

SPE Mini project Report

Done by: Aakash Bhardwaj (MT2023143)

Quick Links:

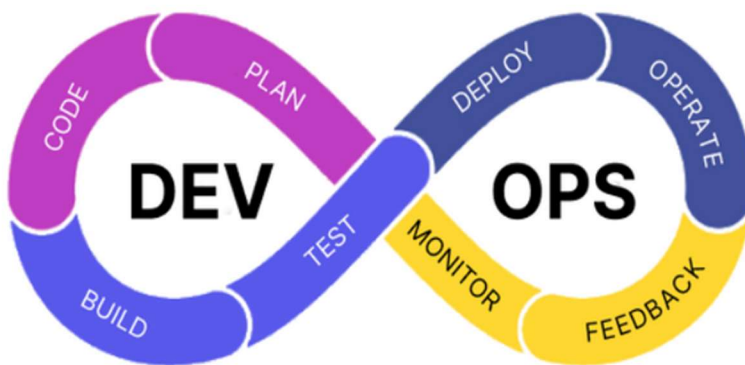
- [GitHub Repository](#)
 - [Docker Image or Docker Hub](#)
-

DevOps

About DevOps

- DevOps connects software development with IT operations.
- It emphasizes close collaboration throughout the software lifecycle.
- Aims to streamline the process of developing, testing, deploying, and maintaining software.
- Focuses on improving delivery speed, efficiency, and quality.
- Involves automating processes to enhance workflow efficiency.
- Encourages continuous integration, continuous delivery, and continuous deployment.
- Emphasizes communication, feedback, and shared responsibility among teams.
- Ultimately aims to deliver high-quality software to customers quickly and reliably.

DevOps Cycle



Why DevOps?

- DevOps enables rapid and frequent releases, enhancing agility and customer satisfaction.
- Enhanced collaboration leads to faster problem detection and resolution, reducing downtime.
- Automation frees up time for innovation, thanks to practices like continuous integration and deployment.
- Utilizing infrastructure as code (IaC) improves resource efficiency, reduces errors, and ensures reproducibility.

Tools Used

- **Version Control System (VCS): Git**
 - Utilized Git for distributed version control.
 - Leveraged GitHub issues and commits for efficient tracking of changes made to files.
- **Unit Testing Framework: JUnit**
 - Employed JUnit for comprehensive unit testing of the four calculator functions.
 - Ensured reliability and functionality of the code through rigorous testing practices.
- **Build Automation Tool: Maven**
 - Implemented Maven for streamlined build processes.
 - Simplified dependency management and facilitated easy deployment of Java projects.
- **Continuous Integration (CI) Tool: GitHub Actions**
 - Integrated GitHub Actions for automated CI/CD workflows.
 - Enabled execution of predefined actions and commands on the codebase at scheduled intervals.
- **Containerization Platform: Docker**

- Utilized Docker for containerizing projects into lightweight and portable containers.
- Improved scalability and consistency of deployment environments.
- **Configuration Management Tool: Ansible**
 - Leveraged Ansible for automating deployment and infrastructure management.
 - Ensured consistency and reliability across various deployment environments.

Plugins Used

1.) Webhook:

- A method of automating data exchange between web applications, where a web server sends data to a specified URL (endpoint) in real-time, triggered by specific events or updates, enabling seamless integration and communication between different systems without requiring continuous polling.

2.) Ngrok:

- Secure tunneling service that exposes local servers behind NATs and firewalls to the public internet over secure tunnels, facilitating easy testing and sharing of web applications.
- With Ngrok's comprehensive dashboard and logging features, users can monitor traffic, inspect requests, and analyze usage patterns to troubleshoot issues and optimize performance.

3.) Ansible Plugin:

- Ansible plugins enhance automation capabilities by extending functionality.
- Custom plugins enable tailored solutions for specific infrastructure needs.
- Community collaboration through plugin sharing enriches the Ansible ecosystem

4.) Docker Plugin:

- Simplifies the management and orchestration of Docker containers within the Ansible framework.
- Enables tasks such as building, running, and managing Docker containers across distributed environments.
- Integrates seamlessly with Ansible playbooks, allowing for consistent and automated Docker container deployment and configuration.

Project Structure

```
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~/Term 2/SPE_Calculator$ tree
```

```
.
├── deploy.yml
├── Dockerfile.txt
├── inventory.txt
├── jenkinsfile
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com
│   │   │   │   ├── swasthsahayak
│   │   │   │   │   └── Main.java
│   │   └── resources
│   └── test
│       ├── java
│       │   ├── com
│       │   │   ├── swasthsahayak
│       │   │   │   └── MainTest.java
│       └── resources
└── target
    ├── classes
    │   ├── com
    │   │   ├── swasthsahayak
    │   │   │   └── Main.class
    │   └── META-INF
    ├── generated-sources
    │   └── annotations
    ├── generated-test-sources
    │   └── test-annotations
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   ├── default-compile
    │       │   │   ├── createdFiles.lst
    │       │   │   └── inputFiles.lst
    │       └── testCompile
    │           ├── default-testCompile
    │           │   ├── createdFiles.lst
    │           │   └── inputFiles.lst
    ├── surefire-reports
    │   ├── com.swasthsahayak.MainTest.txt
    │   └── TEST-com.swasthsahayak.MainTest.xml
    └── test-classes
        ├── com
        │   ├── swasthsahayak
        │   │   └── MainTest.class
        └── META-INF
```

This is the structure of the project. Main file contains the JAVA code and corresponding test cases are written in Test Java file. Another important file here is pom.xml which has configuration file for Maven project. The target folder contains the generated build artifacts for the project.

dockerfile contains containerization instructions and deploy.yml and inventory are used in setting up a local instance of Ansible

```

8      @Test
9      public void testCalculateSquareRoot() {
10         double number = 25;
11         double expectedResult = 5;
12         double actualResult = Main.calculateSquareRoot(number);
13         assertEquals(expectedResult, actualResult, 0.0001);
14     }

```

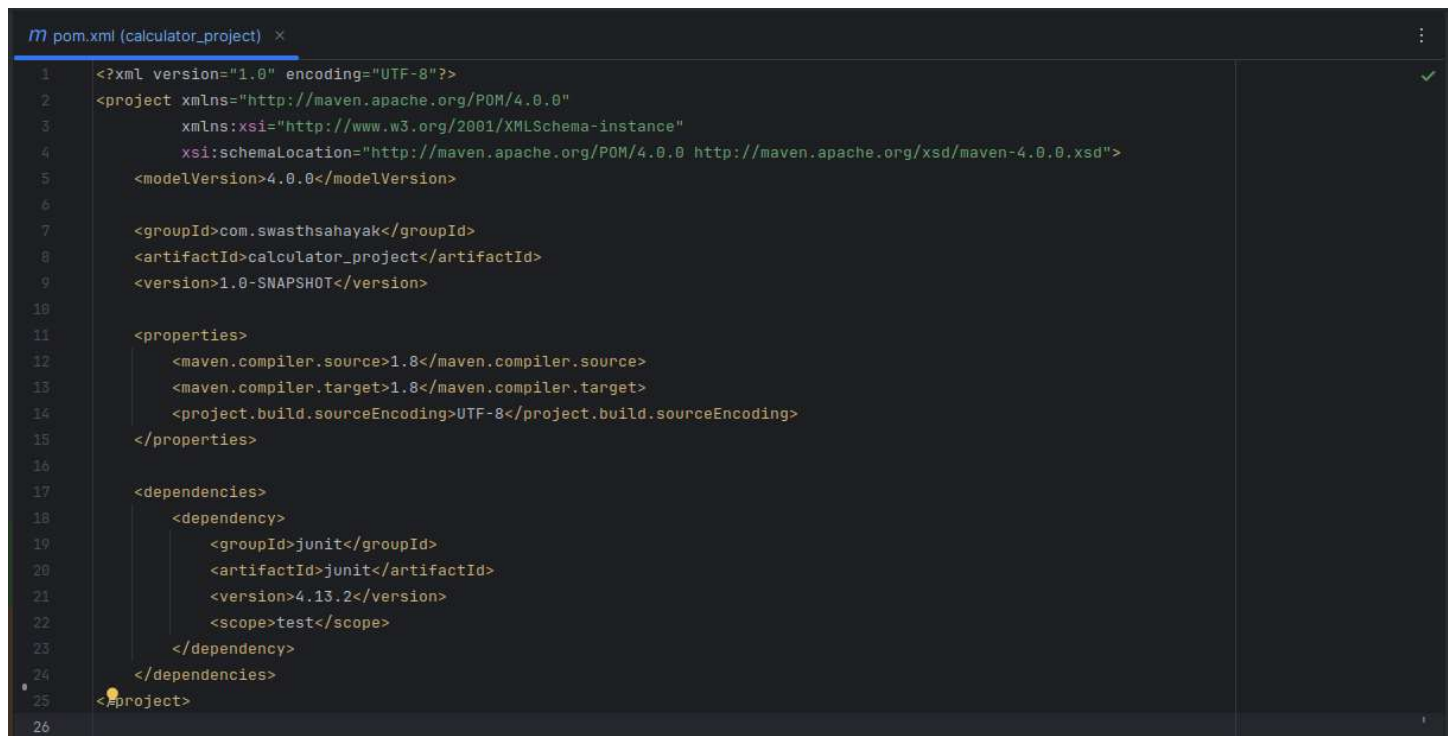
Similar test cases are all the functionalities. If all the test cases pass, only then the jar file will be generated. Here the assertEquals() function compares the actual result returned with the expected result.

@Test: This annotation indicates that the method testCalculateSquareRoot() is a test case and should be executed by the JUnit framework.

JUnit Framework:

Test Suite Overview: Describe the structure and size of the test suite.

Test Results: Provide a summary of the test execution, including the total number of tests executed, passed, and failed.



```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.swasthsahayak</groupId>
8     <artifactId>calculator_project</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <properties>
12         <maven.compiler.source>1.8</maven.compiler.source>
13         <maven.compiler.target>1.8</maven.compiler.target>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15     </properties>
16
17     <dependencies>
18         <dependency>
19             <groupId>junit</groupId>
20             <artifactId>junit</artifactId>
21             <version>4.13.2</version>
22             <scope>test</scope>
23         </dependency>
24     </dependencies>
25 </project>
26

```

This pom.xml file includes properties for encoding and Java version, dependencies for JUnit. It also contains the properties of maven.

This is the pom.xml file that lists the properties and dependencies intended for the project.

The project is cleaned, built, and tested as follows:

mvn clean install/mvn clean package. Both can serve the purpose of creation of .jar file.

If any test case fails, the build will fail and the jar file will not be created.

```
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~/Term 2/SPE_Calculator$ mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.swasthsahayak:calculator_project >-----
[INFO] Building calculator_project 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ calculator_project ---
[INFO] Deleting /home/aakash/Term 2/SPE_Calculator/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ calculator_project ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ calculator_project ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/aakash/Term 2/SPE_Calculator/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ calculator_project ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/aakash/Term 2/SPE_Calculator/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ calculator_project ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/aakash/Term 2/SPE_Calculator/target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ calculator_project ---
[INFO] Surefire report directory: /home/aakash/Term 2/SPE_Calculator/target/surefire-reports

-----
T E S T S
-----
Running com.swasthsahayak.MainTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.053 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ calculator_project ---
[INFO] Building jar: /home/aakash/Term 2/SPE_Calculator/target/calculator_project-1.0-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ calculator_project ---
[INFO] Installing /home/aakash/Term 2/SPE_Calculator/target/calculator_project-1.0-SNAPSHOT.jar to /home/aakash/.m2/repository/com/swasthsahayak/calculator_project/1.0-SNAPSHOT/calculator_project-1.0-SNAPSHOT.jar
[INFO] Installing /home/aakash/Term 2/SPE_Calculator/pom.xml to /home/aakash/.m2/repository/com/swasthsahayak/calculator_project/1.0-SNAPSHOT/calculator_project-1.0-SNAPSHOT.pom
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.883 s
[INFO] Finished at: 2024-02-24T18:55:56+05:30
[INFO] -----
```

Docker build and run

```
1 FROM ubuntu:latest
2 RUN apt-get update && apt-get install -y openjdk-11-jdk
3 COPY ./target/calculator_project-1.0-SNAPSHOT.jar ./
4 WORKDIR ./
5 CMD ["java", "-cp", "calculator_project-1.0-SNAPSHOT.jar", "com.swasthsahayak.Main"]
```

This is the dockerfile that is used to generate a container with the jar file.

Setting up jenkins pipeline:

Step 1: Setting up Jenkins job.

- We create a pipeline project in new projects with build trigger as GitHub hook for webhook. It will automatically build project if any commit happens on GitHub.

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

b. Setting up repository

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/aakash-bhardwaj-1/SPE-MiniProject.git

Credentials ?

- none -

+ Add +

Advanced ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

Script Path ?

jenkinsfile

☒ Lightweight checkout ?

Save Apply

Step 2: Adding maven and tools in Jenkins

Dashboard > Manage Jenkins > Tools

Tools

Maven Configuration

Default settings provider

Use default maven settings

Default global settings provider

Use default maven global settings

JDK installations

JDK installations ▼ Edited

Git installations

≡ Git

Name

Default

Path to Git executable ?

/usr/bin/git

☐ Install automatically ?

Add Git ▼

These tools play a crucial role in streamlining the automation process within the Jenkins pipeline. Their seamless integration with the corresponding plugins enhances the efficiency and effectiveness of the development and deployment workflows. Leveraging these tools ensures smooth execution and management of various tasks, contributing to a more streamlined and productive development environment.

Step 3: Incorporating a GitHub repository into a project containing files such as a Maven project, Jenkinsfile, Dockerfile, inventory, and deploy.yml.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/aakash-bhardwaj-1/SPE-MiniProject.git

Credentials ?

- none -

+ Add ▾

Advanced ▾

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/master

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

Script Path ?

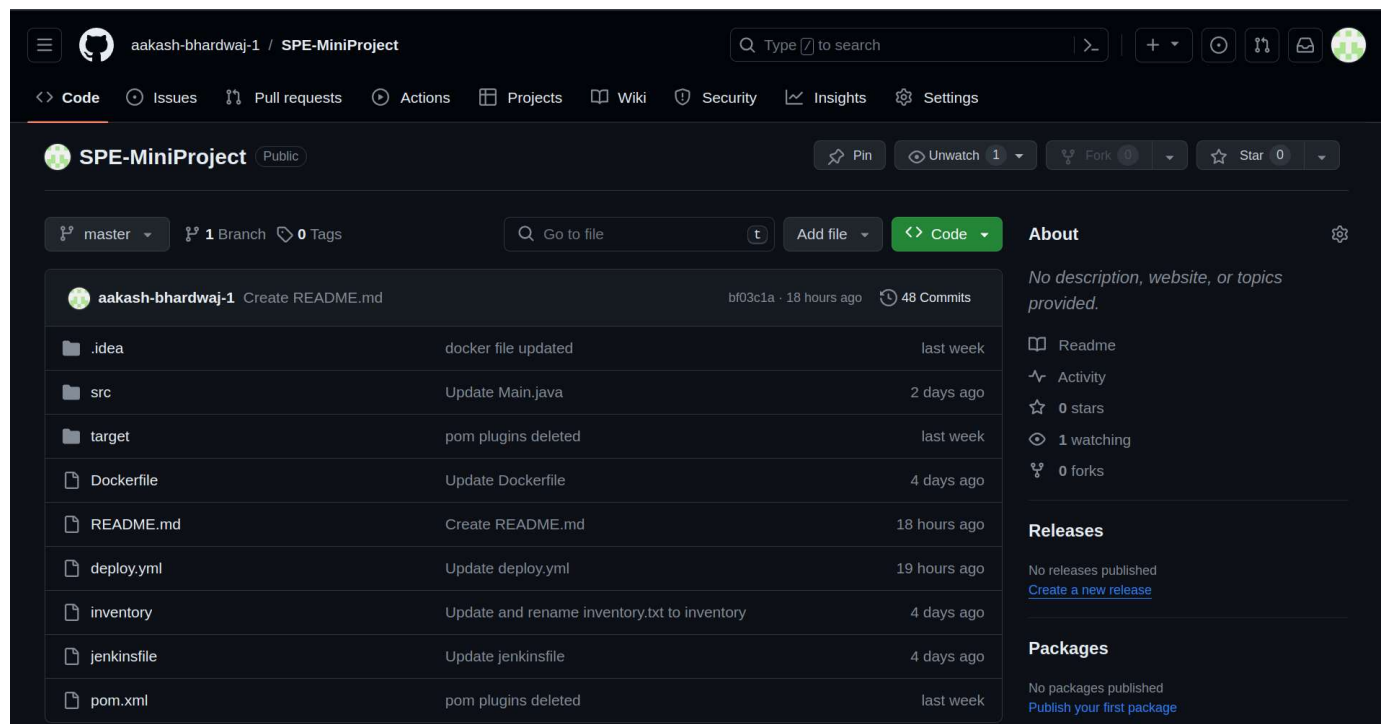
jenkinsfile

☒ Lightweight checkout ?

Save

Apply

This is how GitHub repository looks like.



Step 4 : Adding credentials

Docker credentials: These are the credentials for DockerHub

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > aakashbhardwaj1/***** (Docker Hub Credentials)

Update

Delete

Move

Update credentials

Scope ?
Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?
aakashbhardwaj1

☐ Treat username as secret ?

Password ?
Concealed Change Password

ID ?
DockerHubCred

Description ?
Docker Hub Credentials

Localhost credentials for Ansible

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > xyz/***** (localhost user credentials)

Update

Delete

Move

Update credentials

Scope ?
Global (Jenkins, nodes, items, all child items, etc)

Username ?
xyz

☐ Treat username as secret ?

Password ?
Concealed

Change Password

ID ?
localhost

Description ?
localhost user credentials

Save

We are now set to build the project. Executed pipeline looks like

Average stage times:
(Average full run time: ~30s)

	Declarative: Checkout SCM	Declarative: Tool Install	Checkout	maven build	maven test	maven artifacts	Build Docker Image	Push Docker Images	Run Ansible Playbook
	1s	72ms	963ms	2s	2s	167ms	545ms	17s	2s
#43 Feb 23 23:57 1 commit	1s	56ms	954ms	2s	2s	127ms	445ms	18s	2s

Setting up Webhooks

Tool used: NGROK

Discription:

NGROK is a powerful tool used for creating secure tunnels to localhost or local servers, allowing them to be accessed remotely over the internet. It provides a way to expose local development servers, APIs, or any TCP/UDP service to the internet without the need for public IP addresses or complicated network configurations.

- Secure Tunneling: NGROK establishes secure tunnels from a public endpoint to a locally running network service, ensuring that data transmitted through the tunnel is encrypted and secure.

- **Localhost Exposing:** It enables developers to expose their local development environment, typically running on localhost, to the internet, allowing for testing and sharing of web applications, APIs, or other services.

Run NGROK

Start ngrok: Run ngrok with the desired port of your Jenkins server where the webhook will be listening. For example, if Jenkins is running on port 8080, you would run: `./ngrok http 8080`.

```
ngrok (Ctrl+C to quit)
Take our ngrok in production survey! https://forms.gle/aXiBFWzEA36DudFn6

Session Status      online
Account             Aakash Bhardwaj (Plan: Free)
Version             3.6.0
Region              India (in)
Latency              28ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://90a3-106-51-162-153.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
                    0      0      0.00   0.00   0.00   0.00
```

Copy the ngrok forwarding URL: Once ngrok is running, it will provide a forwarding URL <http://90a3-106-51-162-153.ngrok-free.app>. Copy this URL as it will be used as the webhook URL.

Configure webhook in Jenkins: Go to your Jenkins job configuration, find the section for adding webhooks, and paste the ngrok forwarding URL as the webhook URL. This ensures that Jenkins will receive webhook notifications from your version control system (e.g., GitHub, GitLab) via ngrok.

Dashboard > Manage Jenkins > System >

Jenkins Location

Jenkins URL ?

https://ebea-2409-40f2-103d-2088-cabd-a546-bd66-8c44.ngrok-free.app/

System Admin e-mail address ?

jenkins-Master<aakashbhardwaj727@gmail.com>

Set up webhook in version control system: In your version control system (e.g., GitHub, GitLab), configure a webhook to send push events to the ngrok forwarding URL. This will trigger Jenkins builds whenever a commit is pushed to the repository.

The screenshot shows the GitHub repository settings for 'aakash-bhardwaj-1 / SPE-MiniProject'. The 'Settings' tab is selected, and the 'Webhooks / Manage webhook' section is active. The 'Settings' sub-tab is chosen, displaying the configuration for a new webhook. The 'Payload URL' is set to '-103d-2088-cabd-a546-bd66-8c44.ngrok-free.app/github-webhook/'. The 'Content type' is set to 'application/json'. The 'Secret' section shows a warning: 'If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated. — [Change Secret](#)'. The 'SSL verification' section has 'Enable SSL verification' selected. The 'Which events would you like to trigger this webhook?' section is partially visible at the bottom.

Once you've set up the webhook in your version control system, it will send a POST request to the ngrok forwarding URL whenever the specified event (e.g., code push) occurs. Ngrok will

then forward this request to your locally running Jenkins server, triggering the configured job to build automatically.

Code for various stages

Project Code: Language used is JAVA

```
1  package com.swasthsahayak;
2
3  import java.util.Scanner;
4
5  public class Main {
6      public static void main(String[] args) {
7          Scanner scanner = new Scanner(System.in);
8          double number;
9          int choice;
10
11          do {
12              System.out.println("Choose any operation:");
13              System.out.println("1. Square root");
14              System.out.println("2. Factorial");
15              System.out.println("3. Natural logarithm");
16              //System.out.println("4. Power");
17              System.out.println("5. Exit");
18
19              System.out.print("Enter your choice: ");
20              choice = scanner.nextInt();
21
22              switch (choice) {
23                  case 1:
24                      System.out.print("Enter a number to calculate its square root: ");
25                      number = scanner.nextDouble();
26                      System.out.println("Square root of " + number + " is: " + calculateSquareRoot(number));
27                      break;
28                  case 2:
29                      System.out.print("Enter a number to calculate its factorial: ");
30                      number = scanner.nextDouble();
31                      System.out.println("Factorial of " + (int) number + " is: " + calculateFactorial((int) number));
32                      break;
33                  case 3:
34                      System.out.print("Enter a number to calculate its natural logarithm: ");
35                      number = scanner.nextDouble();
36                      System.out.println("Natural logarithm of " + number + " is: " + calculateNaturalLog(number));
37                      break;
38                  // case 4:
39                  //     System.out.print("Enter the base number: ");
40                  //     double base = scanner.nextDouble();
41                  //     System.out.print("Enter the exponent: ");
42                  //     double exponent = scanner.nextDouble();
43                  //     System.out.println(base + " raised to the power of " + exponent + " is: " + calculatePower(base, exponent));
44                  //     break;
45                  case 5:
46                      System.out.println("Exiting...");
47                      break;
48                  default:
49                      System.out.println("Invalid choice. Please enter a number between 1 and 5.");
50              }
51          } while (choice != 5);
52      }
53
54      public static double calculateSquareRoot(double n) {
55          return Math.sqrt(n);
56      }
```

TestCases Code:

```
1  package com.swasthsahayak;
2
3  import static org.junit.Assert.assertEquals;
4  import org.junit.Test;
5
6  ✓ public class MainTest {
7
8      @Test
9  ✓  public void testCalculateSquareRoot() {
10         double number = 25;
11         double expectedResult = 5;
12         double actualResult = Main.calculateSquareRoot(number);
13         assertEquals(expectedResult, actualResult, 0.0001);
14     }
15
16     @Test
17  ✓  public void testCalculateFactorial() {
18         int number = 5;
19         long expectedResult = 120;
20         long actualResult = Main.calculateFactorial(number);
21         assertEquals(expectedResult, actualResult);
22     }
23
24     @Test
25  ✓  public void testCalculateNaturalLog() {
26         double number = 10;
27         double expectedResult = 2.302585092994046;
28         double actualResult = Main.calculateNaturalLog(number);
29         assertEquals(expectedResult, actualResult, 0.0001);
30     }
```

- This Java code defines test methods for mathematical operations such as square root, factorial, and natural logarithm.
- Each test verifies the correctness of the corresponding method in the Main class by comparing expected results with actual results.
- The commented-out test method for calculating power is not currently in use but can be activated and modified as needed for testing power calculations.

Jenkinsfile

```
1  pipeline {
2      agent any
3      tools {
4          maven "mvn"
5          jdk "jdk"
6      }
7      environment {
8          DOCKER_IMAGE_NAME = 'calculator_mini'
9          GITHUB_REPO_URL = 'https://github.com/aakash-bhardwaj-1/SPE-MiniProject.git'
10     }
11
12     stages {
13         stage('Checkout') {
14             steps {
15                 script {
16                     // Checkout the code from the GitHub repository
17                     git branch: 'master', url: "${GITHUB_REPO_URL}"
18                 }
19             }
20         }
21
22         stage('maven build') {
23             steps {
24                 script {
25                     mvnHome = tool "mvn"
26                     sh "${mvnHome}/bin/mvn clean package"
27                 }
28             }
29         }
30
31         stage('maven test') {
32             steps {
33                 script {
34                     sh "${mvnHome}/bin/mvn test"
35                 }
36             }
37         }
38
39         stage('maven artifacts') {
40             steps {
41                 script {
42                     archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
43                 }
44             }
45         }
46
47         stage('Build Docker Image') {
48             steps {
49                 script {
50                     // Build Docker image
51                     docker.build("${DOCKER_IMAGE_NAME}", '.')
52                 }
53             }
54         }
55
56
57         stage('Push Docker Images') {
58             steps {
59                 script {
60                     docker.withRegistry('https://index.docker.io/v1/', 'DockerHubCred') {
61                         sh 'docker tag calculator_mini aakashbhardwaj1/calculator_mini'
62                         sh 'docker push aakashbhardwaj1/calculator_mini'
63                     }
64                 }
65             }
66         }
67
68         stage('Run Ansible Playbook') {
69             steps {
70                 script {
71                     ansiblePlaybook(
72                         playbook: 'deploy.yml',
73                         inventory: 'inventory'
74                     )
75                 }
76             }
77         }
78     }
79 }
```

- This Jenkinsfile defines a declarative pipeline that automates the build, test, and deployment process of a project.
- It utilizes tools such as Maven and JDK specified in the 'tools' section to build and test the project.
- The pipeline includes stages for checking out code from a GitHub repository, building Maven artifacts, building a Docker image, pushing the Docker image to DockerHub, and running an Ansible playbook for deployment.

Inventory

```

1  all:
2      hosts:
3          127.0.0.1:
4              ansible_connection: local
5              ansible_user: xyz
6              ansible_ssh_pass: xyz
7              ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
8              ansible_python_interpreter: /usr/bin/python3

```

- This Ansible inventory file defines a group named "all" containing a single host with the IP address 127.0.0.1.
- The host is configured to use a local connection (`ansible_connection: local`) with the specified user (`ansible_user`) and SSH password (`ansible_ssh_pass`).
- Additional SSH arguments are provided to disable strict host key checking (`ansible_ssh_common_args: '-o StrictHostKeyChecking=no'`) and specify the Python interpreter (`ansible_python_interpreter`).

Ansible File(Deploy.yml):

- Given below is a Ansible playbook defines tasks to pull a Docker image from Docker Hub (`aakashbhardwaj1/calculator_mini`) and start a container.
- The playbook targets the `localhost` host with the specified remote user (`aakash`) and does not escalate privileges (`become: false`).

- It includes tasks to pull the Docker image using the `docker_image` module, display the result using the `debug` module, start the Docker service using the `service` module, and run a container using the `shell` module with the `docker run` command.

```
1  ---
2  - name: Pull Docker Image from Docker Hub
3    hosts: localhost
4    remote_user: aakash
5    become: false
6    tasks:
7      - name: Pull Docker Image
8        docker_image:
9          name: "aakashbhardwaj1/calculator_mini"
10         | source: pull
11        register: docker_pull_result
12
13      - name: Display Docker Pull Result
14        debug:
15          var: docker_pull_result
16
17      - name: Start Docker service
18        service:
19          name: docker
20          state: started
21
22      - name: Running container
23        shell: docker run -it -d aakashbhardwaj1/calculator_mini
```

Running Container on local machine:

1. Check if container already exists

```
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

Using '`docker ps`' without any flags displays only the running containers. However, appending the '`-a`' flag extends this functionality to show all containers, including those that have exited or been stopped. This comprehensive view can be useful for managing and troubleshooting containers across various states.

2. Building Pipeline



- Building a pipeline to create a container accessible via the terminal involves several steps. Firstly, you would typically define your container environment using a Dockerfile, specifying the base image, dependencies, and commands to run. Then, you would use a CI/CD tool like Jenkins, GitLab CI/CD, or GitHub Actions to automate the build process. This involves setting up triggers, such as code commits or scheduled builds, to initiate the pipeline.
- Within the pipeline, you would include stages such as building the Docker image from the Dockerfile, tagging the image for versioning, and pushing it to a container registry like Docker Hub or a private registry. Once the image is successfully built and pushed, you can use commands like '`docker run`' to instantiate a container from the image. To ensure accessibility via the terminal, you may expose ports and configure networking appropriately in the Dockerfile or through runtime flags when running the container.

- Additionally, you might incorporate tests within the pipeline to validate the functionality of the containerized application. This could involve running unit tests, integration tests, or even deploying the container to a staging environment for further validation. Overall, building a pipeline for creating a container accessible via the terminal streamlines the development and deployment process, enabling consistent and reliable delivery of containerized applications.

3. Validating Docker Container Visibility

```
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e46fb3486b4a	aakashbhardwaj1/calculator_mini	"java -cp calculator..."	22 seconds ago	Up 22 seconds		musing_villani

- Verifying the presence and accessibility of a Docker container within your environment is crucial for effective management and troubleshooting. By confirming its visibility, you can ensure that the container is properly running and accessible for interaction and utilization. This step is essential for maintaining operational integrity and facilitating seamless deployment and usage of containerized applications.
- Using 'docker ps' without any flags displays only the running containers. However, appending the '-a' flag extends this functionality to show all containers, including those that have exited or been stopped. This comprehensive view can be useful for managing and troubleshooting containers across various states.
- To interact with a container running in detached mode, you can attach to it using the `docker attach` command followed by the container's name or ID. This action allows you to access the container's terminal and interact with its processes in real-time. Once attached, you can execute commands, view logs, and monitor the container's behavior. Detached mode is often used for long-running processes, and attaching to it provides a way to manage and troubleshoot the container as needed.
- Furthermore, attaching to a detached container enables you to debug issues, inspect runtime behavior, and make on-the-fly adjustments to configurations or settings. It's a valuable tool for troubleshooting complex applications or conducting live debugging sessions. Once your tasks are complete, you can detach from the container without stopping it by using the escape sequence Ctrl + P, Ctrl + Q. This flexibility empowers developers and administrators to efficiently manage containerized environments and ensure smooth operations.

4. Attaching to a Container in Detached Mode

Attaching to a container in detached mode allows you to regain interactive access to its terminal session. By utilizing the `docker attach` command followed by the container's name or ID, you can establish a direct connection to the container's processes. This enables real-time monitoring, debugging, and configuration adjustments, facilitating effective management and troubleshooting. Additionally, attaching to a container in detached mode provides an opportunity to execute commands and view logs directly within its environment, enhancing operational control and efficiency.

```
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~$ docker attach e46fb3486b4a
2
Enter a number to calculate its factorial: 4
Factorial of 4 is: 24
Choose any operation:
1. Square root
2. Factorial
3. Natural logarithm
5. Exit
Enter your choice: _
```

docker attach <container id>. The `docker attach` command is used to attach the terminal session of a running container. By specifying the container's name or ID as an argument, you can establish a direct connection to its standard input, output, and error streams. This allows you to interact with the container's processes in real-time, enabling tasks such as monitoring, debugging, and executing commands within the container's environment. Once attached, you can detach from the container without stopping it by using the escape sequence Ctrl + P, Ctrl + Q. This command is particularly useful for accessing and managing containers running in detached mode.

Testing the attached code inside the container

```
Choose any operation:
1. Square root
2. Factorial
3. Natural logarithm
5. Exit
Enter your choice: 1
Enter a number to calculate its square root: 4
Square root of 4.0 is: 2.0
Choose any operation:
1. Square root
2. Factorial
3. Natural logarithm
5. Exit
Enter your choice: 3
Enter a number to calculate its natural logarithm: 2.13
Natural logarithm of 2.13 is: 0.7561219797213337
Choose any operation:
1. Square root
2. Factorial
3. Natural logarithm
5. Exit
Enter your choice: 5
Exiting...
aakash@aakash-VivoBook-ASUSLaptop-M1603QA-M1603QA:~$ _
```

The attached code executed smoothly within the Docker container, demonstrating successful integration and functionality. This achievement underscores the seamless execution environment provided by Docker, facilitating the smooth operation of code within isolated and reproducible environments. The successful execution of the code validates its compatibility with the containerized environment, ensuring reliable and consistent performance across different deployment scenarios. This milestone signifies a crucial step forward in leveraging containerization technology to streamline development workflows and enhance the scalability and portability of software applications.

Commit History

aakash-bhardwaj-1 / SPE-MiniProject

Type to search

CodeIssuesPull requestsActionsProjectsWikiSecurityInsightsSettings

Commits

master

All usersAll time

Commits on Feb 24, 2024

Create README.md

aakash-bhardwaj-1 committed yesterday

Verifiedbfe3c1a

Commits on Feb 23, 2024

Update deploy.yml

aakash-bhardwaj-1 committed yesterday

Verifiede296d7b

Update deploy.yml

aakash-bhardwaj-1 committed yesterday

Verifiedc38c26a

Update deploy.yml

aakash-bhardwaj-1 committed yesterday

Verifiedec8f923

Update deploy.yml

aakash-bhardwaj-1 committed yesterday

Verified5d484ed

Commits on Feb 22, 2024

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verified5d455ab

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedfc6d158

Update MainTest.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedb2aef67

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verified9b1a364

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verified16f9231

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedae5f6b3

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiede3b6d63

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedd87e37b

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedf88b6dc

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifieda811c97

Update Main.java

aakash-bhardwaj-1 committed 2 days ago

Verifiedb02f6dd

The attached commit histories provide a comprehensive overview of the project's development timeline on GitHub. They detail the sequence of changes made to the codebase, including additions, modifications, and deletions across various files and directories. This detailed history offers valuable insights into the evolution of the project, highlighting key milestones, feature implementations, bug fixes, and code refactorings. Analyzing these commit histories can help track the progress of development efforts, identify patterns in coding practices, and understand the collaborative contributions of team members over time.

Conclusions

The developed DevOps toolchain assists development and operations teams in estimating implementation costs using popular DevOps tools. This calculator provides a simple and user-friendly interface for estimating potential expenses associated with deploying a toolchain utilizing these tools. By leveraging this tool, teams can make informed decisions regarding the suitable toolchain for their project within their budget constraints. This initiative showcases how these tools can seamlessly integrate to automate the software development process effectively. Integrating these tools enables developers to streamline development, diminish manual task overheads, and enhance the final product's quality. Maven serves as a robust dependency management and project building tool, while Jenkins facilitates continuous integration and delivery. Ansible, on the other hand, empowers automation for deployment and configuration management, while Docker simplifies containerization and deployment across diverse environments. Overall, this project exemplifies the significance of employing DevOps toolchains to automate software development processes. These tools aid in reducing manual task durations, improving final product quality, and fostering collaboration among various teams. Moreover, it emphasizes the importance of continuous integration and delivery, which aids in early issue detection and diminishes troubleshooting time. Embracing a DevOps methodology and utilizing these tools enable developers to optimize workflows, minimize manual tasks, and prioritize delivering top-notch software products.

Insights

During the development of this toolchain calculator, we gained several valuable insights into integrating these specific DevOps tools. Initially, we observed that Maven and Jenkins complemented each other well, facilitating a streamlined build and deployment process. Ansible proved invaluable for configuration management, automating the provisioning and configuration of necessary resources. Additionally, Docker played a pivotal role in establishing a consistent and reproducible environment for testing and deployment. Furthermore, we discovered that estimating the cost of a DevOps toolchain can be intricate due to various factors that influence the final cost. These factors include not only the costs of the tools themselves but also the expenses associated with operating and maintaining the infrastructure.

Report by,

Aakash Bhardwaj, MT2023143- M.Tech. in CSE

International Institute of Information Technology, Bangalore.

(aakash.bhardwaj@iiitb.ac.in)