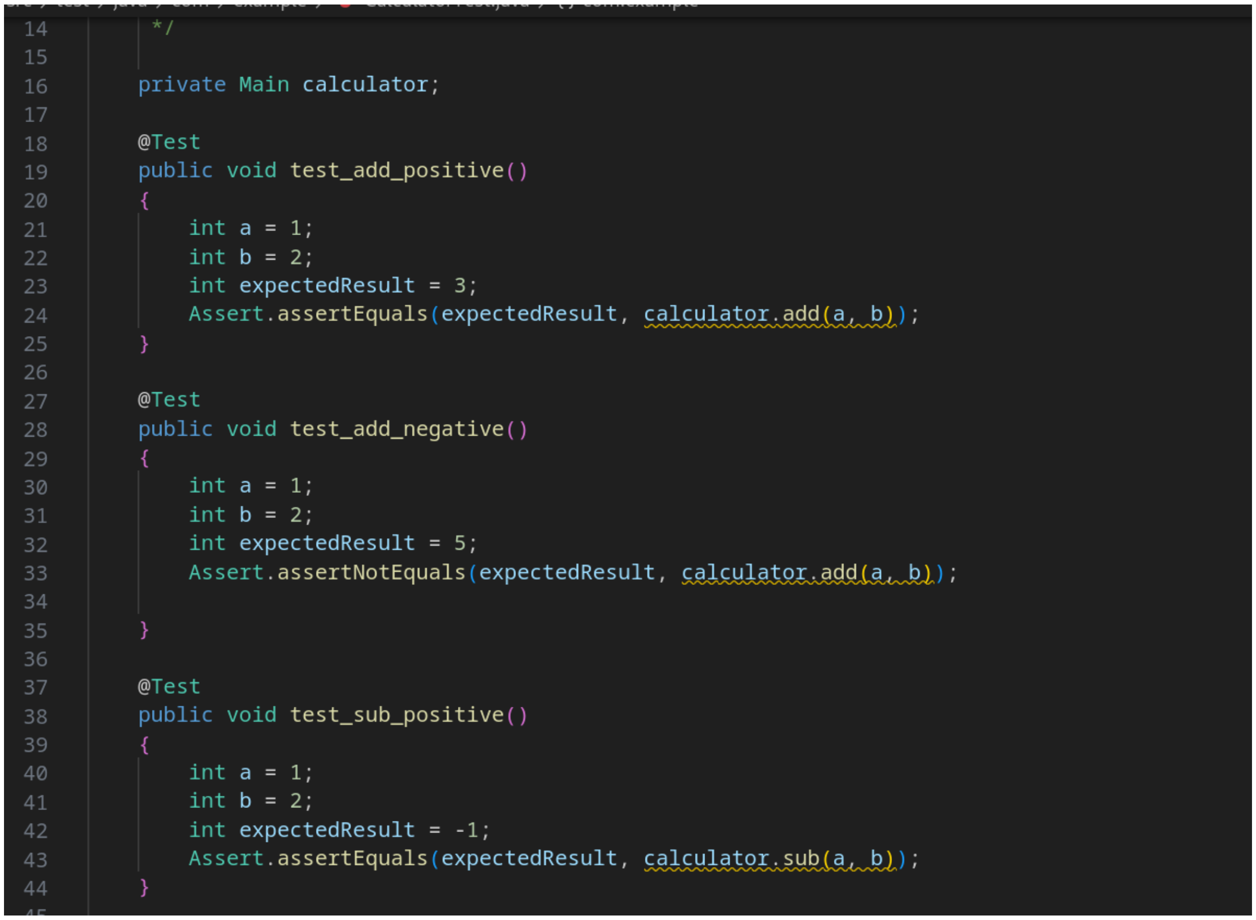


```java
            System.out.println("1. Addition");
            System.out.println("2. Subtraction");
            System.out.println("3. Multiplication");
            System.out.println("4. Division");
            System.out.println("5. Exit");


            Scanner sc = new Scanner(System.in);

            operation = sc.nextInt();

            if(operation==5){
                sc.close();
                System.out.println("Application Closed!\n");
                System.exit(0);
            }

            int number1 = sc.nextInt();
            int number2 = sc.nextInt();

            switch(operation) {
                case 1:
                    System.out.println("Result of adding " + number2 + " to " + number1 + " is " + add(number1, number2));
                    break;
                case 2:
                    System.out.println("Result of subtracting " + number2 + " from " + number1 + " is " + sub(number1, number2));
                    break;
                case 3:
                    System.out.println("Result of multiplying " + number2 + " with " + number1 + " is " + mul(number1, number2));
                    break;
                case 4:
                    if(number2 == 0) {
                        System.out.println("number2 expects non-zero integer");
                        break;
                    }
                    System.out.println("Result of dividing " + number1 + " by " + number2 + " is " + div(number1, number2));
                    break;
            }

        }

    }

    public static int add(int a, int b){
        return a+b;
    }
    public static int sub(int a, int b){
        return a-b;
    }
    public static int mul(int a, int b){
        return a*b;
    }
    public static int div(int a, int b){
        return a/b;
    }
}
```

The code is very naive as the aim of the project is getting used to the DevOps Methodologies.

Testing

Once the code is ready and working, I have implemented Testing using Junit to ensure for a well-working of code when further changes are made.



```
     */

    private Main calculator;

    @Test
    public void test_add_positive()
    {
        int a = 1;
        int b = 2;
        int expectedResult = 3;
        Assert.assertEquals(expectedResult, calculator.add(a, b));
    }

    @Test
    public void test_add_negative()
    {
        int a = 1;
        int b = 2;
        int expectedResult = 5;
        Assert.assertNotEquals(expectedResult, calculator.add(a, b));

    }

    @Test
    public void test_sub_positive()
    {
        int a = 1;
        int b = 2;
        int expectedResult = -1;
        Assert.assertEquals(expectedResult, calculator.sub(a, b));
    }
```

Avoiding the screenshot of total code due to space constraint.

Running of the Application

The screenshots of the application after making a commit has been provided at OneDrive . The images are 1-indexed and their title explains their role. Note the timestamp at the commit stage and build stage. We can observe that the container is running as long as we access the application and closes once we exit.

Conclusion

From the run of the application, it can be observed that the complete project has been automated once a safe commit has been done and also that we are able to run the docker image in a completely new folder where no dependencies are present. So, it's automated and independent!