CS 816: Software Production Engineering November 7, 2023

Calculator Mini Project Report

Instructor: Prof. B. Thangaraju Sarthak Harne (IMT2020032)

NOTE: This report is for the project made on Ubuntu 22.04.

Link to the repository: https://github.com/sarthakharne/Calculator

About the Project

The project's goal is to set up a DevOps automation pipeline using a calculator program as the development task. The DevOps automation pipeline consists of the following:

- Polling SCM
- Building the project
- Testing the project
- Building docker image, pushing it to DockerHub and cleaning older images
- Deployment

Using this pipeline, every time a new change is pushed to the program repository, all steps are executed automatically. These steps are very generic and are commonly used in all SDLC pipelines. The calculator program can easily be swapped later with another program that someone is working on with minor changes in the pipeline.

All modern code is managed via a source code management software repository like Git, the idea of the pipeline is to use this repository to automate all the processes till deployment. Using a small program like Calculator will help to learn the setup of this pipeline as it does not take a lot of time to execute the pipeline and debug the same.

Tools Used

The following tools are used in this mini-project:

Git/GitHub

Git and GitHub were used for SCM. The link for the public repository is Calculator Repository. Git helps us track the changes made using commits, modularise the development process using branches and also provide the option to roll back incorrect commits.

Using GitHub to store the code remotely helps us share the code so that other people can view the code and are free to contribute to the same. GitHub will also help us set up webhooks to Jenkins later which makes the automation process much more efficient. It can be installed via the following command:

```
sudo apt install git
```

Jenkins

Jenkins forms the basis of the DevOps automation pipeline. Triggers are set on Jenkins to start the pipeline which in turn triggers other tools in various stages. Jenkins is very user-friendly and has support for different

plugins for many tools. Even if a plugin is not available, the flexible pipeline scripting in Jenkins can help us execute shell scripts to operate the tools and execute the commands.

The following commands are required to install Jenkins:

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

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

sudo apt install ca-certificates

sudo apt update

sudo apt install jenkins
```

ngrok

As Jenkins is being hosted on the local machine, which does not have a public IP, the GitHub webhook will not be able to locate Jenkins and will fail to send any triggers. So, ngrok is used to securely expose the port on which Jenkins is hosted on so that it becomes visible to GitHub.

The following commands can be used to install ngrok:

```
sudo tar xvzf ~/Downloads/ngrok-v3-stable-linux-amd64.tgz -C /usr/local/bin

ngrok config add-authtoken <ngrok-auth-token>
```

Maven

Maven is used for efficient and standardised project management. It simplifies the build process by automating tasks like compilation, testing, and dependency management. Maven also enforces a consistent project structure, making it easier for team members to collaborate and understand the project's layout. Maven also has a simple trigger procedure making it easy to operate in a Jenkins pipeline.

Docker, DockerHub

Docker enables the encapsulation of software and its dependencies into lightweight, isolated containers, ensuring consistency across different environments and portability. This makes development and deployment seamless, reducing compatibility issues. Due to these factors among many others, Docker is useful in a DevOps pipeline. The following commands are used to install Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

The script can then be run

```
sh get-docker.sh
```

After the installation, jenkins and docker needed to be added in the same user group. This can be done via the following:

```
sudo usermod -aG docker jenkins

sudo systemctl restart jenkins
```

DockerHub simplifies the sharing and distribution of container images. This helps us to collaborate with others more efficiently as using just the image, someone can create a container with an identical environment. Just like GitHub, DockerHub also has features like webhooks, which make automation easier.

Ansible

Ansible is used to automate the deployment of the software. Being an automation tool, Ansible has excellent support for triggers and endpoints, like the Jenkins plugin. Ansible will be used to automate the process of pulling a docker image and starting the container. For this project, the container will be started on the local machine, but Ansible can be used to do this easily on remote machines with easy setup and triggers. The following commands can be used to install Ansible:

```
sudo apt-add-repository ppa:ansible/ansible

sudo apt update

sudo apt install ansible
```

Procedure

Setup of Tools

The following tools were set up for this project:

- Git - Installation on local machine
- GitHub - Account creation
- Jenkins - Installation and setup, along with plugins for Ansible, Docker, Docker Pipeline, Git Client, Git Plugin, GitHub and JUnit were also installed
- ngrok - Installation on the local machine, account creation and setup
- IntelliJ IDEA - Installation
- Docker - Installation on the local machine, putting it in the Jenkins user group
- DockerHub - Account creation
- Ansible - Installation on the local machine

Setting up the Project and Code

First, the project is created using IntelliJ IDEA with the Build System as Maven and JDK version 11. This gives us a project structure which is required by Maven. A pom.xml file is also created which is required by Maven.

Every time Maven compiles the project, it creates a jar file with a different name. We require a constant file name for the jar file as it makes automation easier. To do this, we can use the Maven assembly plugin to give the output jar a consistent name.

Next, we will replace the Main.java file with a Calculator.java file which contains the code for the calculator. The following functions are provided in the calculator:

- Square Root
- Factorial
- Natural Logarithm
- Power

Apart from this, an option for exiting the program is also provided. All of the functions are presented as a menu-driven program. Different operations are implemented in different functions.

Every function other than factorial uses the Java Math library. Factorial is implemented using a simple for loop. Proper conditions are put in place to prevent illegal operations and the user is informed about the same.

Setting up SCM and Remote Repository

This step can take place at any time during the development, but ideally, it should take place just after the initial setup of the project. First, git init can be used in the root directory of the project to initialised to initialise tracking of the project. Next, we can add the files, and commit the changes to the local repository.

Next, a remote repository can be initialised on GitHub. The remote origin can be added to this GitHub repository and the changes can be pushed to this repository.

Setting up the test cases using JUnit

A new test directory is created following the Maven directory structure. A CalculatorTest.java file is created which uses JUnit to create test cases for the different calculator functions. Eight cases are set up for each of the functions. The JUnit dependency should also be added to the pom.xml file so that Maven installs it while building the project.

Now, when the project needs to be built mvn clean can be used to remove the target folder, if any so that after compilation, we will have the latest build. The mvn compile command will compile the project and run the test cases made using JUnit. Lastly, mvn install will create the JAR file which is present in the target folder.

Setting up the Jenkins Pipeline

First, the Pipeline project needs to be set up. The pipeline script can be written on the Jenkins dashboard or can be added as a separate Jenkinsfile to the repository. In the later case, appropriate options to locate the pipeline script needs to be selected in the pipeline configuration.

Next, the following stages are set in the Jenkins pipeline:

- The first stage is Cloning the GitHub repository. In that, the URL and branch need to be mentioned. The following is added to the pipeline script:

```
stage('Stage 1: Git Clone'){
    steps{
        git branch: 'master',
        url: 'https://github.com/sarthakharne/Calculator.git'
    }
}
```

- After the repo is cloned, the Jenkins user will have the source code, which needs to be built. In the second stage, we trigger the Maven build of the project.
  In this, the older builds are first discarded and then the source code is rebuilt. The testing part is also performed in this stage as part of the build by Maven. The following is added to the pipeline script:

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

- In the pipeline script a global variable (docker_image) for the docker image needs to be created. In the next stage, the dockerfile is used to create a container for the project. This can be implemented in the Jenkins pipeline using the Docker extension. The variable docker_image is populated with this created image. The following is added to the pipeline script:

```
stage('Stage 3: Build Docker Image'){
    steps{
        script{
            docker_image = docker.build "sarthakharne2262/calculator:latest"
        }
    }
}
```

- In the next stage, the docker image is pushed to docker hub using the credentials set up in the Jenkins configuration. These credentials are the email/username and the password set on DockerHub. The following is added to the pipeline script:

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

- Even if we create new image and containers for each build, the older ones are not required and just take up space in the machine. Hence, in the next stage the older containers and images are pruned. The following is added to the pipeline script:

```
stage('Stage 5: Clean Docker Images'){
    steps{
        script{
            sh 'docker container prune -f'
            sh 'docker image prune -f'
        }
    }
}
```

- Usually, when the images are pushed to DockerHub, they are pulled in some other machine where the service needs to be hosted. But, for this project, the images are pulled on to the local machine. Accordingly, the Ansible inventory and playbook are created by specifying the hosts.

  The steps that are needed for setting up the container are also mentioned in the playbook. Namely, pulling the image, starting the docker service and then finally starting the container. The following is added to the pipeline script:

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

Setting up SCM Polling

This step can be done after the remote repository is added and the Jenkins pipeline is created. Usually, when Jenkins is hosted on a machine which is exposed to the public internet, Git SCM Polling can be set up easily by providing GitHub the URL to send the POST Requests to. But, in this case, one of the ports of the localmachine needs to be exposed.

This is done via ngrok. The URL generated by ngrok is set as the Jenkins URL and the same is given to the GitHub Webhook. GitHub Personal Access Token is also provided to the GitHub Webhook and the same is added to the GitHub server credentials. After this is done, the option 'GitHub hook trigger for GITScm polling' is switched on.

Screenshots

```
4 usages  sarthakharne
8   public Calculator() { logger  = LogManager.getLogger(Calculator.class); }
    1 usage  sarthakharne
11  public static void menu(){
12      System.out.println("Scientific Calculator");
13      System.out.println("*************************");
14      System.out.println("1. Square Root");
15      System.out.println("2. Factorial");
16      System.out.println("3. Natural Logarithm");
17      System.out.println("4. Power");
18      System.out.println("0. Exit");
19      System.out.print("Enter your choice: ");
20  }
21
    sarthakharne *
22  public static void main(String [] arg){
23      Calculator cal = new Calculator();
24
25      Scanner sc = new Scanner(System.in);
26
27      int choice;
28
29      do {
30          Calculator.menu();
31
32          choice = sc.nextInt();
33
34          double num1, num2, ans;
35
36          switch(choice) {
37              case 1:
```

Figure 1: Code for the Menu

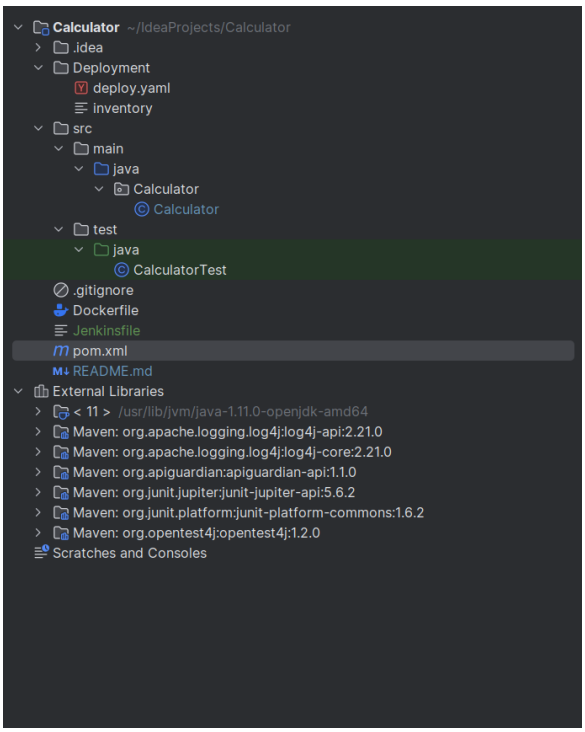
```
double num1, num2, ans;

switch(choice) {
    case 1:
        System.out.print("Enter Number: ");
        num1 = sc.nextDouble();
        ans = cal.mySqrt(num1);
        if(ans == -1) {
            System.out.println("Invalid Input");
        }
        else {
            System.out.println("Square root of " + num1 + " is " + ans);
        }
        break;

    case 2:
        System.out.print("Enter Number: ");
        num1 = sc.nextDouble();
        ans = cal.myFactorial(num1);
        if(ans == -1) {
            System.out.println("Invalid Input");
        }
        else {
            System.out.println("Factorial of " + num1 + " is " + ans);
        }
        break;
```

(a) Code for Square Root and Factorial

```
sarthakharne
@Test
public void powerTruePositive()
{
    assertEquals( expected: 4, calculator.myPow(2, 2), DELTA, message: "Power Test 1");
    assertEquals( expected: 125, calculator.myPow(5, 3), DELTA, message: "Power Test 2");
    assertEquals( expected: 3, calculator.myPow(3, 1), DELTA, message: "Power Test 3");
    assertEquals( expected: 1, calculator.myPow(10, 0), DELTA, message: "Power Test 4");
}

sarthakharne
@Test
public void powerFalsePositive()
{
    assertNotEquals( unexpected: 10, calculator.myPow(2, 2), DELTA, message: "Power Test 1");
    assertNotEquals( unexpected: 5, calculator.myPow(5, 3), DELTA, message: "Power Test 2");
    assertNotEquals( unexpected: 0, calculator.myPow(3, 1), DELTA, message: "Power Test 3");
    assertNotEquals( unexpected: 10, calculator.myPow(10, 0), DELTA, message: "Power Test 4");
}
```

(b) Test Case code for Power Function

Figure 2: Function Code and Test Cases



(a) Project Directory Structure as required by Maven

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M6</version>
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.6.2</version>
        </dependency>
    </dependencies>
</plugin>
```

(b) Maven JUnit Plugin

```
<dependencies>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.21.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.21.0</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.6.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

(c) Maven Dependencies

Figure 3: Maven Project Structure and Dependencies

```
FROM openjdk:11
ADD ./target/Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java", "-jar", "Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

Figure 4: Dockerfile

```
[host_machine]
localhost ansible_connection = jenkins
```

(a) Ansible Inventory

```
---
- name: Pull Docker Image of Calculator
  hosts: localhost
  vars:
    ansible_python_interpreter: /usr/bin/python3
  tasks:
    - name: Pull image
      docker_image:
        name: sarthakharne2262/calculator:latest
        source: pull
    - name: Start docker service
      service:
        name: docker
        state: started
    - name: Running container
      shell: docker run -it --name Calculator -d sarthakharne2262/calculator
```

(b) Ansible Playbook

Figure 5: Ansible Code

```
pipeline{
    environment{
        docker_image = ""
    }
    agent any
    stages{
        stage('Stage 1: Git Clone'){
            steps{
                git branch: 'master',
                url: 'https://github.com/sarthakharne/Calculator.git'
            }
        }
        stage('Stage 2: Maven Build'){
            steps{
                sh 'mvn clean install'
            }
        }
        stage('Stage 3: Build Docker Image'){
            steps{
                script{
                    docker_image = docker.build "sarthakharne2262/calculator:latest"
                }
            }
        }
```

(a) Stage 1-3

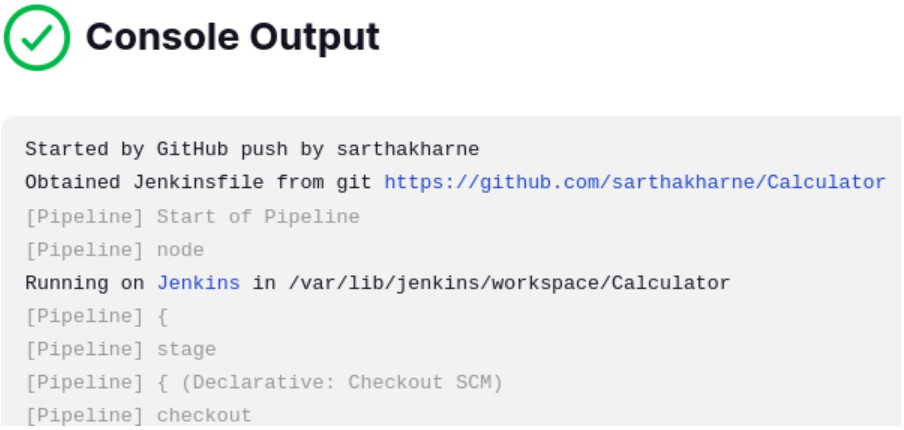
```
        stage('Stage 4: Push docker image to hub'){
            steps{
                script{
                    docker.withRegistry('', 'DockerHubCred'){
                        docker_image.push()
                    }
                }
            }
        }
        stage('Stage 5: Clean Docker Images'){
            steps{
                script{
                    sh 'docker container prune -f'
                    sh 'docker image prune -f'
                }
            }
        }
        stage('Stage 6: Ansible Deployment'){
            steps{
                ansiblePlaybook becomeUser: null,
                colorized: true,
                credentialsId: 'localhost',
                disableHostKeyChecking: true,
                installation: 'Ansible',
                inventory: 'Deployment/inventory',
                playbook: 'Deployment/deploy.yaml',
                sudoUser: null
            }
        }
    }
}
```

(b) Stage 4-6

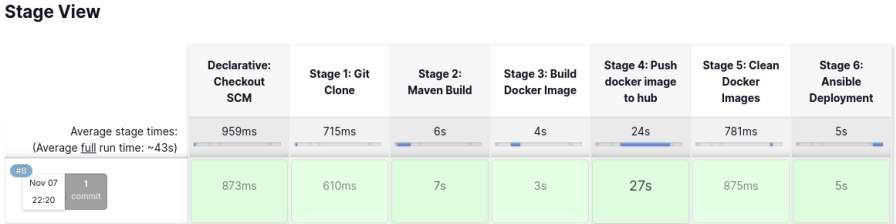
Figure 6: Pipeline Script

```
sarthak@sarthak-Inspiron-5502:~$ sudo docker start -ai Calculator
Scientific Calculator
**************************
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
0. Exit
Enter your choice: 4
Enter First Number: 5
Enter Second Number: 4
Power of 5.0 to 4.0 is 625.0

Scientific Calculator
**************************
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
0. Exit
Enter your choice: 1
Enter Number: -5
17:04:36.391 [main] ERROR Calculator.Calculator - [SQUARE_ROOT] - -5.0 - [RESULT] - null
Invalid Input

Scientific Calculator
**************************
1. Square Root
2. Factorial
3. Natural Logarithm
4. Power
0. Exit
Enter your choice: 0
Exiting...
```

Figure 7: Running the Program in the Container



(a) Console Output for GitHub trigger



(b) Pipeline Status

Figure 8: GitHub Trigger and Pipeline Execution