Mini Project: Calculator_DevOps

Mayank Chadha — IMT2020045

Instructor: Prof. Thangaraju *Date: November 2, 2023*

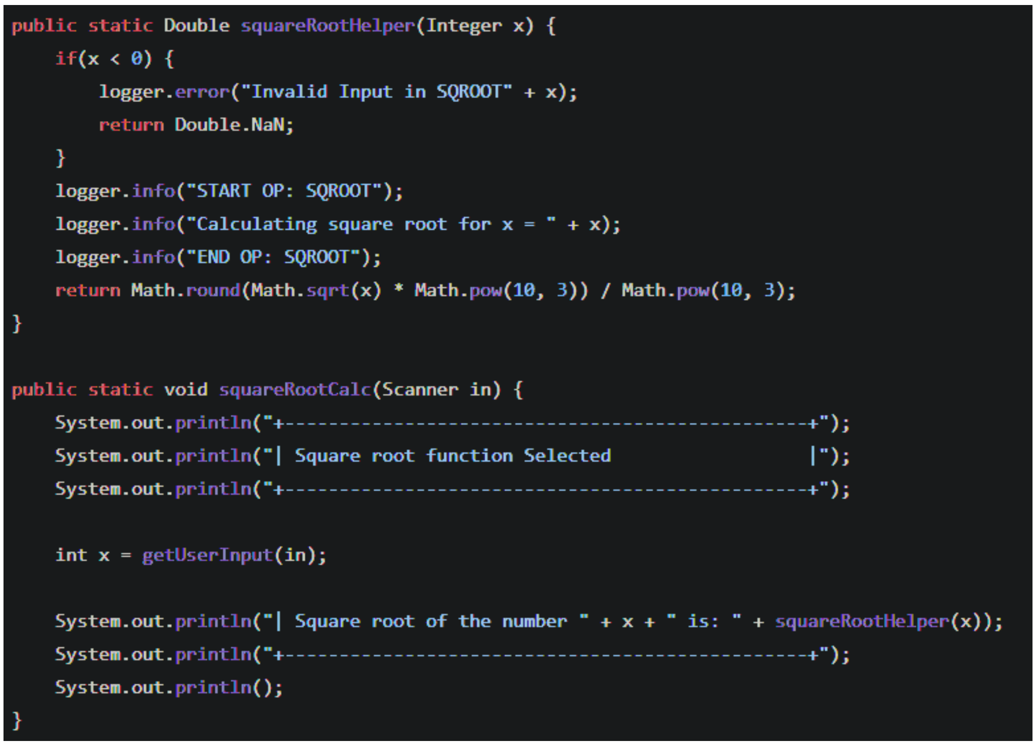
1 Explanation of the Project

The project aims to provide a hands-on experience with DevOps tools and workflows by developing a simple calculator program. It involves creating a local project, setting up version control with Git, establishing a pipeline for automated building and testing using Jenkins, containerizing the project with Docker, and utilizing Docker Hub for remote container storage. The process exemplifies the typical DevOps flow, with the development phase focusing on version control and automated builds, and the operations phase on containerization and deployment. Additionally, the project emphasizes the use of open-source tools like Git, Jenkins, Docker, Docker Hub.

2 Tools Used for the Project

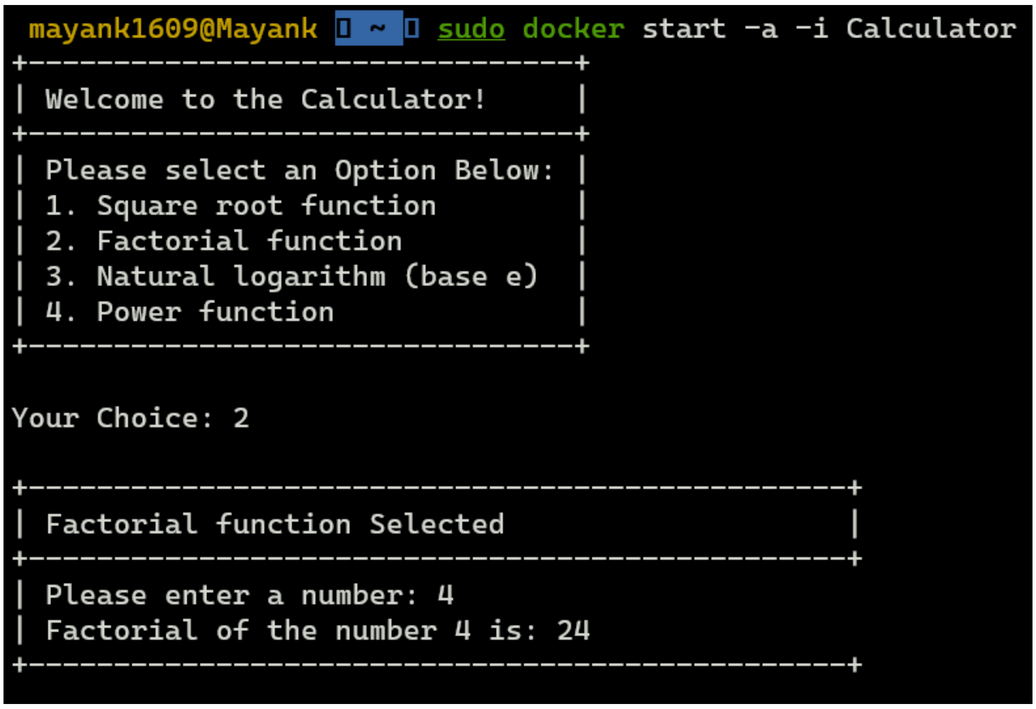
2.1 Code

- The menu-driven calculator allows users to select from a range of mathematical functions using a menu interface. The operations include calculating square roots, factorials, natural logarithms (base e), and powers.
- The calculator begins by displaying a user-friendly menu that presents a list of available mathematical functions. Users are prompted to select an option from the menu by entering a corresponding numeric choice.



```java
public static Double squareRootHelper(Integer x) {
    if(x < 0) {
        logger.error("Invalid Input in SQROOT" + x);
        return Double.NaN;
    }
    logger.info("START OP: SQROOT");
    logger.info("Calculating square root for x = " + x);
    logger.info("END OP: SQROOT");
    return Math.round(Math.sqrt(x) * Math.pow(10, 3)) / Math.pow(10, 3);
}

public static void squareRootCalc(Scanner in) {
    System.out.println("+------------------------------------------+");
    System.out.println("| Square root function Selected            |");
    System.out.println("+------------------------------------------+");

    int x = getUserInput(in);

    System.out.println("| Square root of the number " + x + " is: " + squareRootHelper(x));
    System.out.println("+------------------------------------------+");
    System.out.println();
}
```

Figure 1: Source Code from Repo



```
 mayank1609@Mayank  ~  sudo docker start -a -i Calculator
+--------------------------------+
| Welcome to the Calculator!     |
+--------------------------------+
| Please select an Option Below: |
| 1. Square root function        |
| 2. Factorial function          |
| 3. Natural logarithm (base e)  |
| 4. Power function              |
+--------------------------------+

Your Choice: 2

+-----------------------------------------------+
| Factorial function Selected                   |
+-----------------------------------------------+
| Please enter a number: 4
| Factorial of the number 4 is: 24
+-----------------------------------------------+
```

Figure 2: Calculator Functionality after everything is done

2.2 GitHub Cloning

2.2.1 Introduction to Version Control

Version control is an critical aspect of modern software development. It enables teams to efficiently collaborate on a codebase, maintain a historical record of code changes, and manage concurrent development. GitHub, one of the most widely used version control platforms, offers a range of features for effectively handling code repositories. Developers can use GitHub to create, fork, and manage repositories, collaborate with others, track issues, and maintain a well-structured codebase.

2.2.2 GitHub Cloning

GitHub Cloning is the process of creating a local copy of a project repository hosted on GitHub. This local copy is an exact replica of the remote repository, allowing developers to work on the codebase independently. By cloning a GitHub repository, you can have the project's entire history and files available on your development machine. This local copy, which can be accessed through Git commands, is essential for development, testing, and contribution. It empowers developers to make changes to the code, experiment with new features, and test code updates in a controlled environment before merging them back into the remote repository.

2.3 Maven

2.3.1 Introduction to Maven

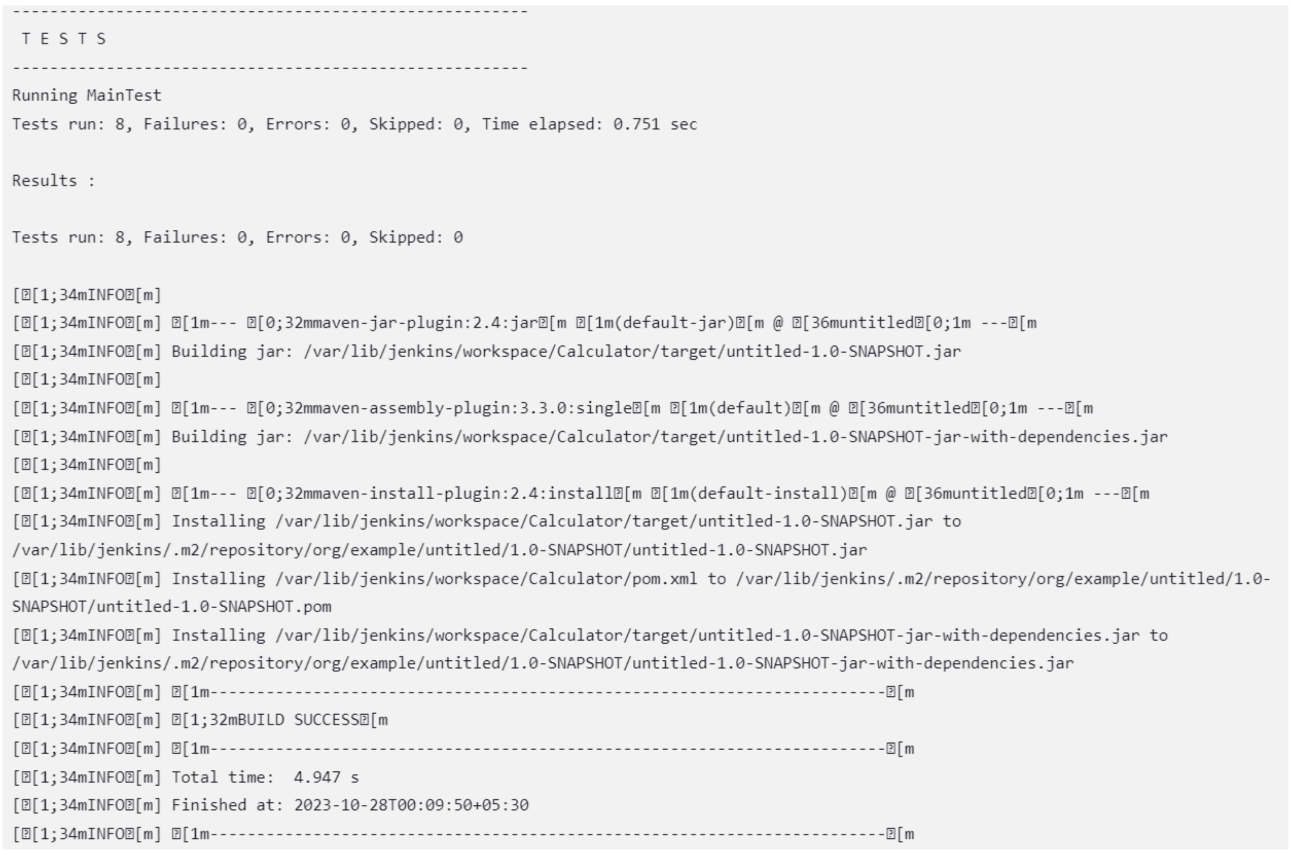
Maven is a robust build automation tool that primarily caters to Java-based projects. It serves as a fundamental pillar of the Java development ecosystem, streamlining the management of project dependencies, build processes, and project lifecycle. Maven simplifies and standardizes the project structure and offers an extensive plugin ecosystem for various tasks, such as compiling source code, running tests, generating documentation, and packaging the application. This automation reduces the likelihood of human error, ensures consistency, and enhances the development process.

2.3.2 Maven Build Lifecycle

Maven follows a well-structured build lifecycle, composed of different phases that define the order of operations during a build. It starts with the "clean" phase, which removes previously generated files, ensuring a fresh build. The "compile" phase compiles the project's source code, and the "test" phase executes unit tests. The "install" phase packages the project into JAR files, ensuring that it's ready for distribution or deployment. The lifecycle design ensures that each step is executed in the appropriate sequence, facilitating a reliable and repeatable build process.

2.3.3 Customizing the pom.xml

The `pom.xml` file in Maven projects serves as the Project Object Model (POM), containing configuration details and instructions for the build. Developers can customize the build process by modifying the pom.xml. This customization allows for setting properties, specifying build plugins, configuring dependencies, and controlling the project's structure. A common customization is providing a unique name for the generated JAR file to ensure consistency in naming conventions and avoid conflicts. Correctly specifying the Main class and matching it with the project's package is essential for executing the JAR file effectively.



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running MainTest
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.751 sec

Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

[⍰[1;34mINFO⍰[m]
[⍰[1;34mINFO⍰[m] ⍰[1m--- ⍰[0;32mmaven-jar-plugin:2.4:jar⍰[m ⍰[1m(default-jar)⍰[m @ ⍰[36muntitled⍰[0;1m ---⍰[m
[⍰[1;34mINFO⍰[m] Building jar: /var/lib/jenkins/workspace/Calculator/target/untitled-1.0-SNAPSHOT.jar
[⍰[1;34mINFO⍰[m]
[⍰[1;34mINFO⍰[m] ⍰[1m--- ⍰[0;32mmaven-assembly-plugin:3.3.0:single⍰[m ⍰[1m(default)⍰[m @ ⍰[36muntitled⍰[0;1m ---⍰[m
[⍰[1;34mINFO⍰[m] Building jar: /var/lib/jenkins/workspace/Calculator/target/untitled-1.0-SNAPSHOT-jar-with-dependencies.jar
[⍰[1;34mINFO⍰[m]
[⍰[1;34mINFO⍰[m] ⍰[1m--- ⍰[0;32mmaven-install-plugin:2.4:install⍰[m ⍰[1m(default-install)⍰[m @ ⍰[36muntitled⍰[0;1m ---⍰[m
[⍰[1;34mINFO⍰[m] Installing /var/lib/jenkins/workspace/Calculator/target/untitled-1.0-SNAPSHOT.jar to
/var/lib/jenkins/.m2/repository/org/example/untitled/1.0-SNAPSHOT/untitled-1.0-SNAPSHOT.jar
[⍰[1;34mINFO⍰[m] Installing /var/lib/jenkins/workspace/Calculator/pom.xml to /var/lib/jenkins/.m2/repository/org/example/untitled/1.0-
SNAPSHOT/untitled-1.0-SNAPSHOT.pom
[⍰[1;34mINFO⍰[m] Installing /var/lib/jenkins/workspace/Calculator/target/untitled-1.0-SNAPSHOT-jar-with-dependencies.jar to
/var/lib/jenkins/.m2/repository/org/example/untitled/1.0-SNAPSHOT/untitled-1.0-SNAPSHOT-jar-with-dependencies.jar
[⍰[1;34mINFO⍰[m] ⍰[1m------------------------------------------------------------------------⍰[m
[⍰[1;34mINFO⍰[m] ⍰[1;32mBUILD SUCCESS⍰[m
[⍰[1;34mINFO⍰[m] ⍰[1m------------------------------------------------------------------------⍰[m
[⍰[1;34mINFO⍰[m] Total time:  4.947 s
[⍰[1;34mINFO⍰[m] Finished at: 2023-10-28T00:09:50+05:30
[⍰[1;34mINFO⍰[m] ⍰[1m------------------------------------------------------------------------⍰[m
```

Figure 3: Maven install: packing the code in JAR file and running testcases

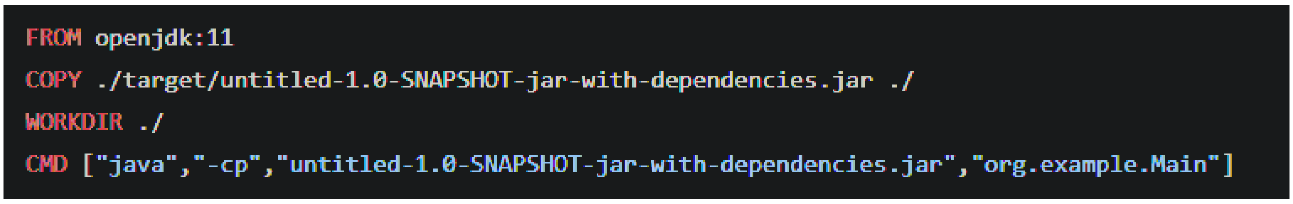
2.4 Docker

2.4.1 Introduction to Docker

Docker is a revolutionary containerization platform that revolutionizes how software is packaged, distributed, and deployed. It enables developers to encapsulate applications and their dependencies into lightweight, portable containers. These containers are isolated, ensuring consistency between development and production environments. Docker simplifies application packaging, making it possible to bundle all the required libraries, binaries, and configuration into a single unit.

2.4.2 Dockerfile

Creating a Docker container for a project starts with the creation of a Dockerfile. A Dockerfile is a text document that contains a set of instructions for building a Docker image. It defines the base image, sets up the environment, copies project files into the container, and configures any necessary settings. This file acts as a blueprint for creating the Docker image, ensuring that every container built from the same Dockerfile is identical. Dockerfiles are crucial for achieving reproducibility in deployments, as they define precisely how the application environment should be set up.

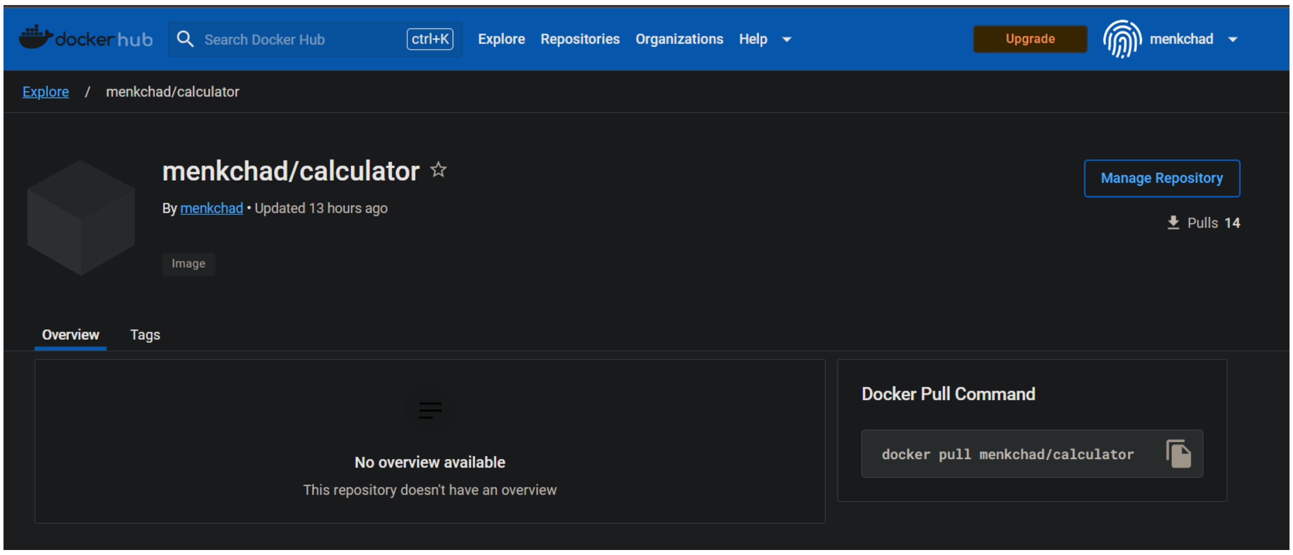


```
FROM openjdk:11
COPY ./target/untitled-1.0-SNAPSHOT-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java","-cp","untitled-1.0-SNAPSHOT-jar-with-dependencies.jar","org.example.Main"]
```

Figure 4: Dockerfile for the project

2.4.3 Building and Pushing Docker Images

To containerize a project, developers utilize the `docker build` command. This command takes a Dockerfile and a "context" as input and generates a Docker image. The Docker image is essentially a snapshot of the application and its environment. These images can be tagged for versioning and pushed to a container registry like Docker Hub for easy distribution. By pushing images to a registry, they become accessible to other team members or systems for deployment. This centralized approach simplifies the distribution and management of containerized applications.



menkchad/calculator

By menkchad • Updated 13 hours ago

Pulls 14

No overview available

This repository doesn't have an overview

Docker Pull Command

```
docker pull menkchad/calculator
```

Figure 5: Docker image pushed to Dockerhub

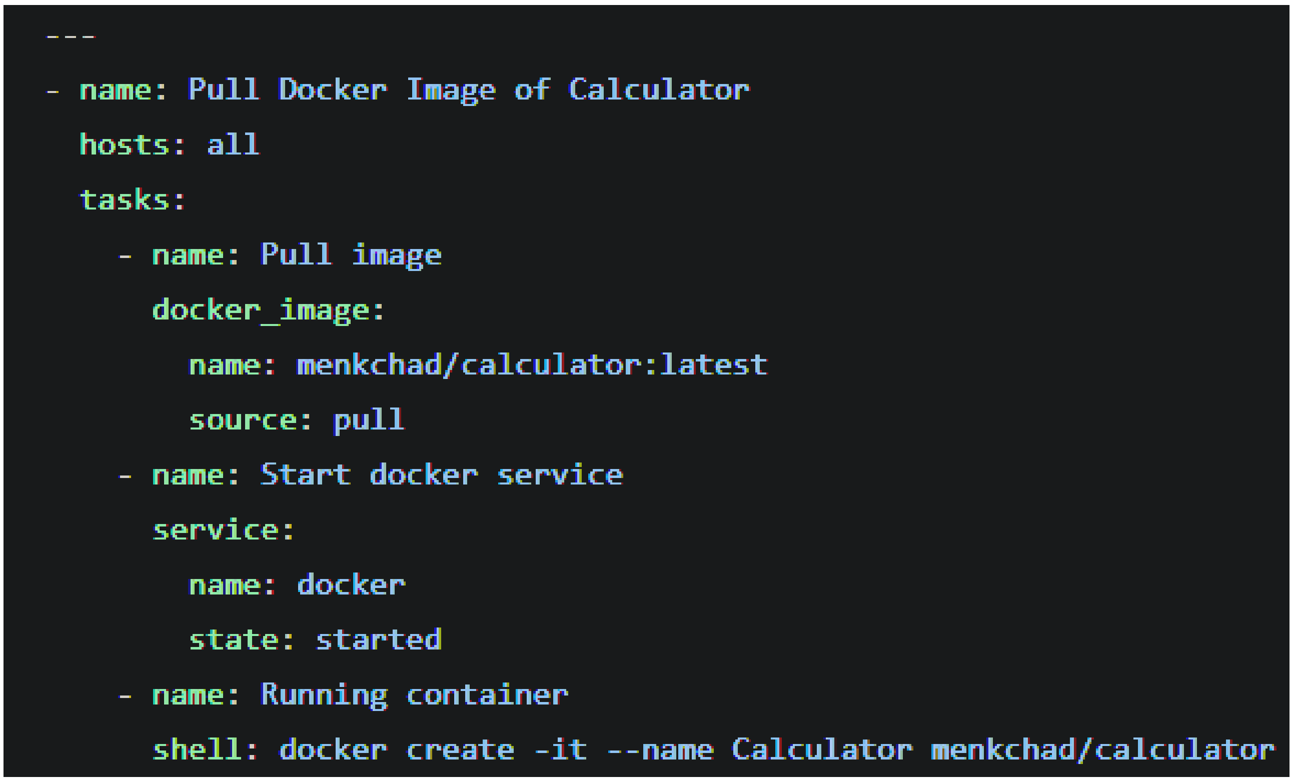
2.5 Ansible

2.5.1 Introduction to Ansible

Ansible is an automation tool that plays a pivotal role in application deployment and configuration management. It follows a declarative approach, where you define the desired state of your systems in Ansible playbooks. Ansible can automate the process of deploying Docker containers and managing their configurations, making it an invaluable asset in modern software development.

2.5.2 Deployment Process

When deploying Docker containers with Ansible, developers typically create Ansible playbooks that specify the desired state of the target systems. These playbooks may include tasks for installing Docker, configuring system settings, pulling Docker images, and launching containers. Ansible allows you to specify an inventory file, which lists the target systems where these tasks should be executed. Additionally, a deploy YAML file is used to specify the Docker image to be pulled and deployed. Mentioning the appropriate credentials is vital, particularly when deploying to your local system. Ansible ensures that these processes are automated and repeatable, reducing manual intervention and potential errors.



```yaml
---
- name: Pull Docker Image of Calculator
  hosts: all
  tasks:
    - name: Pull image
      docker_image:
        name: menkchad/calculator:latest
        source: pull
    - name: Start docker service
      service:
        name: docker
        state: started
    - name: Running container
      shell: docker create -it --name Calculator menkchad/calculator
```

Figure 6: Ansible: Our Playbook

2.6 Webhooks

2.6.1 Introduction to Webhooks

Webhooks are a powerful mechanism for external services to respond to specific events or triggers. In the context of software development, webhooks serve as the connective tissue between different parts of the development workflow. They enable developers to automate a variety of actions, from running tests to deploying code or sending notifications, based on events in a code repository or development environment.

2.6.2 Ngrok

Ngrok is a versatile tool that simplifies the process of exposing local web servers to the internet. It accomplishes this by hosting the local server on its subdomain and creating a secure tunnel to the local server. This functionality is particularly valuable for local development and testing. Ngrok allows developers to make their local development environment accessible from the internet, facilitating collaboration, testing, and integration with external services and APIs. It ensures that local development environments can be seamlessly integrated into the development and deployment process.

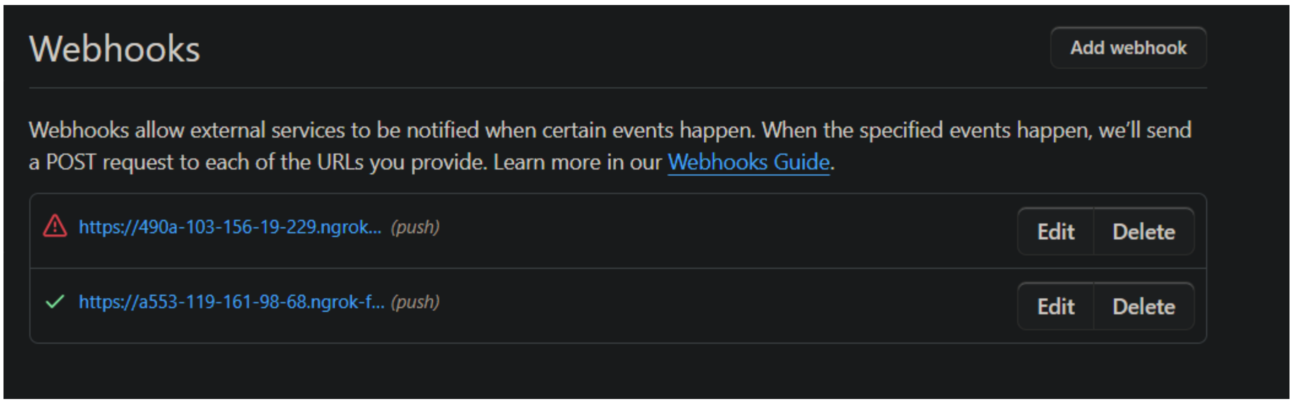
2.7 Continuous Integration with Jenkins and GitHub

2.7.1 Introduction to Continuous Integration

Continuous Integration (CI) is a development practice that encourages developers to frequently merge code changes into a shared repository. The primary goal is to automatically build and test code changes to identify errors and conflicts early in the development process. This practice ensures that the codebase is consistently in a releasable state and promotes a collaborative and efficient development environment.

2.7.2 Jenkins-GitHub Integration

Jenkins is a versatile automation server that can be integrated with GitHub to automate the build and testing process. This integration is enabled through webhooks, which allow GitHub events, such as code commits or pull requests, to trigger Jenkins jobs. Jenkins can automatically initiate builds based on code changes in a GitHub repository. It's important to distinguish between Jenkins' GitHub Plugin and the Git plugin. While the GitHub Plugin facilitates interaction between Jenkins and GitHub repositories, the Git plugin is responsible for polling and initiating builds based on changes in the Git repository. This integration ensures that code changes are rigorously tested and built, providing rapid feedback to developers. This is typically done using Webhooks.


Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

https://490a-103-156-19-229.ngrok... (push)

https://a553-119-161-98-68.ngrok-f... (push)

2.8 Continuous Deployment

2.8.1 Introduction to Continuous Deployment

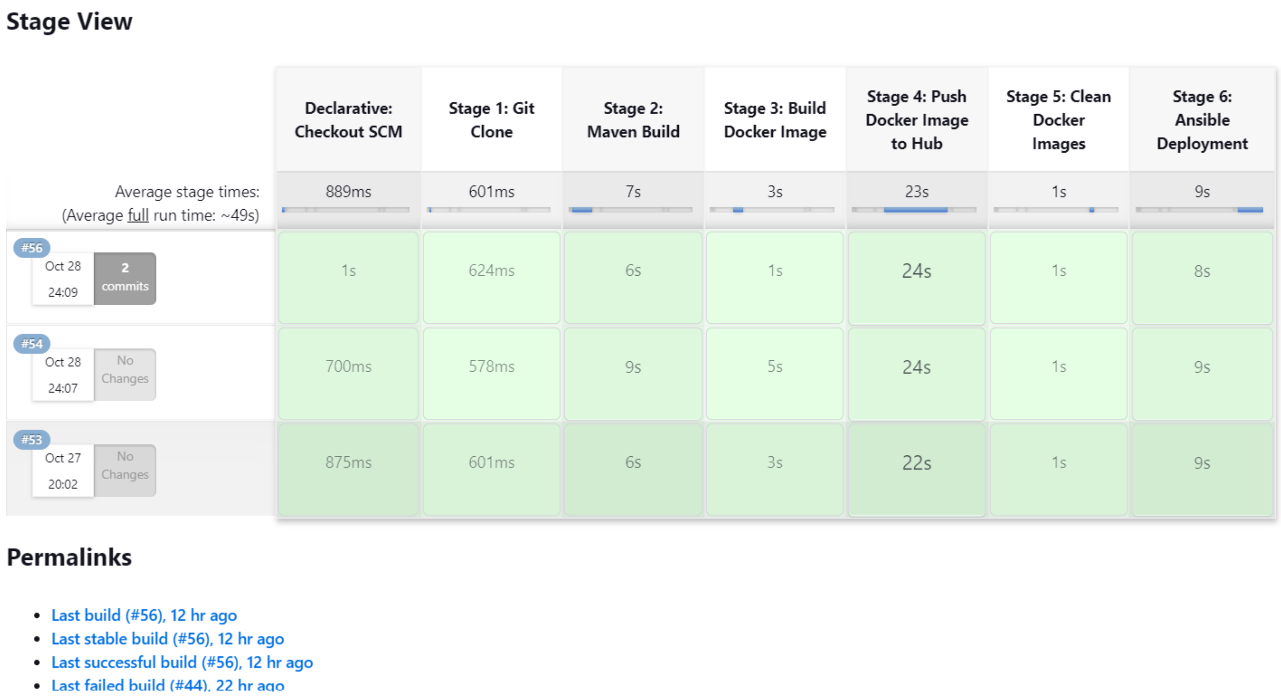
Continuous Deployment (CD) is a natural extension of Continuous Integration. It aims to automate the release and deployment of code changes to production environments. CD ensures that code changes that pass all tests and quality checks are automatically deployed to production without manual intervention. This practice provides a seamless and reliable means of delivering software to end-users.

2.8.2 Continuous Deployment Pipeline

A Continuous Deployment Pipeline is the cornerstone of the CD process. It is a set of stages and steps that automate the deployment of code changes. These stages can encompass building, testing, staging, and production deployment. A typical pipeline ensures that each stage is executed in a controlled and automated manner. This approach guarantees that code changes are deployed to production with a high degree of confidence and reliability.

2.8.3 Jenkins for Continuous Deployment

Jenkins is a powerful tool that can be configured to support Continuous Deployment. Developers set up Jenkins jobs that define how code changes are deployed to different environments, from development to production. Jenkins can automatically trigger deployment processes when code changes meet specific criteria, such as passing tests, code reviews, or approvals. Jenkins ensures that deployments are consistent, repeatable, and automated, minimizing the potential for human error.



Stage View

	Declarative: Checkout SCM	Stage 1: Git Clone	Stage 2: Maven Build	Stage 3: Build Docker Image	Stage 4: Push Docker Image to Hub	Stage 5: Clean Docker Images	Stage 6: Ansible Deployment
Average stage times: (Average full run time: ~49s)	889ms	601ms	7s	3s	23s	1s	9s
#56 Oct 28 24:09 — 2 commits	1s	624ms	6s	1s	24s	1s	8s
#54 Oct 28 24:07 — No Changes	700ms	578ms	9s	5s	24s	1s	9s
#53 Oct 27 20:02 — No Changes	875ms	601ms	6s	3s	22s	1s	9s

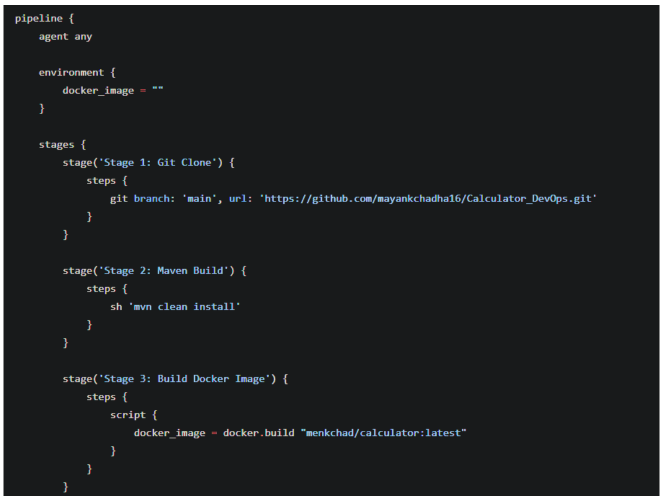
Permalinks

- Last build (#56), 12 hr ago
- Last stable build (#56), 12 hr ago
- Last successful build (#56), 12 hr ago
- Last failed build (#44), 22 hr ago

Figure 7: CI/CD using Jenkins

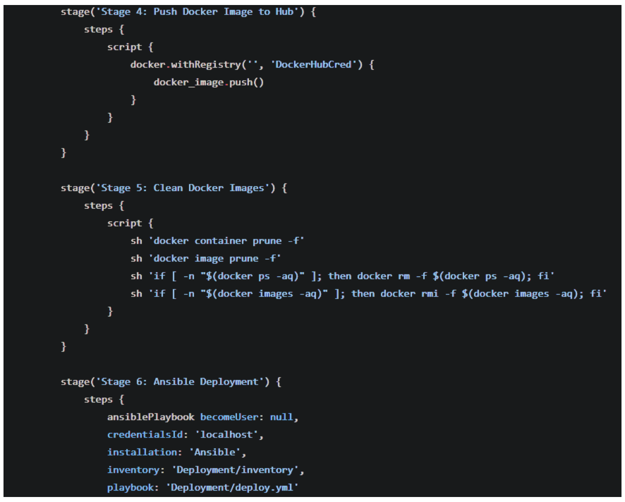
3 Steps to achieve the end result

To achieve CI/CD, you set up these processes, tools, and integrations to automate the build, test, and deployment cycle of your software. This ensures that code changes are consistently built, tested, and deployed, reducing manual intervention and increasing the reliability and speed of software development and release. The specific details of how you configure these tools and processes would depend on your project's requirements and the technologies you're using.



```
pipeline {
    agent any

    environment {
        docker_image = ""
    }

    stages {
        stage('Stage 1: Git Clone') {
            steps {
                git branch: 'main', url: 'https://github.com/mayankchadha16/Calculator_DevOps.git'
            }
        }

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

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

Pipeline Script - 1



```
        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'
                    sh 'if [ -n "$(docker ps -aq)" ]; then docker rm -f $(docker ps -aq); fi'
                    sh 'if [ -n "$(docker images -aq)" ]; then docker rmi -f $(docker images -aq); fi'
                }
            }
        }

        stage('Stage 6: Ansible Deployment') {
            steps {
                ansiblePlaybook becomeUser: null,
                credentialsId: 'localhost',
                installation: 'Ansible',
                inventory: 'Deployment/inventory',
                playbook: 'Deployment/deploy.yml'
```

Pipeline Script - 2

3.1 How do we run the Project?

To execute the project, follow these steps:

3.1.1 Setting up ngrok

1. Start by setting up ngrok for exposing your local web service. Run the following command:

   ```
   ngrok http 8080
   ```

2. Next, configure a webhook in your GitHub repository. You can find your repository at the following URL: https://github.com/mayankchadha16/Calculator_DevOps.
3. Update the Jenkins configuration by navigating to `System -> Jenkins Location`. Replace the Jenkins URL with the forwarding address provided by ngrok, which you used for setting up the webhook.

3.1.2 Building the Project

1. Initiate the build process in Jenkins by clicking on the "Build Now" button. This action will trigger the pipeline to execute.
2. Alternatively, if you have configured Git SCM polling, the project will automatically build upon new commits to the repository.

3.1.3 Docker Container

1. Once the build process is completed, a Docker image is created. You can view a list of Docker images by running the following command:

   ```
   docker images
   ```

2. Similarly, a Docker container is also created. You can list all Docker containers, including the running and stopped ones, using this command:

   ```
   docker ps -a
   ```

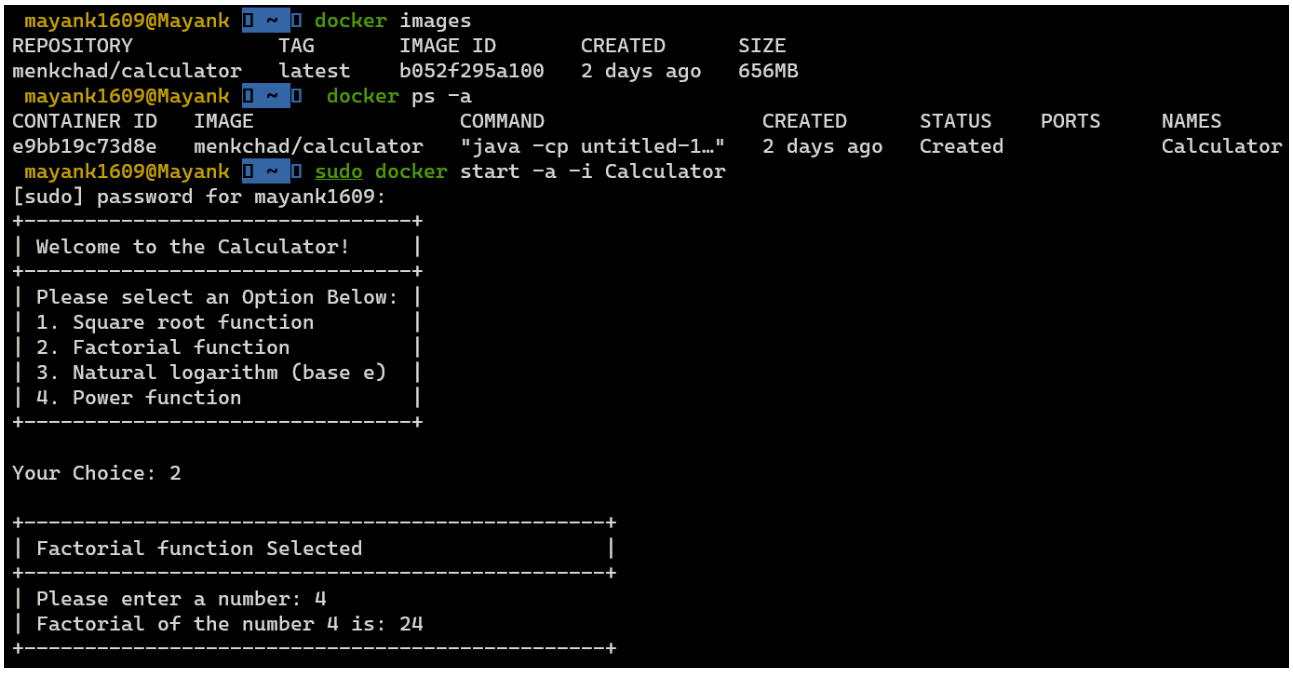
3. To access the Docker container, start it with the following command, allowing for interaction and keeping the standard input open:

   ```
   sudo docker start -a -i Calculator
   ```

   (Note: The '-a' option enables interaction with the container so you can see the container's output and provide input as if you were using a terminal within the container., while '-i' keeps the standard input open, even if not attached.)
4. Optionally, to access the container's log files, you can open the container like a file directory with the following commands:

   ```
   sudo docker start Calculator
   sudo docker exec -it Calculator /bin/bash
   ```

   (-it flags are used for interactive sessions. '-i' makes the session interactive, and '-t' allocates a pseudo-TTY for terminal interaction.)



```
 mayank1609@Mayank  ~  docker images
REPOSITORY            TAG       IMAGE ID       CREATED      SIZE
menkchad/calculator   latest    b052f295a100   2 days ago   656MB
 mayank1609@Mayank  ~   docker ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED      STATUS    PORTS     NAMES
e9bb19c73d8e   menkchad/calculator   "java -cp untitled-1…"   2 days ago   Created             Calculator
 mayank1609@Mayank  ~  sudo docker start -a -i Calculator
[sudo] password for mayank1609:
+---------------------------------+
| Welcome to the Calculator!      |
+---------------------------------+
| Please select an Option Below:  |
| 1. Square root function         |
| 2. Factorial function           |
| 3. Natural logarithm (base e)   |
| 4. Power function               |
+---------------------------------+

Your Choice: 2

+--------------------------------------------------+
| Factorial function Selected                      |
+--------------------------------------------------+
| Please enter a number: 4
| Factorial of the number 4 is: 24
+--------------------------------------------------+
```

Figure 8: Running the Container

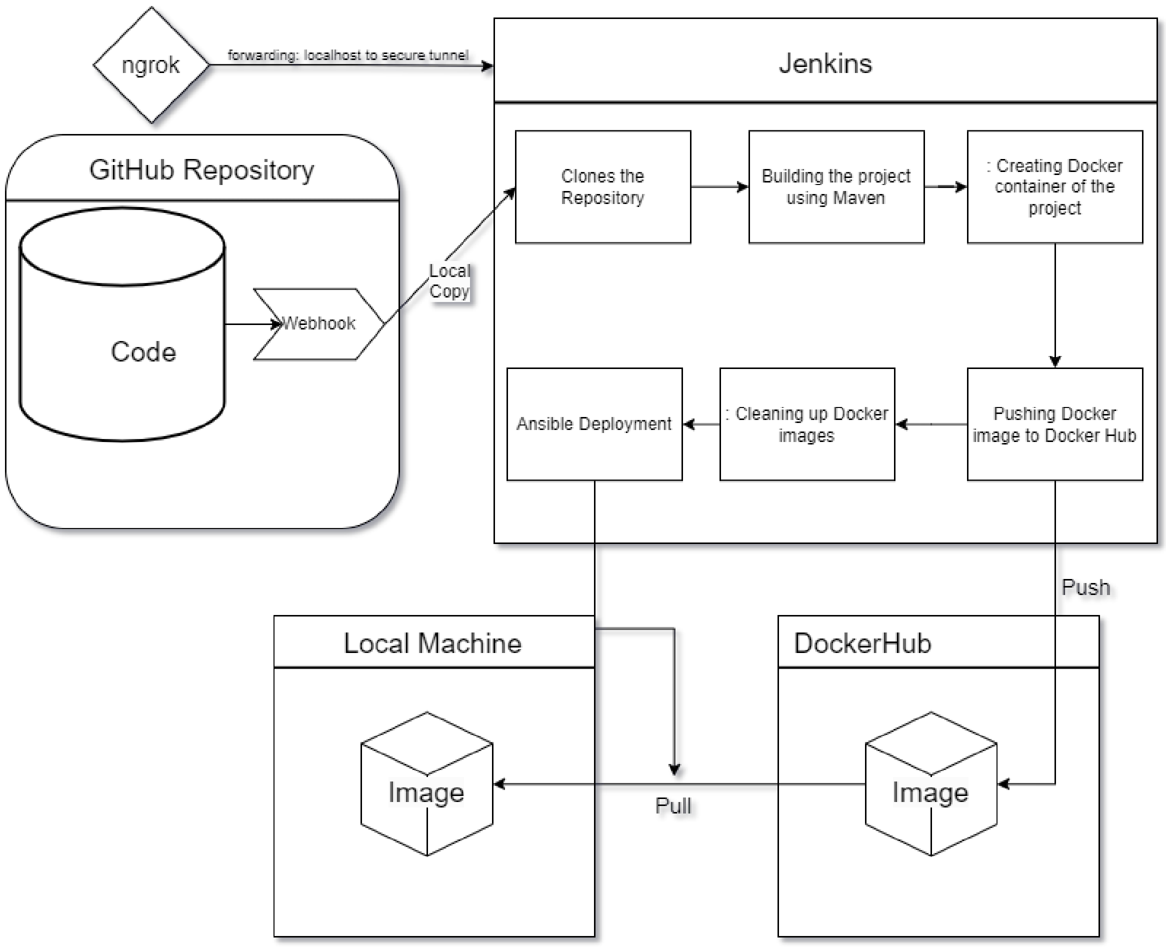

ngrok
forwarding: localhost to secure tunnel
Jenkins
GitHub Repository
Code
Webhook
Local Copy
Clones the Repository
Building the project using Maven
: Creating Docker container of the project
Pushing Docker image to Docker Hub
: Cleaning up Docker images
Ansible Deployment
Push
Local Machine
DockerHub
Image
Pull
Image

Figure 9: Steps took to achieve the end result in a nutshell

4 Links:

GitHub Repository

DockerHub Repository