Report: Scientific Calculator with DevOps

Kandarp Dave
IMT2022115

Contents

1 Introduction

1.1 Project Overview

In this project, we were required to develop a scientific calculator with user driven menu operations. I developed a terminal-based application in Java. In order to build and deliver the project, I implemented a DevOps pipeline. The following report discusses the DevOps approach along with the setup details.

GitHub repository: Calculator_SPE
DockerHub registry: calculator_spe

1.2 What is DevOps?

Software developers usually adopt a structured process to design, develop, test, deploy, and maintain software applications. This ensures that the developed software meets the user requirements, while also being high-quality. Some of the methodologies used for this are:

1. **Waterfall Model:** A sequential approach. Each phase is started only after the previous phase ends.

2. **Agile Model:** A flexible and iterative approach. The work is broken down into small phases called sprints. Development happens incrementally.

3. **DevOps Model:** Extends agile with continuous integration and delivery. Blends development and operations for faster deployments.

1.3 Why is DevOps used?

In traditional models like `Waterfall` and `Agile`, developers solely focus on writing code. The operations team is responsible for deploying and maintaining the application. But, this often causes delays and miscommunication. This is clearly not desirable while developing any application.

The aim of `DevOps` is to integrate development and operations into a single collaborative process. It is a cultural philosophy that empowers teams to work together more effectively. Frequent updates and fixes can be deployed automatically, thus reducing manual work and human errors.

Various different tools are used to implement Continuous Integration and Continuous Deployment pipelines. Testing and monitoring of applications is also automated. This results in efficient resource usage which leads to lower costs.

2 Tools Used

2.1 Source Control Management

`GitHub` was used as the source control management system. It allows multiple developers to work on the same project without interfering in each other's work. It can also connect with tools like `Jenkins`, `Docker`, `Kubernetes`, and cloud services.

2.2 Test

The `JUnit` framework was used to create unit tests for the calculator application. Unit tests are meant to test small units of code. They help to catch errors at the unit level before they spread into larger modules.

2.3 Build

`Maven` was used to run the unit tests and build the Java project. It helps to automatically build processes and manage the various dependencies of the project.

2.4 Continuous Integration

Continuous integration is the practice of merging additional code into an already existing code repository. Each integration must be tested and build. `Jenkins` was used for this purpose.

`Jenkins` can be integrated with version control systems like `GitHub` to detect changes in the repository. After this, it can run the unit tests provided and then build the application. It can also works with tools like `Docker`, `Kubernetes`, and `Ansible` for deployment.

2.5 Containerization

A container is a self-contained unit that includes the code, runtime, libraries, and system tools required to run an application. Containerizing an application allows it to be portable and run in an isolated environment. For this, `Docker` was used.

`Docker` allows the developer to provide the necessary instructions to containerize an application. Using these instructions, a docker image can be created. This image can be run by end users in the form of containers.

2.6 Deployment

`Ansible` is an open-source IT automation tool that helps you manage systems, deploy applications, and orchestrate IT infrastructure. For this project, `Ansible` was used to deploy the application.

3 Setup

3.1 Creating a Maven Project

For the project, I used the `IntelliJ IDE`. I used `Java (21)` for this project. To install it, run these commands:

```
sudo apt update && sudo apt upgrade -y
sudo apt install openjdk-21-jdk -y
java -version
```

After this, create a new `Maven` Project. Set the archetype to be `maven-archetype-quickstart`. I used the bundled `Maven`, which was version **3.9.9**. Refer to my `GitHub` repository to get `pom.xml`.

Note: Change the `mainClass` attribute in the `maven-jar-plugin` as per your main class. I used org.calculator.App as my main class.

Write your code in the `src` directory and tests in the `test` directory. Additionally, create `Jenkinsfile`, `Dockerfile`, `.dockerignore`, `inventory.ini`, and `playbook.yml` files in the **root directory** of the project.

3.2 Setting up the GitHub repository

Create a new **public** repository on GitHub. In your root directory of the maven project, run these commands:

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/your-username/your-repo.git
git branch -M main
git push -u origin main
```

This will push your maven project to GitHub.

3.3 Creating a Jenkins project

3.3.1 Installation

First, make sure that `java` is installed. After that, install `Jenkins`.

```
sudo apt update
sudo apt upgrade -y

curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update
sudo apt install jenkins -y
```

Start the `Jenkins` service.

```
sudo systemctl start jenkins
sudo systemctl status jenkins
```

You should see that `Jenkins` is active and running. Open `Jenkins` on your browser. The url is `http://localhost:8080`. Follow the instructions given. You may also create a new user if needed.

3.3.2   Exposing Jenkins using ngrok

`Jenkins` is currently running locally. In order to integrate it with `GitHub`, we must expose it so that `GitHub` can connect with it. This can be done using `ngrok`. First, sign up on ngrok.com. Then, install `ngrok` on your system.

```
wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
unzip ngrok-stable-linux-amd64.zip
sudo mv ngrok /usr/local/bin
```

Connect to your ngrok account using the `authtoken`. You can get this token from here once you sign in.

```
ngrok authtoken YOUR_NGROK_AUTH_TOKEN
```

Each ngrok user has access to a static domain. It can be found here. Go to **Jenkins > Manage Jenkins > System** and update the **Jenkins URL**.

Figure 1: Update Jenkins URL.

To expose `localhost:8080` to this static domain, run the command:

```
ngrok http --url=your-static-domain 8080
```

3.3.3   Create GitHub Webhook

On your `GitHub` profile, go to **Settings > Developer Settings > Personal access tokens > Tokens (classic)**. Follow the given images to generate a new token.
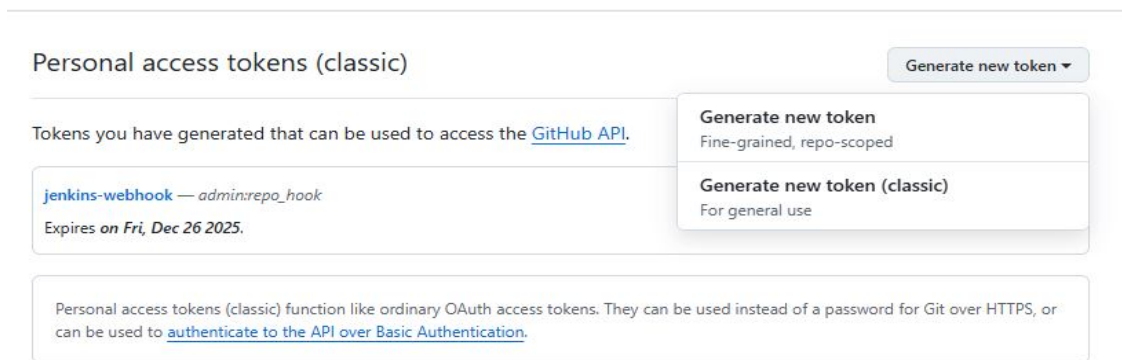
Figure 2: Generate new token (classic).

Figure 3: Set appropriate name and expiration date.

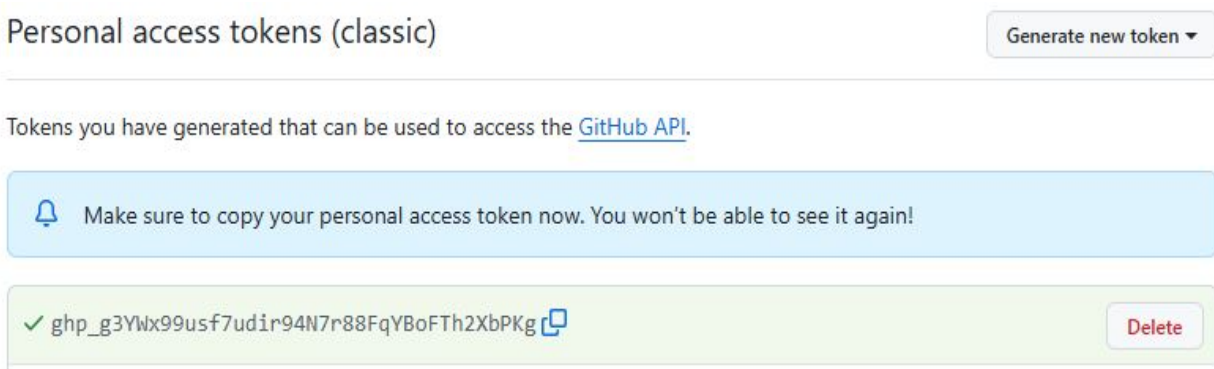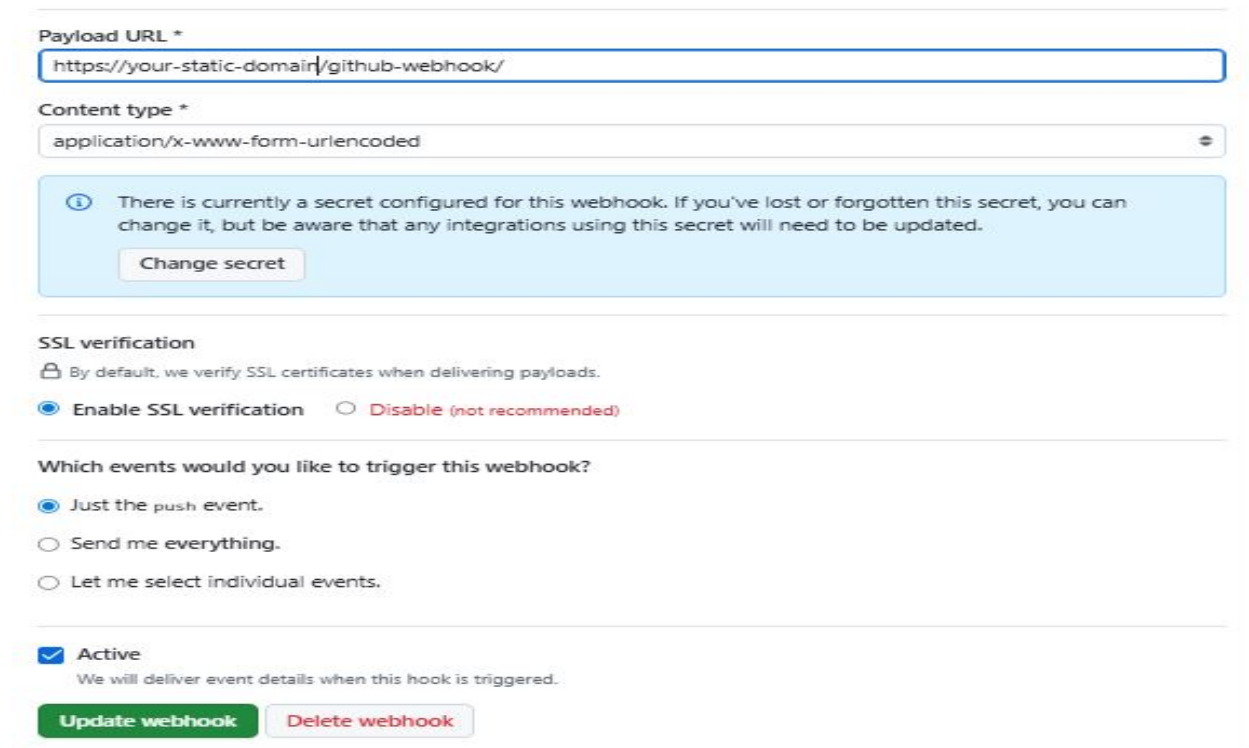Figure 4: Set appropriate control.

Figure 5: Get token. Make sure to copy it.

In your `GitHub` repository, go to **Settings** > **Webhooks** and click on `Add webhook`. Refer to the image for the creation of the webhook. Copy the token generated in the previous step here.

Figure 6: Create webhook.

Figure 7: Also setup credentials in **Jenkins** > **System**.

3.3.4 Create Jenkins Pipeline project

On the `Jenkins` Dashboard, click on `New Item` and create a **Pipeline** project. Apply these configurations:

- **GitHub project:** Provide `GitHub` url.

6

- **Triggers:** GitHub hook trigger for GITScm polling.

- **Pipeline:** Pipeline script from SCM.

  - SCM: Git.
  - Repository URL: Your `GitHub` url.
  - Credentials: <u>Add a secret text credential. The secret has to be the same as provided in the GitHub webhook.</u>
  - Branches to build: */main.
  - Script path: Jenkinsfile

3.4 Integrating Docker with Jenkins

3.4.1 Installation

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release -y

sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o \
/etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin \
docker-compose-plugin -y
```

Also sign up on DockerHub.

3.4.2 Integration with Jenkins

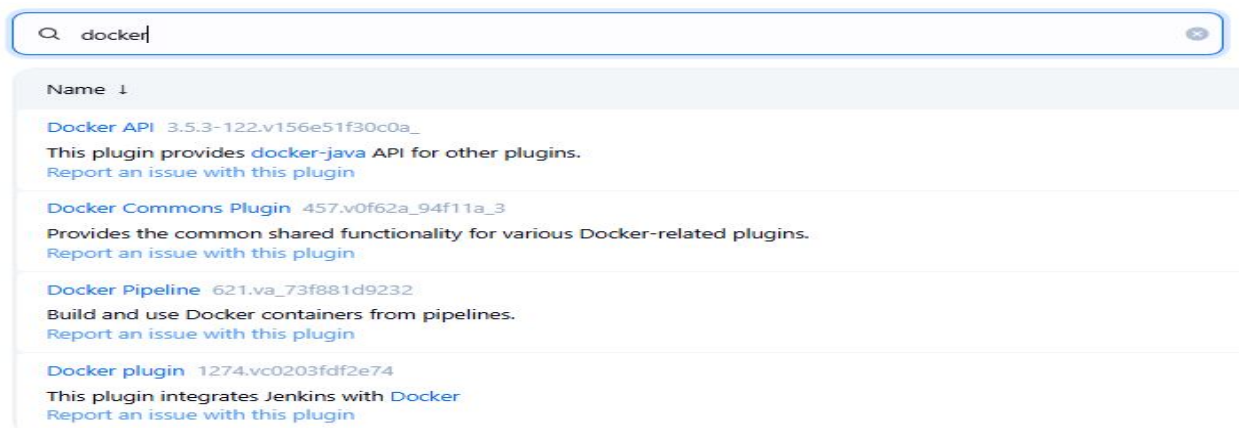In `Jenkins`, install the necessary `Docker` plugins.

Figure 8: Install `Docker` plugins.

Add `Jenkins` user to `Docker` group.

```
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```


Figure 9: Add `DockerHub` credentials in `Jenkins`. Password should be same as `DockerHub` password.

3.5   Ansible

3.5.1   Installation

```
sudo apt update
sudo apt upgrade -y

sudo apt install ansible -y
```

3.5.2   Integration with Jenkins

Make sure to give appropriate permissions to the `jenkins` user in order to run the `ansible-playbook` command.

4 Pipeline Explanation

1. **Push changes to GitHub:** Write and modify your source code. This activates the webhook, which communicates the changes to Jenkins.

2. **Jenkins pipeline execution**

   The stages of the pipeline are mentioned in `Jenkinsfile`.

   (a) <u>Run Tests:</u> Runs `JUnit` tests using `Maven`.
   (b) <u>Check Docker version:</u> An easy way to verify whether `Jenkins` is able to run the `docker` command or not.
   (c) <u>Build Docker image:</u> Builds the docker image as specified in the `Dockerfile`.
   (d) <u>Push Docker image to Registry:</u> Using the `DockerHub` credentials, pushes the docker image to `DockerHub` registry.
   (e) <u>Deploy using Ansible:</u> Pulls the docker image from `DockerHub` registry and creates a container named `calculator`.

   After the successful execution of the pipeline, the workspace is cleared.

Figure 10: Jenkins Pipeline Execution.

3. **Run the application:**

   If the `Jenkins` pipeline gets successfully executed, then a docker container named `calculator` is created on the user's system. In order to execute the application, use the command:

   ```
   docker exec -it calculator java -jar /app/app.jar
   ```

Figure 11: Execute the application.