SPE Mini-project Report

Prem Shah - IMT2020044

Introduction to the project: Calculator

We will develop a calculator program with simple functionalities like addition, subtraction, etc. The main task of this project is to use the DevOps workflow for developing the program using different tools and operations using containerization. We will apply DevOps practices such as version control, Continuous integration, Automated testing and Infrastructure as code. By these we can create a well-structured, efficient and maintainable development pipeline.

Functionalities of the Calculator

- Addition of two numbers
- Subtraction of two numbers
- Multiplication of two numbers
- Division of two numbers
- Logarithm of base 10
- Power of exponent (e)

Tools Used

Git

Git is a distributed version control system. We have used Git to create repository locally. Then we connect it to GitHub as our remote repository. Git helps to manage and track changes in the remote repository. We can add and commit changes made in the local repository to the remote repository using the command line easily with git.

Maven

Maven is a popular build and dependency management tool. We have used Maven to automate the build process. This means automatically compiling source code, running tests and it also resolves and downloads required dependencies from repositories which ensures consistent builds across different environments.

Jenkins

Jenkins is a popular open-source automation server. We will create a pipeline and use Jenkins for automatically building and testing code changes as soon as they are pushed into the version control system, which is called Continuous Integration. The Jenkins pipeline will pull the project from the Docker container and run the project.

Docker

Docker is a containerization platform. We used Docker Hub to remotely store our containers. We will package the program source code and its dependencies into a single, lightweight container image. Docker image will be integrated with the CI pipeline to be used during build and test.

Ansible

Ansible is a powerful automation and configuration management tool used to streamline and automate various tasks related to infrastructure provisioning, configuration management, application deployment.

Ngrok

Ngrok is a tunnelling and reverse proxy service that allows you to expose a local web server or service to the public internet securely. Ngrok simplifies testing of webhooks by allowing developers to receive HTTP requests from external sources, including third-party services. This ensures that webhooks function as expected before integrating them into the application.

Webhooks

Webhooks are a mechanism for real-time communication between applications and services over the internet. They allow one application to notify another application or service about specific events or data changes as they happen. We have used webhooks in the version control system GitHub. When code repositories are updated (e.g., code commits or pull requests), webhooks trigger automatic builds, tests, and deployment processes in Jenkins pipelines.

Steps Involved

1. Create a Maven project locally

Create a new project in any IDE with maven build and Java language. After installing maven run the following commands:

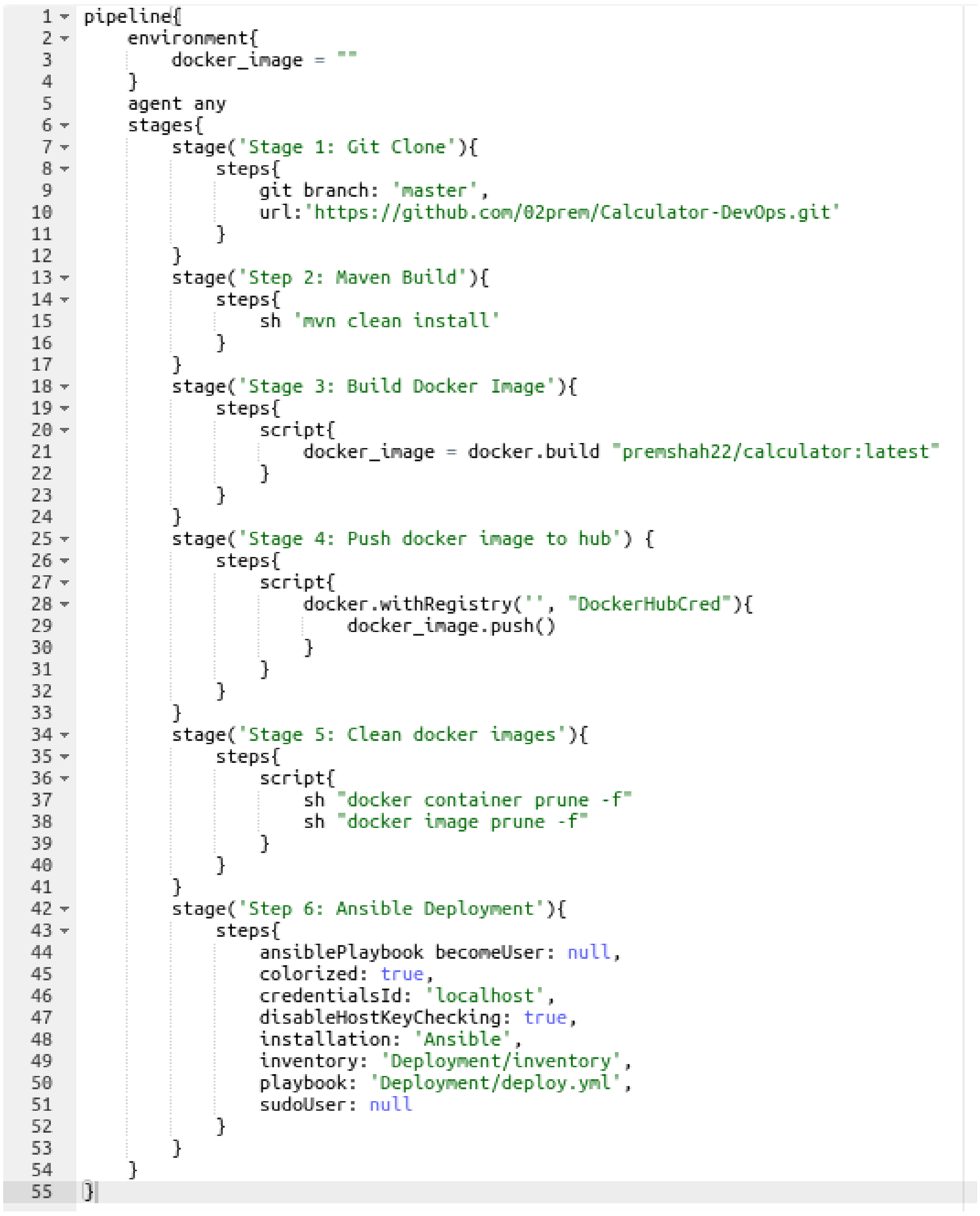
- **mvn clean**: removes the target folder and makes sure to compile the latest version of the project.
- **mvn compile:** compiles the project and checks for the errors.
- **mvn install:** it creates the JAR file.

2. Add Version Control using Git and GitHub

We will initialise git for version control and then connect it to GitHub which makes our remote version control repository. We have to create a new repository in GitHub and then add files from the local repository to the remote repository. Finally, commit the changes to the github repository.

3. Jenkins and Pipeline creation

Now we want to create a CI/CD pipeline. It will automate the build and execution by pulling the changes committed in the github repository and running the project. The steps and order of the execution is defined in the pipeline. The pipeline script should look like below:



```
pipeline{
    environment{
        docker_image = ""
    }
    agent any
    stages{
        stage('Stage 1: Git Clone'){
            steps{
                git branch: 'master',
                url:'https://github.com/02prem/Calculator-DevOps.git'
            }
        }
        stage('Step 2: Maven Build'){
            steps{
                sh 'mvn clean install'
            }
        }
        stage('Stage 3: Build Docker Image'){
            steps{
                script{
                    docker_image = docker.build "premshah22/calculator:latest"
                }
            }
        }
        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"
                }
            }
        }
        stage('Step 6: Ansible Deployment'){
            steps{
                ansiblePlaybook becomeUser: null,
                colorized: true,
                credentialsId: 'localhost',
                disableHostKeyChecking: true,
                installation: 'Ansible',
                inventory: 'Deployment/inventory',
                playbook: 'Deployment/deploy.yml',
                sudoUser: null
            }
        }
    }
}
```

The following are the first two stages of the pipeline:

- Git Clone

  The first stage of the pipeline deals with cloning the project master branch. Since the pipeline will be triggered by the commits to the repository, we are required to clone the latest changes.

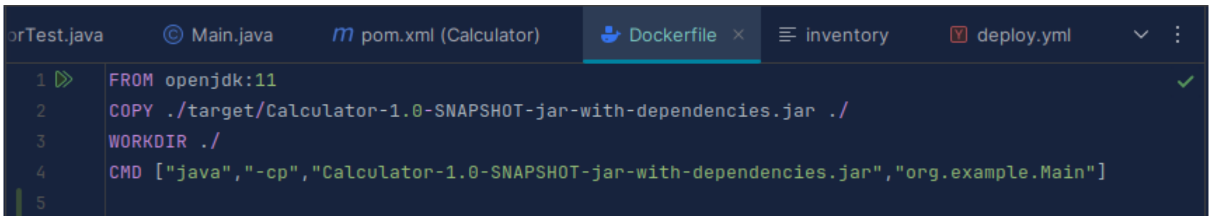
- Maven Build

  The second stage involves running the cloned repository. We will check if the project is compiling or not. The script will be as below:

```
stage('Step 2: Maven Build') {
        steps {
                sh 'mvn clean install'
        }
}
```

4. Install and Initialize Docker

Before running stage-2 we have to install and initialise docker. Once docker is installed, we will create a new file called Dockerfile in the project directory. The Dockerfile should look like below and then push it to github.



```
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","org.example.Main"]
```

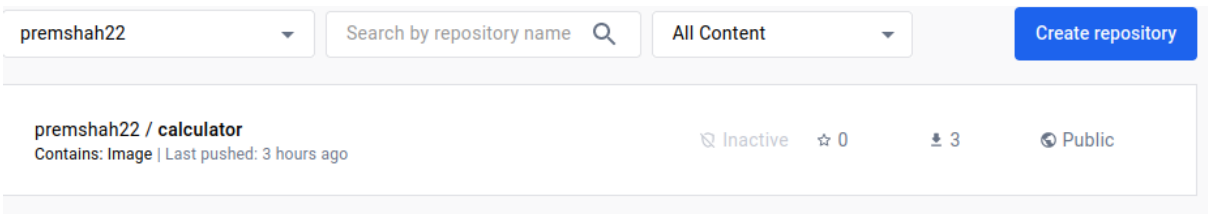
- Build Docker Image

  Building a Docker image is a crucial step in a DevOps project, as it allows you to package an application, its dependencies, and the runtime environment into a portable and reproducible container. The process begins with the creation of a Dockerfile, which is a plain text file that defines the instructions for building a Docker image. The Dockerfile typically starts with a base image (e.g., an operating system image like Ubuntu or a language-specific image like Python) and specifies the subsequent steps for configuring and installing your application and dependencies. The script for the following:

  stage('Stage 3: Build Docker Image'){

```
        steps{
                script{
                        docker_image = docker.build
"premshah22/calculator:latest"
                }
        }
}
```

- ## Push Docker image to hub



premshah22 / **calculator**
Contains: Image | Last pushed: 3 hours ago
Inactive ☆ 0 ⤓ 3 Public

Docker Hub is a service provided by Docker for finding and sharing container images. This allows us to build and generate a docker image in one system, push the image to Docker Hub and then pull that image on another system and make and execute the container. The pipeline script for the following stage:

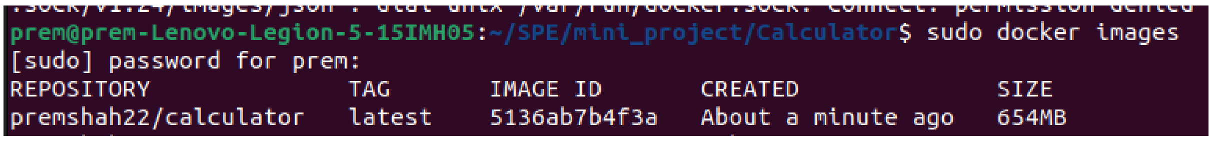
```
stage('Stage 4: Push docker image to hub') {
        steps{
                script{
                        docker.withRegistry('', "DockerHubCred"){
                        docker_image.push()
                        }
                }
        }
}
```

- ## Clean Docker images

A new docker image is created whenever we run the pipeline script instead of overwriting the existing image. So, we have to remove all images with the same name of the current project to ensure that we do not run out of resources. To do so, we have to stop the container of our project first. Then, we will remove the containers locally. The pipeline script for the following stage:

```
stage('Stage 5: Clean docker images'){
        steps{
```

```
                        script{
                                sh "docker container prune -f"
                                sh "docker image prune -f"
                        }
                }
        }
```

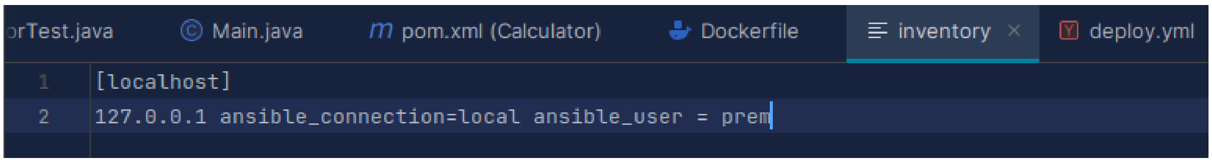


```
prem@prem-Lenovo-Legion-5-15IMH05:~/SPE/mini_project/Calculator$ sudo docker images
[sudo] password for prem:
REPOSITORY                TAG       IMAGE ID       CREATED              SIZE
premshah22/calculator     latest    5136ab7b4f3a   About a minute ago   654MB
```

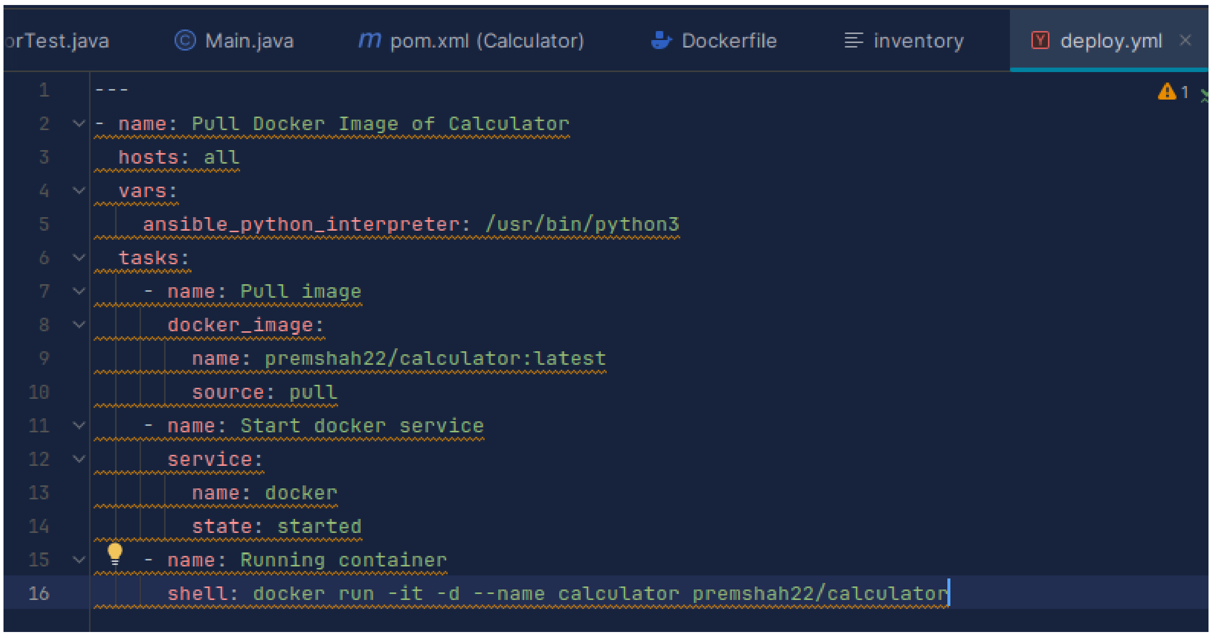
5. Pull Docker Images using Ansible

Ansible is a suite of software tools that enables infrastructure as code. We will use Ansible to pull and run docker images. After installing ansible we have to set up the host file and add the appropriate IP address.

We have to create two files: **inventory** and **deploy.yml** in a separate folder. The inventory file contains a list of clients that will deploy the project. The deploy yaml file will contain the specifications of the image that is being pulled and deployed.



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

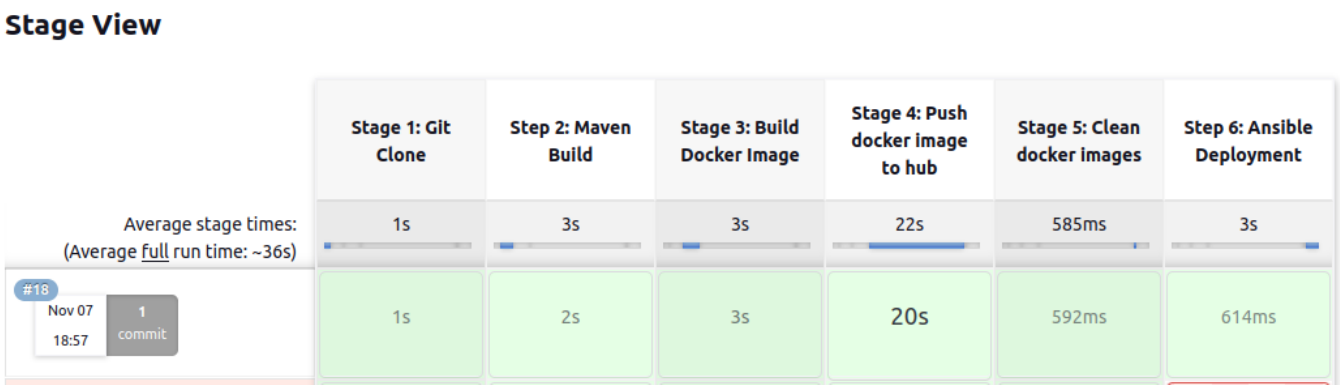


```
---
- name: Pull Docker Image of Calculator
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3
  tasks:
    - name: Pull image
      docker_image:
        name: premshah22/calculator:latest
        source: pull
    - name: Start docker service
      service:
        name: docker
        state: started
    - name: Running container
      shell: docker run -it -d --name calculator premshah22/calculator
```

The pipeline script to pull the docker image using Ansible is as follows:

```
stage('Step 6: Ansible Deployment'){
        steps{
                ansiblePlaybook becomeUser: null,
                colorized: true,
                credentialsId: 'localhost',
                disableHostKeyChecking: true,
                installation: 'Ansible',
                inventory: 'Deployment/inventory',
                playbook: 'Deployment/deploy.yml',
                sudoUser: null
        }
}
```

Pipeline of Calculator:



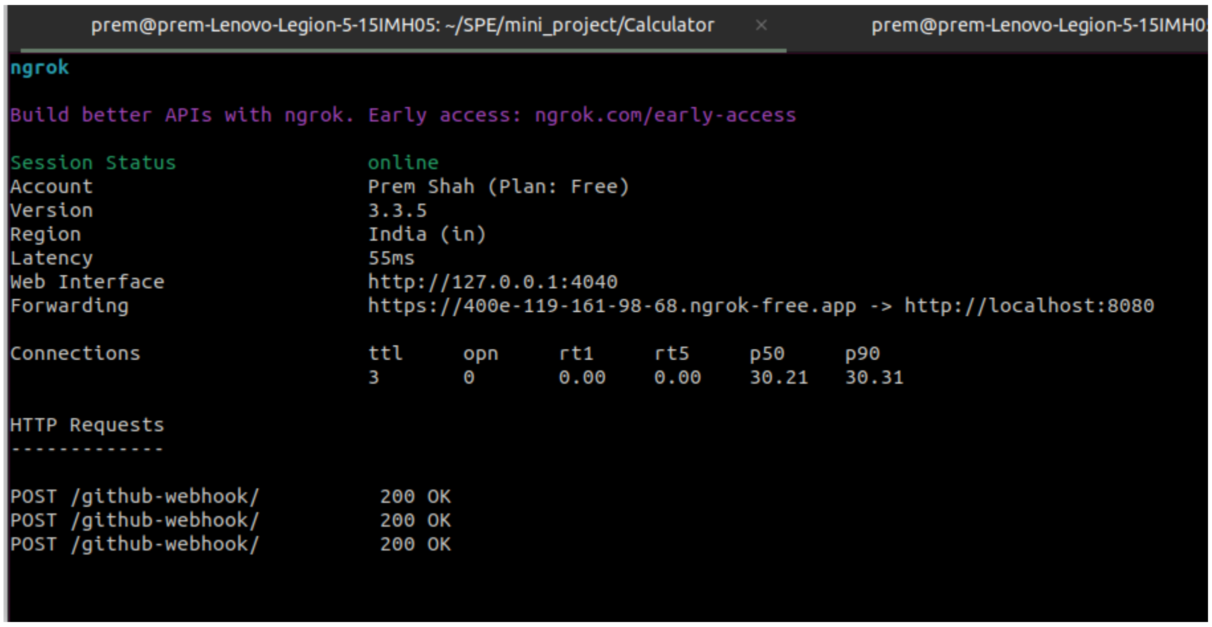
Stage View

	Stage 1: Git Clone	Step 2: Maven Build	Stage 3: Build Docker Image	Stage 4: Push docker image to hub	Stage 5: Clean docker images	Step 6: Ansible Deployment
Average stage times: (Average full run time: ~36s)	1s	3s	3s	22s	585ms	3s
#18 Nov 07 18:57 1 commit	1s	2s	3s	20s	592ms	614ms

6. GitSCM polling in Jenkins and Ngrok

Automating the execution of a pipeline using Git SCM polling and Ngrok involves setting up a Continuous Integration/Continuous Deployment (CI/CD) pipeline that automatically triggers builds and deployments whenever changes are pushed to a Git repository. Ngrok can be used to expose your local development environment to the internet for testing and integration purposes.

- First we start ngrok on local machine using the command: `ngrok http 8080`
- Ngrok provides a publicly accessible URL that forwards incoming HTTP requests to your local server. Note down the Ngrok forwarding URL.
- Add the above URL in the webhooks section of the github repository.
- Now go to Jenkins pipeline, and tick mark the GitSCM polling trigger.

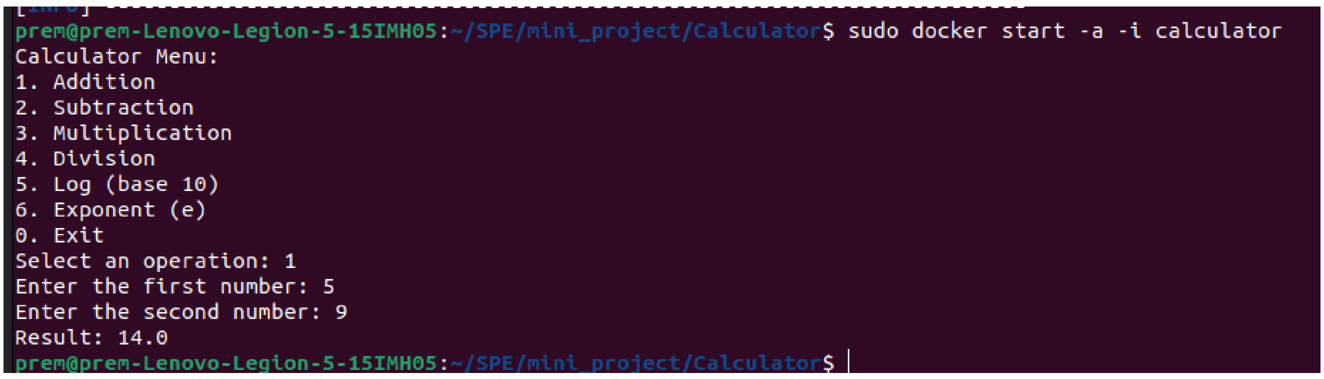


```
prem@prem-Lenovo-Legion-5-15IMH05: ~/SPE/mini_project/Calculator          prem@prem-Lenovo-Legion-5-15IMH0
ngrok

Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status                online
Account                       Prem Shah (Plan: Free)
Version                       3.3.5
Region                        India (in)
Latency                       55ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://400e-119-161-98-68.ngrok-free.app -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              3       0       0.00    0.00    30.21   30.31

HTTP Requests
-------------

POST /github-webhook/          200 OK
POST /github-webhook/          200 OK
POST /github-webhook/          200 OK
```

Now if we make any changes to the code and commit them, the moment we push the changes to the remote repository, the event gets triggered to Jenkins through webhooks. From there, Jenkins pulls the repo, builds it and runs tests using Maven and then creates a docker container and pushes to docker hub. Finally, the image is pulled, and the container is run on the machine using Ansible.

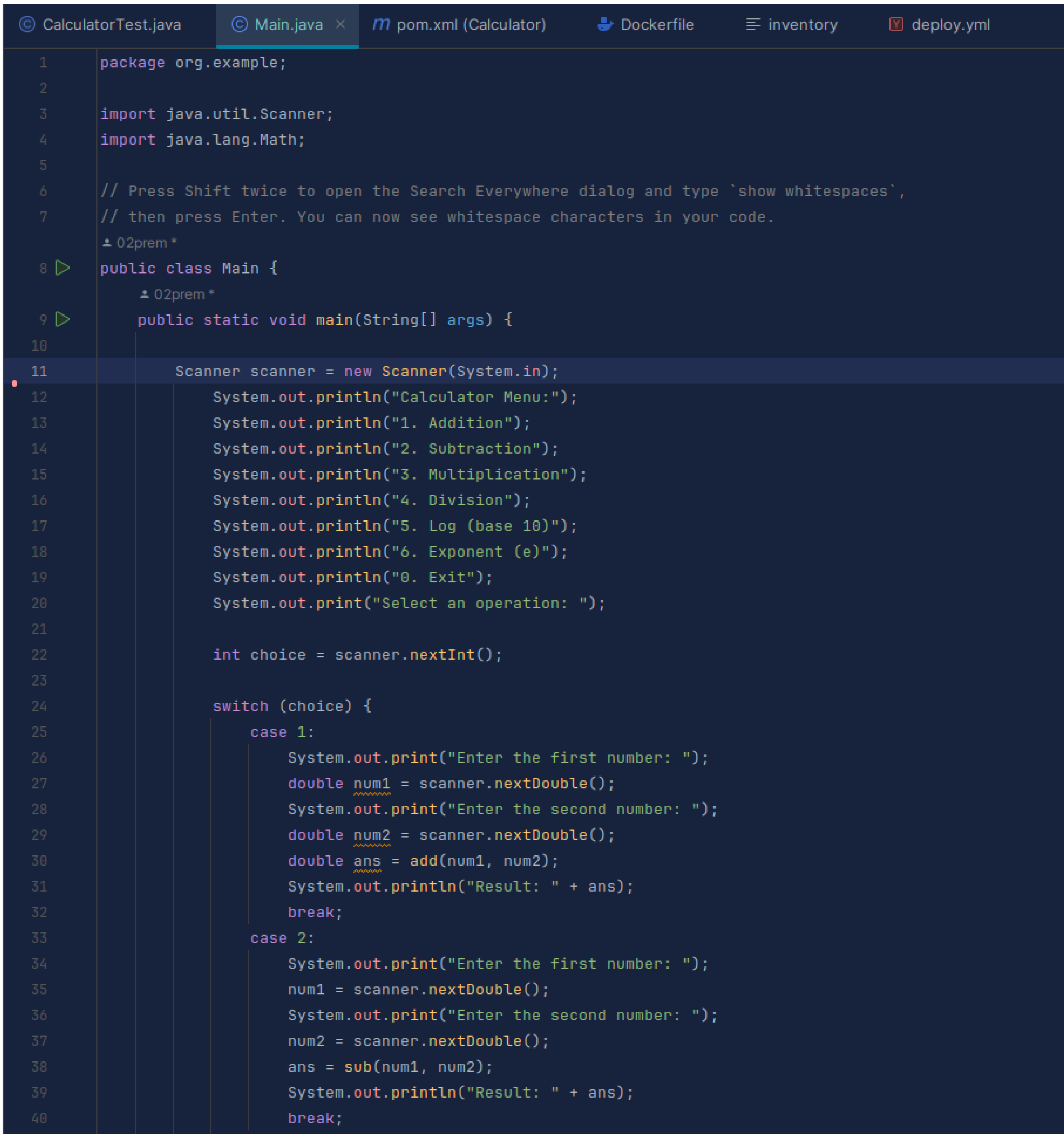
7. Running Calculator

Firstly we have to run the project. Once the project is running, we can access the container and run the program of the calculator. The container should already be running as the pipeline will automatically execute Ansible Playbook. Screenshot of the following is below.

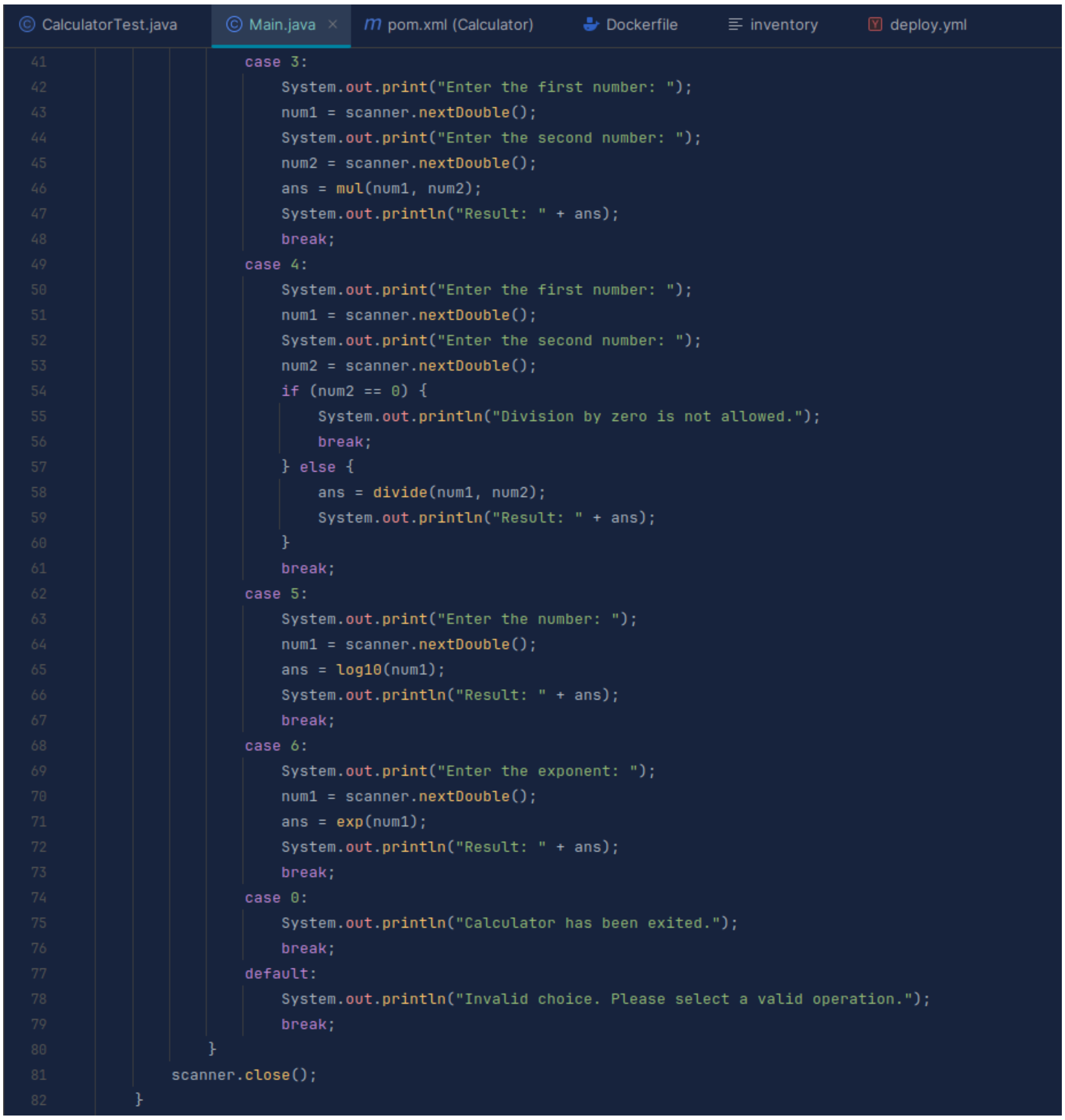


```
prem@prem-Lenovo-Legion-5-15IMH05:~/SPE/mini_project/Calculator$ sudo docker start -a -i calculator
Calculator Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Log (base 10)
6. Exponent (e)
0. Exit
Select an operation: 1
Enter the first number: 5
Enter the second number: 9
Result: 14.0
prem@prem-Lenovo-Legion-5-15IMH05:~/SPE/mini_project/Calculator$
```

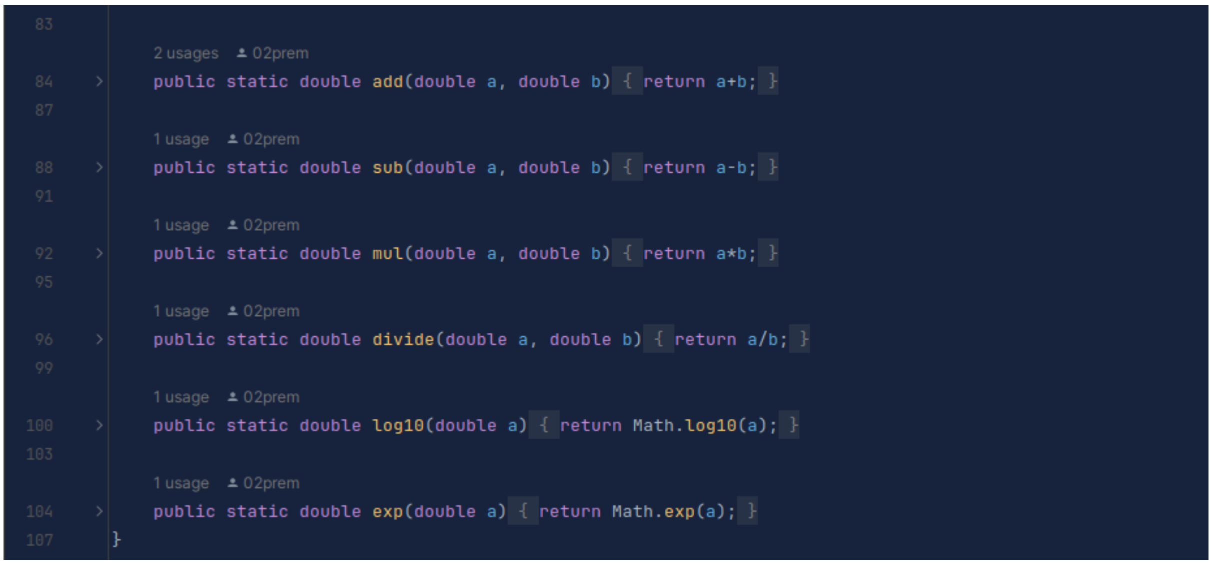
8. Calculator code



```java
package org.example;

import java.util.Scanner;
import java.lang.Math;

// Press Shift twice to open the Search Everywhere dialog and type `show whitespaces`,
// then press Enter. You can now see whitespace characters in your code.
public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
            System.out.println("Calculator Menu:");
            System.out.println("1. Addition");
            System.out.println("2. Subtraction");
            System.out.println("3. Multiplication");
            System.out.println("4. Division");
            System.out.println("5. Log (base 10)");
            System.out.println("6. Exponent (e)");
            System.out.println("0. Exit");
            System.out.print("Select an operation: ");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter the first number: ");
                    double num1 = scanner.nextDouble();
                    System.out.print("Enter the second number: ");
                    double num2 = scanner.nextDouble();
                    double ans = add(num1, num2);
                    System.out.println("Result: " + ans);
                    break;
                case 2:
                    System.out.print("Enter the first number: ");
                    num1 = scanner.nextDouble();
                    System.out.print("Enter the second number: ");
                    num2 = scanner.nextDouble();
                    ans = sub(num1, num2);
                    System.out.println("Result: " + ans);
                    break;
```



```java
                case 3:
                    System.out.print("Enter the first number: ");
                    num1 = scanner.nextDouble();
                    System.out.print("Enter the second number: ");
                    num2 = scanner.nextDouble();
                    ans = mul(num1, num2);
                    System.out.println("Result: " + ans);
                    break;
                case 4:
                    System.out.print("Enter the first number: ");
                    num1 = scanner.nextDouble();
                    System.out.print("Enter the second number: ");
                    num2 = scanner.nextDouble();
                    if (num2 == 0) {
                        System.out.println("Division by zero is not allowed.");
                        break;
                    } else {
                        ans = divide(num1, num2);
                        System.out.println("Result: " + ans);
                    }
                    break;
                case 5:
                    System.out.print("Enter the number: ");
                    num1 = scanner.nextDouble();
                    ans = log10(num1);
                    System.out.println("Result: " + ans);
                    break;
                case 6:
                    System.out.print("Enter the exponent: ");
                    num1 = scanner.nextDouble();
                    ans = exp(num1);
                    System.out.println("Result: " + ans);
                    break;
                case 0:
                    System.out.println("Calculator has been exited.");
                    break;
                default:
                    System.out.println("Invalid choice. Please select a valid operation.");
                    break;
            }
        scanner.close();
    }
```



```java

    public static double add(double a, double b) { return a+b; }

    public static double sub(double a, double b) { return a-b; }

    public static double mul(double a, double b) { return a*b; }

    public static double divide(double a, double b) { return a/b; }

    public static double log10(double a) { return Math.log10(a); }

    public static double exp(double a) { return Math.exp(a); }
}
```