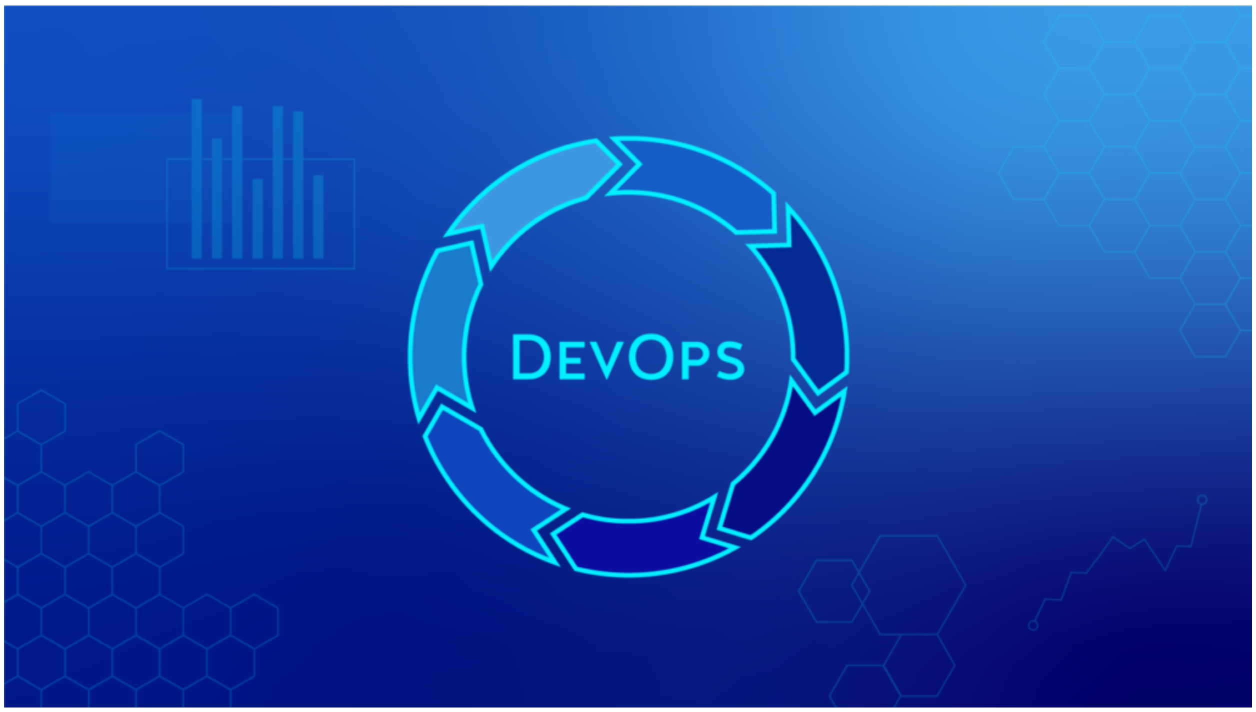


SPE MiniProject - Scientific Calculator using DevOps

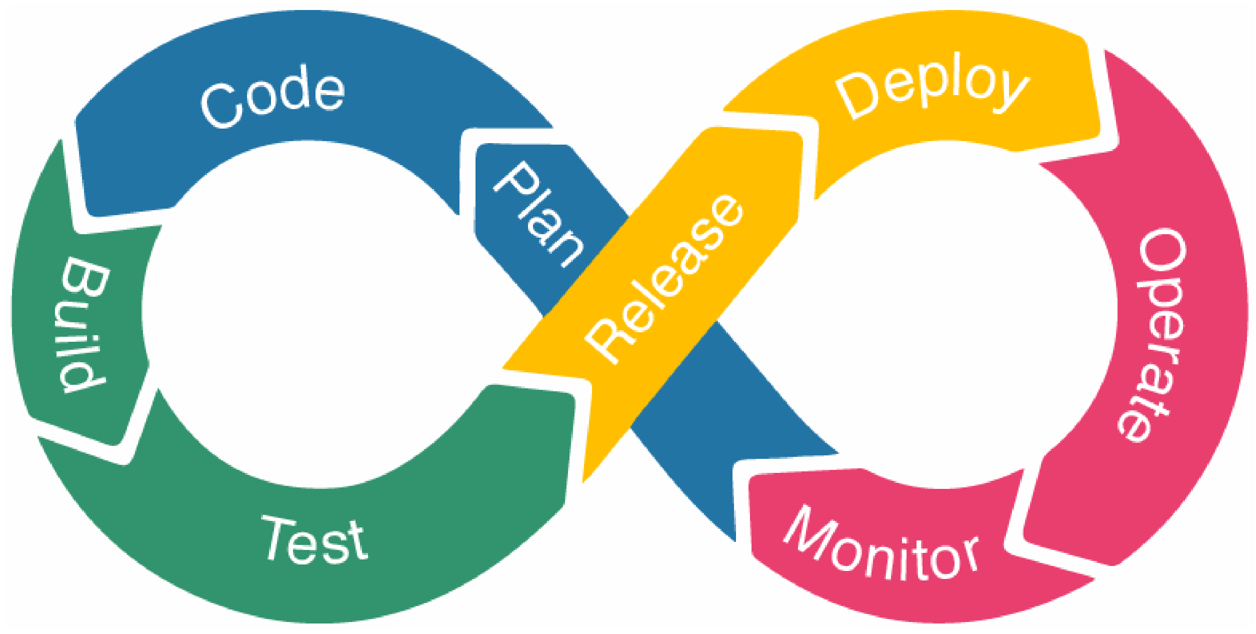
MT2020041 - Poojan Khatri

DevOps Basics

- Software Development is one of the most interesting and rapidly changing part of the IT infrastructure today.
- People use various methods for developing software and these methods change as per the changing times. DevOps is one such method for developing software.
- Development team is the team that designs the software according to the requirements specified by the customers. Operations team manage the software after it is released so that to assure that the users have seamless experience.
- There are many problems having two different teams for Development and Operations and so this is where DevOps comes into the picture where we integrate the Development and Operations side into one.

DevOps Pipeline

- The entire process of getting the system requirements, writing the code, making changes, deploying, monitoring etc. are all part of what is known as the DevOps pipeline. In DevOps pipeline people work together to execute each stage of the pipeline for reaching the same end-goal of creating a good product.
- The pipeline contains many stages and we will show the stages and the tools we have used in order to accomplish the task.



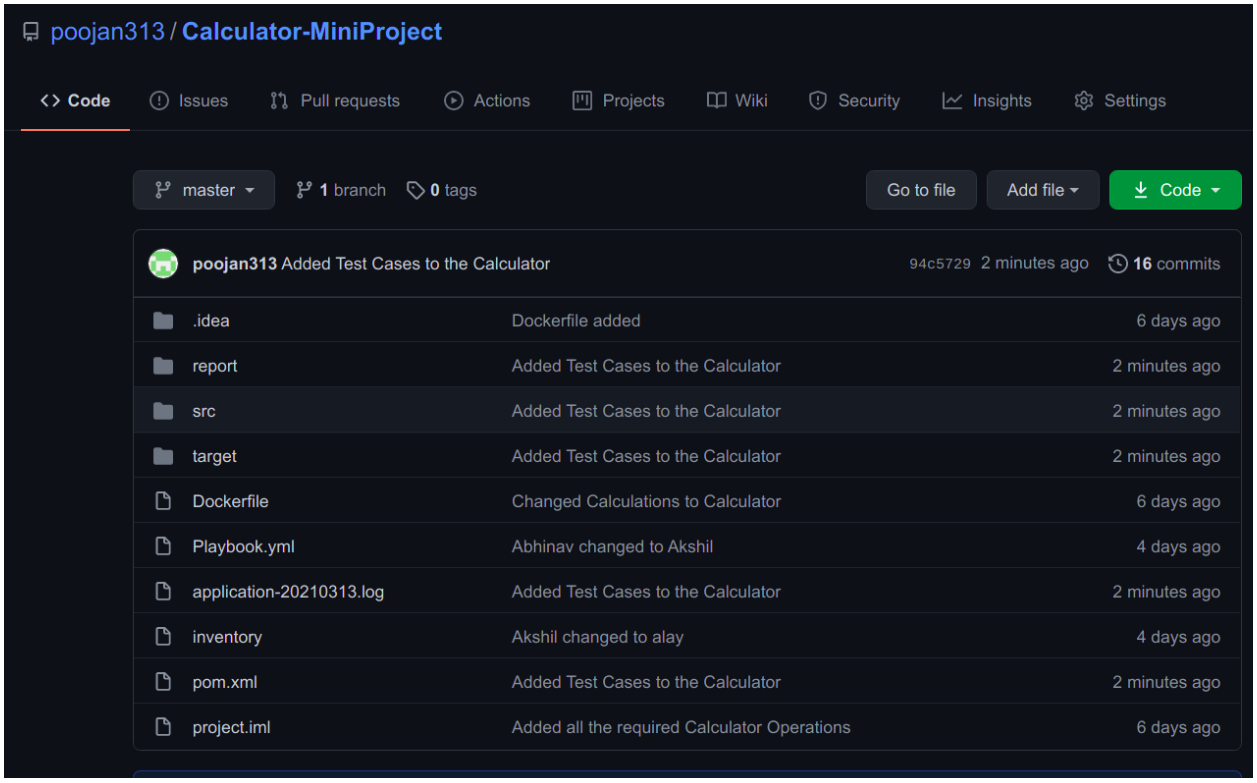
1. Programming Language - Java
2. Version Control - Github
3. Build Tool - Maven
4. Testing Tool - Junit
5. CI/CD Pipeline - Jenkins
6. Deployment and Configuration Management - Docker and Ansible
7. Logging - Log4j
8. Monitoring - ELK

Links :

1. Github Repo - https://github.com/poojan313/Calculator-MiniProject
2. Docker Image-

   https://hub.docker.com/repository/docker/poojan31399/miniproject

Source Code Management/ Version Control



poojan313 Added Test Cases to the Calculator 94c5729 2 minutes ago 16 commits

.idea	Dockerfile added	6 days ago
report	Added Test Cases to the Calculator	2 minutes ago
src	Added Test Cases to the Calculator	2 minutes ago
target	Added Test Cases to the Calculator	2 minutes ago
Dockerfile	Changed Calculations to Calculator	6 days ago
Playbook.yml	Abhinav changed to Akshil	4 days ago
application-20210313.log	Added Test Cases to the Calculator	2 minutes ago
inventory	Akshil changed to alay	4 days ago
pom.xml	Added Test Cases to the Calculator	2 minutes ago
project.iml	Added all the required Calculator Operations	6 days ago

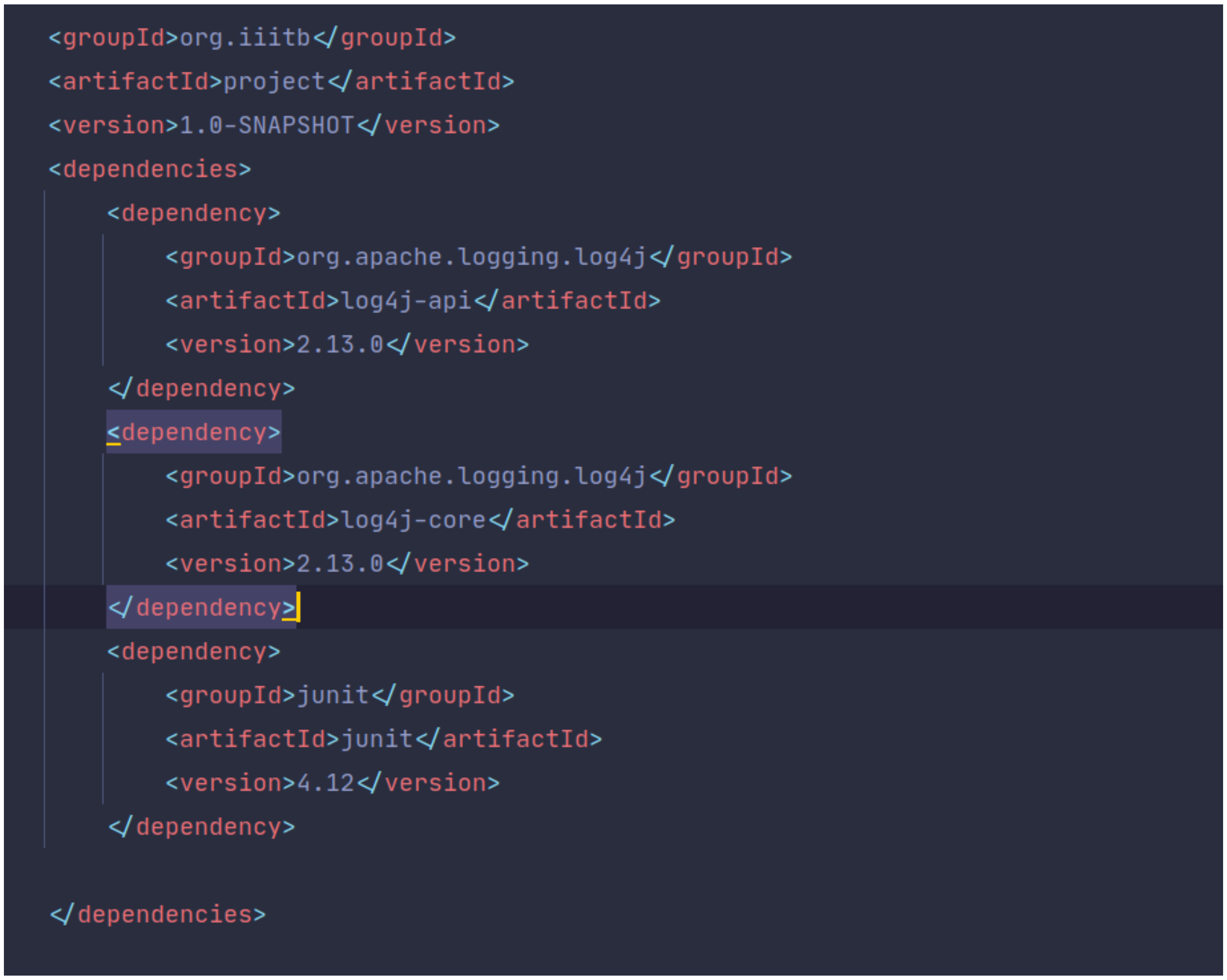
The commands to do version control are -

1. git init - Initialize a github repository
2. git add <filename> - Add the file to a staging area
3. git commit -m <message> - Commit the files to local repository
4. git push origin -u master - Push the changes to a remote repository

Using these and many more commands we can do version control.

Build Management

Here we use maven as a build tool. Instead of downloading each jar file and adding them to the java file, maven does this automatically for us. We just need to add the dependency in the pom.xml file and the rest is taken care of by the maven build tool. The pom.xml file is shown below.



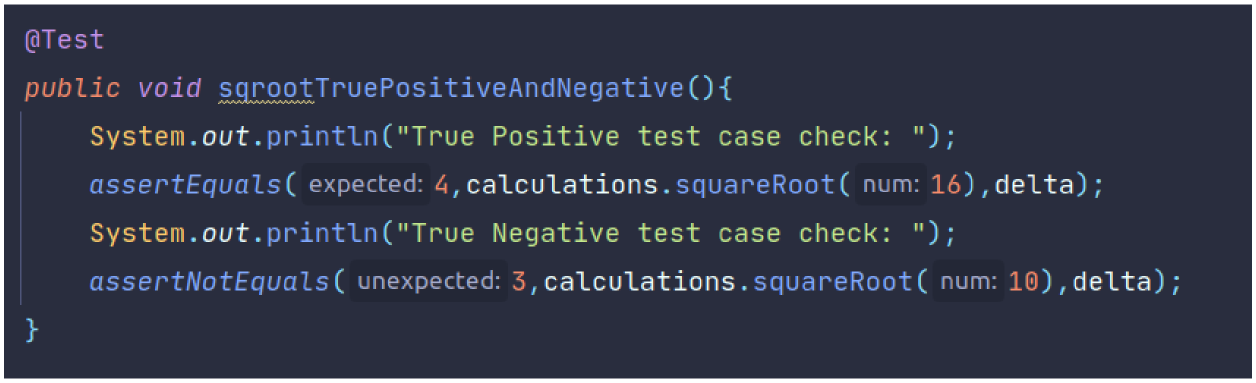
```xml
<groupId>org.iiitb</groupId>
<artifactId>project</artifactId>
<version>1.0-SNAPSHOT</version>
<dependencies>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.13.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.13.0</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
    </dependency>

</dependencies>
```

Testing(Unit Testing)

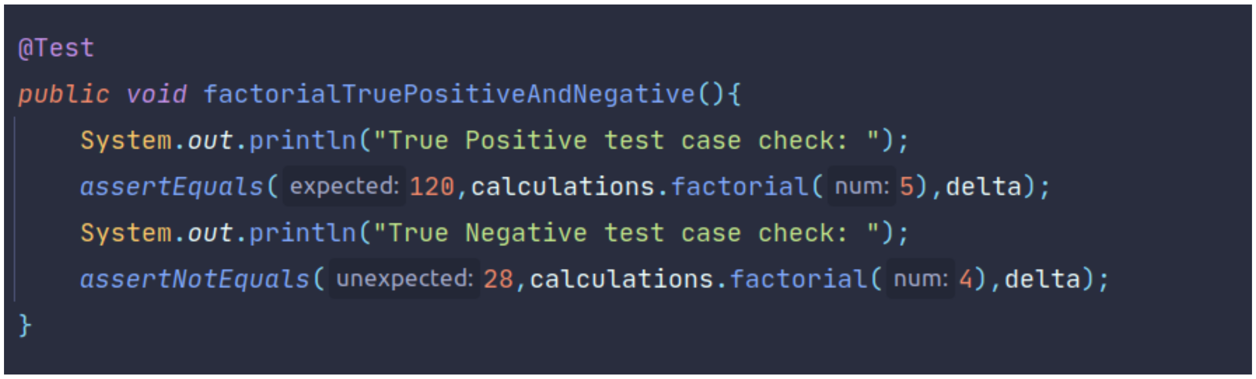
We use Junit Test cases to test the working of the Calculator. The true positive and negative cases for each of the functionalities implemented by us are shown below.

1. Square Root Function



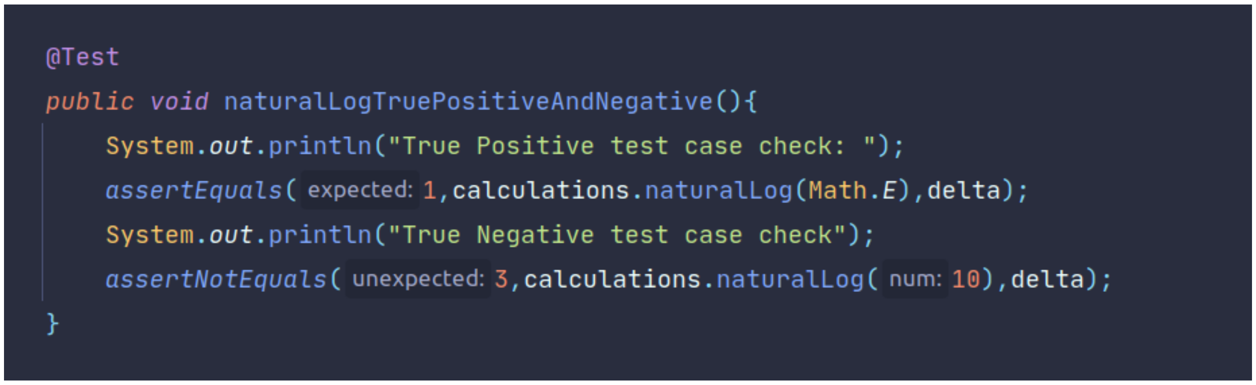
```
@Test
public void sqrootTruePositiveAndNegative(){
    System.out.println("True Positive test case check: ");
    assertEquals( expected: 4,calculations.squareRoot( num: 16),delta);
    System.out.println("True Negative test case check: ");
    assertNotEquals( unexpected: 3,calculations.squareRoot( num: 10),delta);
}
```

2. Factorial Function



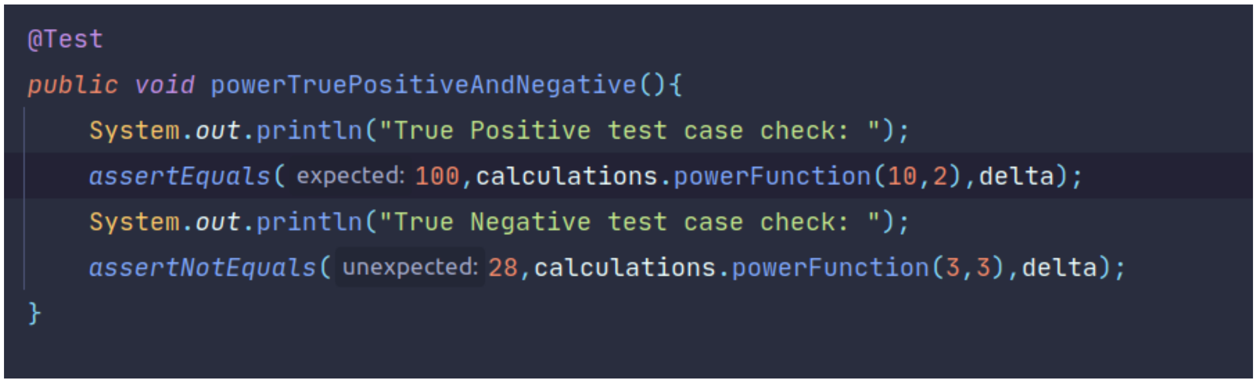
```
@Test
public void factorialTruePositiveAndNegative(){
    System.out.println("True Positive test case check: ");
    assertEquals( expected: 120,calculations.factorial( num: 5),delta);
    System.out.println("True Negative test case check: ");
    assertNotEquals( unexpected: 28,calculations.factorial( num: 4),delta);
}
```

3. Natural Log Function



```
@Test
public void naturalLogTruePositiveAndNegative(){
    System.out.println("True Positive test case check: ");
    assertEquals( expected: 1,calculations.naturalLog(Math.E),delta);
    System.out.println("True Negative test case check");
    assertNotEquals( unexpected: 3,calculations.naturalLog( num: 10),delta);
}
```

4. Power Function

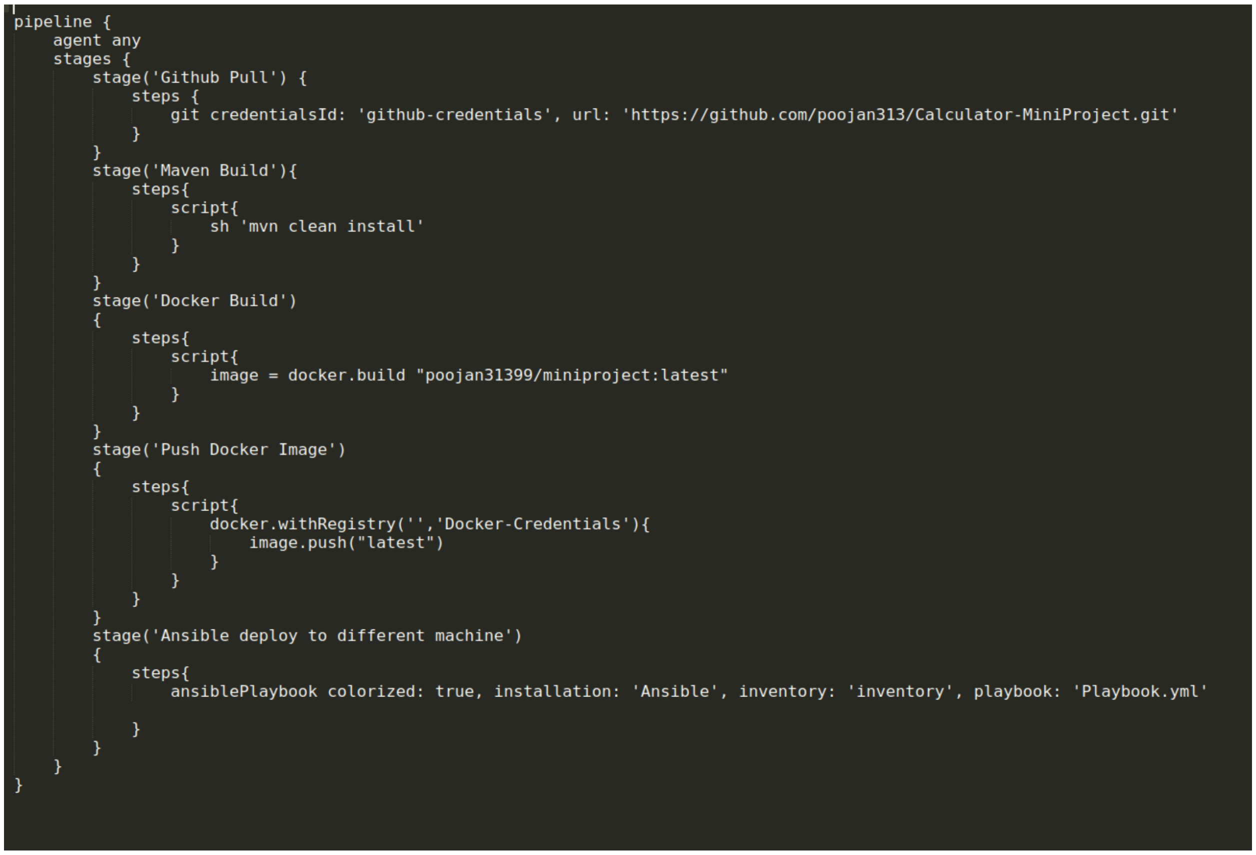


```
@Test
public void powerTruePositiveAndNegative(){
    System.out.println("True Positive test case check: ");
    assertEquals( expected: 100,calculations.powerFunction(10,2),delta);
    System.out.println("True Negative test case check: ");
    assertNotEquals( unexpected: 28,calculations.powerFunction(3,3),delta);
}
```

CI/CD Pipeline

To do continuous integration and development, we make use of jenkins which automatically runs all the necessary steps required for the different stages in the SDLC life cycle. The Jenkins Pipeline file and it's running is shown below.

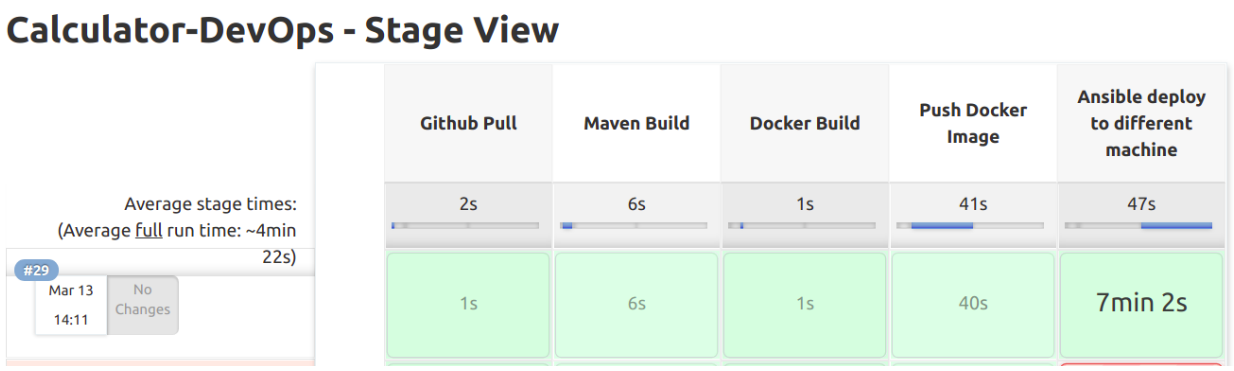
1. Pipeline file



```
pipeline {
    agent any
    stages {
        stage('Github Pull') {
            steps {
                git credentialsId: 'github-credentials', url: 'https://github.com/poojan313/Calculator-MiniProject.git'
            }
        }
        stage('Maven Build'){
            steps{
                script{
                    sh 'mvn clean install'
                }
            }
        }
        stage('Docker Build')
        {
            steps{
                script{
                    image = docker.build "poojan31399/miniproject:latest"
                }
            }
        }
        stage('Push Docker Image')
        {
            steps{
                script{
                    docker.withRegistry('','Docker-Credentials'){
                        image.push("latest")
                    }
                }
            }
        }
        stage('Ansible deploy to different machine')
        {
            steps{
                ansiblePlaybook colorized: true, installation: 'Ansible', inventory: 'inventory', playbook: 'Playbook.yml'

            }
        }
    }
}
```

2. Full Stage View

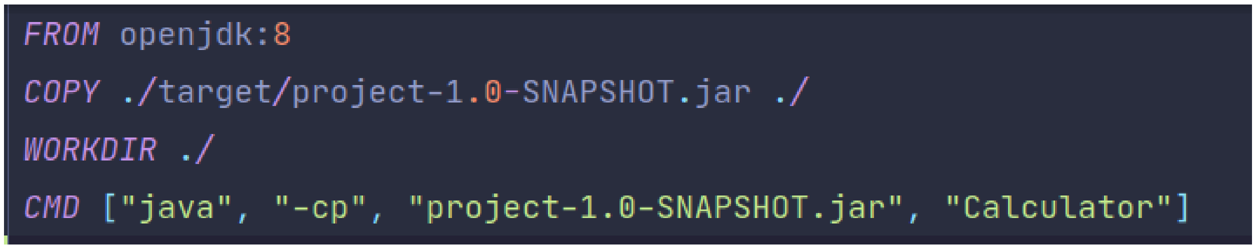


Calculator-DevOps - Stage View

	Github Pull	Maven Build	Docker Build	Push Docker Image	Ansible deploy to different machine
Average stage times: (Average full run time: ~4min 22s)	2s	6s	1s	41s	47s
#29 Mar 13 14:11 No Changes	1s	6s	1s	40s	7min 2s

Deployment

As using VM to check if the code is working on other machines is expensive, we use Docker instead. It creates a lightweight VM called container on which we run our code. Here we first use an OpenJDK8 image and build our project on top of it. Then we deploy this newly created image onto dockerhub. The Docker file is shown below along with the docker commands.



```
FROM openjdk:8
COPY ./target/project-1.0-SNAPSHOT.jar ./
WORKDIR ./
CMD ["java", "-cp", "project-1.0-SNAPSHOT.jar", "Calculator"]
```

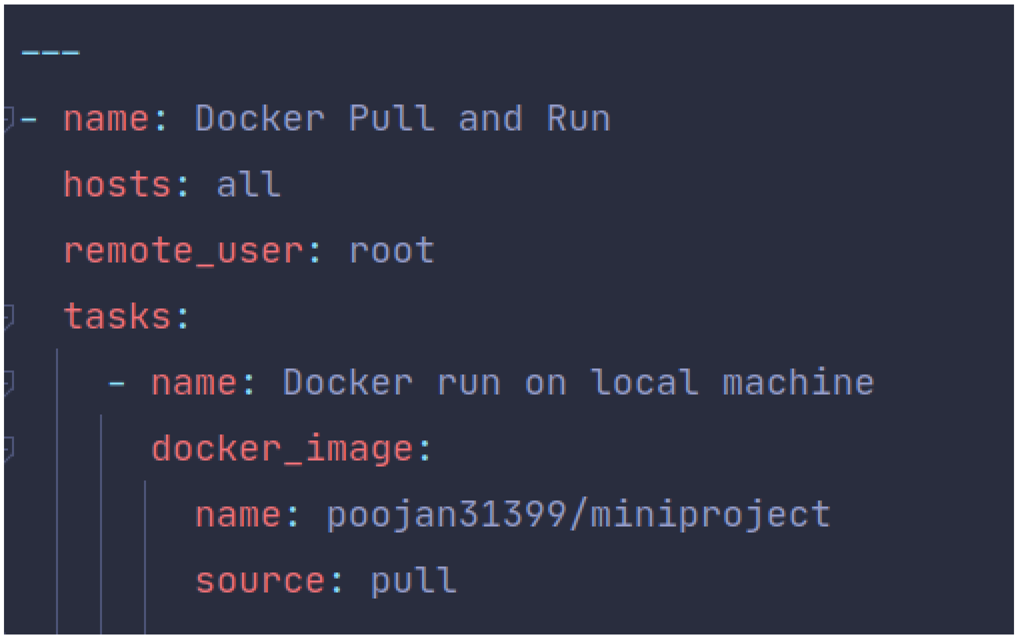
The docker commands we used are mentioned below :-

1. docker images - Get all existing docker images from local machine
2. docker pull <imagename:tag> - Get the docker image from docker hub
3. docker build <imagename:tag> - Create our own docker image
4. docker push <imagename:tag> - Push the image to dockerhub

Configuration Management

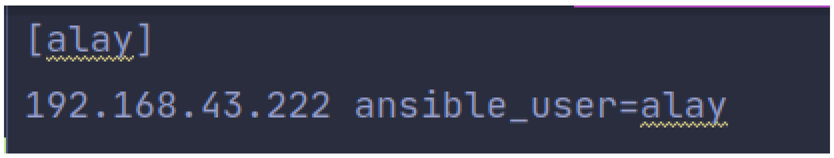
When running our project on different machines, some configurations need to match in order to run the project without any errors. For this we use Ansible where we use Infrastructure as Code (IaC) to configure all the hosts. We use playbook to describe all the tasks that need to be run on the remote hosts and we use an inventory file that tells what all hosts are there who have to be configured. The playbook and inventory files are shown below. I have used another machine as a host and my machine as the controller so the installations happen on the host machine whenever I run my Jenkins pipeline.

1. Playbook.yml



```yaml
---
- name: Docker Pull and Run
  hosts: all
  remote_user: root
  tasks:
    - name: Docker run on local machine
      docker_image:
        name: poojan31399/miniproject
        source: pull
```

2. Inventory

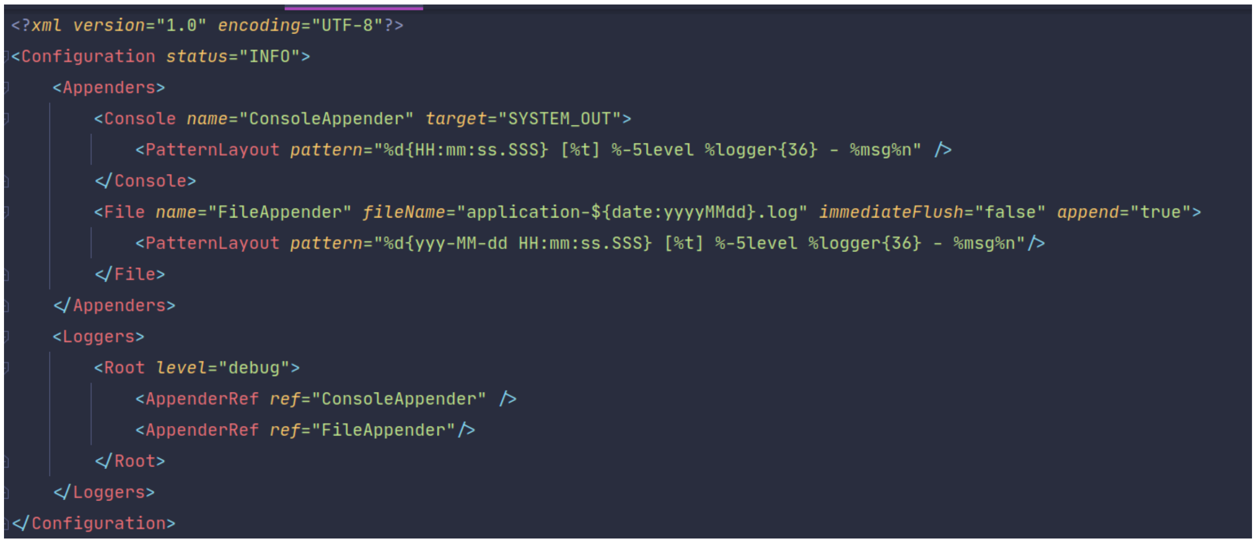


```
[alay]
192.168.43.222 ansible_user=alay
```

Logging

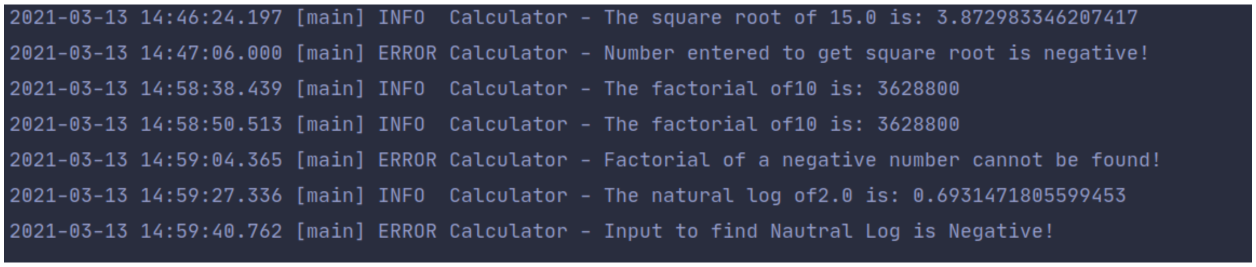
We generate a log file that contains the logs of all the actions performed and the errors generated during the execution. We have used Log4j which creates a log file as described in the log4j2.xml and stored inside the project folder itself. The log4j2.xml and the log file generated are shown below.

1. Log4j2.xml



```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="ConsoleAppender" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
        </Console>
        <File name="FileAppender" fileName="application-${date:yyyyMMdd}.log" immediateFlush="false" append="true">
            <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="ConsoleAppender" />
            <AppenderRef ref="FileAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

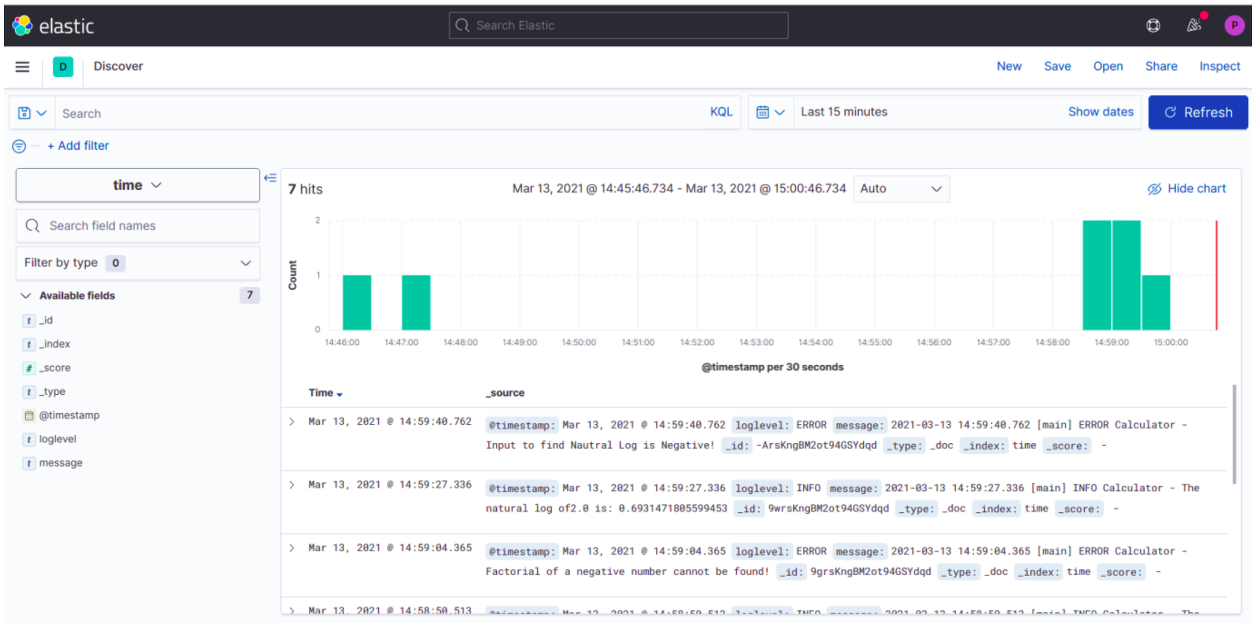
2. Log file



```
2021-03-13 14:46:24.197 [main] INFO  Calculator - The square root of 15.0 is: 3.872983346207417
2021-03-13 14:47:06.000 [main] ERROR Calculator - Number entered to get square root is negative!
2021-03-13 14:58:38.439 [main] INFO  Calculator - The factorial of10 is: 3628800
2021-03-13 14:58:50.513 [main] INFO  Calculator - The factorial of10 is: 3628800
2021-03-13 14:59:04.365 [main] ERROR Calculator - Factorial of a negative number cannot be found!
2021-03-13 14:59:27.336 [main] INFO  Calculator - The natural log of2.0 is: 0.6931471805599453
2021-03-13 14:59:40.762 [main] ERROR Calculator - Input to find Nautral Log is Negative!
```

Monitoring

We have to now monitor what is going on with our project and if it is functioning as expected or are there some issues. We do all this type of monitoring using ELK - ElastiSearch, LogStash and Kibana. We use our log file to generate a visual representation of with the help of Kibana which are shown below.



```
Time    _source
Mar 13, 2021 @ 14:59:40.762  @timestamp: Mar 13, 2021 @ 14:59:40.762 loglevel: ERROR message: 2021-03-13 14:59:40.762 [main] ERROR Calculator -
Input to find Nautral Log is Negative! _id: -ArsKngBM2ot94GSYdqd _type: _doc _index: time _score: -

Mar 13, 2021 @ 14:59:27.336  @timestamp: Mar 13, 2021 @ 14:59:27.336 loglevel: INFO message: 2021-03-13 14:59:27.336 [main] INFO Calculator - The
natural log of2.0 is: 0.6931471805599453 _id: 9wrsKngBM2ot94GSYdqd _type: _doc _index: time _score: -

Mar 13, 2021 @ 14:59:04.365  @timestamp: Mar 13, 2021 @ 14:59:04.365 loglevel: ERROR message: 2021-03-13 14:59:04.365 [main] ERROR Calculator -
Factorial of a negative number cannot be found! _id: 9grsKngBM2ot94GSYdqd _type: _doc _index: time _score: -

Mar 13, 2021 @ 14:58:50.513
```

Conclusion

Thus, using all the above DevOps tools we are successfully able to Build, Test, Deploy and Monitor our Calculator Application.