CSoftware Production Engineering September 26, 2024

SPE Project

Instructor: Prof. Thangaraju Goutham U R (IMT2021045)

1 Introduction

The focus of this project is to develop a simple scientific calculator program while emphasizing the use of DevOps tools and workflows. The primary goal is not only to create a functional application but to leverage modern DevOps practices for efficient development, deployment, and monitoring of the program. This involves integrating continuous integration (CI) and continuous deployment (CD) pipelines, containerization, and automated deployment processes, which are essential components of the DevOps ecosystem.

The DevOps approach enhances collaboration between development and operations teams, ensuring quicker release cycles and reliable performance monitoring. Throughout this project, tools like Jenkins, Docker, and Git will be used to automate workflows and streamline project delivery.

2 Objectives

The objectives of this project are as follows:

1. **Local Project Creation**: To initiate the development by setting up a local project within an Integrated Development Environment (IDE).
2. **Version Control**: To establish a local repository using a version control system, enabling efficient tracking of code changes.
3. **Remote Repository Integration**: To push the local repository to a remote version control platform, enabling collaboration and centralized code management.
4. **CI Pipeline Setup**: To create a CI pipeline that automatically builds the project and runs tests whenever changes are pushed to the repository, ensuring early detection of issues.
5. **Containerization**: To automate the process of containerizing the project and pushing the container image to a remote repository for portability and consistency.
6. **Automated Deployment**: To deploy the containerized application to a target machine using Jenkins, ensuring streamlined and consistent deployment across environments.

3 Tools Used

The following tools and technologies were employed throughout the development and deployment phases of this project:

- **Development Environment**: IntelliJ IDEA Ultimate Edition was used for coding, debugging, and testing the scientific calculator application.
- **Version Control**: Git served as the version control system to manage code changes and maintain a history of project development.
- **Remote Repository Hosting**: GitHub provided a remote platform for hosting the project repository, facilitating collaboration and backup.
- **Build Automation**: Apache Maven was used to automate the build process, ensuring consistency and simplifying dependency management.

- **Continuous Integration/Continuous Deployment (CI/CD)**: Jenkins was utilized to automate the build, test, and deployment process, enabling continuous integration and continuous delivery.
- **Containerization Platform**: Docker was used to containerize the application, ensuring portability across different environments and systems.
- **Configuration Management**: Ansible was employed for automated configuration management and deployment, simplifying the orchestration of system setups.
- **Git SCM Polling and Automation**: Ngrok and GitHub Webhooks were configured to automate Git SCM polling, allowing Jenkins to detect changes in the repository and trigger automated tasks.

4 Links

Below are the links to the key resources used in this project:

- GitHub Repository: The source code and project files are hosted on GitHub, enabling version control and collaboration.
- Docker: Docker was used for containerizing the application to ensure environment consistency and portability.

5 Installation Pre-Requisites

5.1 Updating Package Sources

```
sudo apt-get update
sudo apt-get upgrade
```

5.2 Installing Git

```
sudo apt install git
git --version
```

5.3 Installing and Initialising Jenkins

```
1. curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key |
sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
2. 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
3. sudo apt install ca-certificates
4. sudo apt update
5. sudo apt install jenkins
6. sudo systemctl start jenkins
7. sudo systemctl status jenkins
8. sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

5.4 Installing and Initialising Docker

```
sudo apt install curl
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
sudo docker --version
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
```

5.5 Installing and Initialising Ansible

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

5.6 Installing ngrok

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

6 Implementation

6.1 Creating the Project in IntelliJ

To create a new project in IntelliJ, follow these steps:

1. Open IntelliJ IDEA and click on **New Project**.
2. In the pop-up window, name your project **Calculator**.
3. Select **Maven** as your build tool.
4. In the JDK dropdown, select the default Java version.

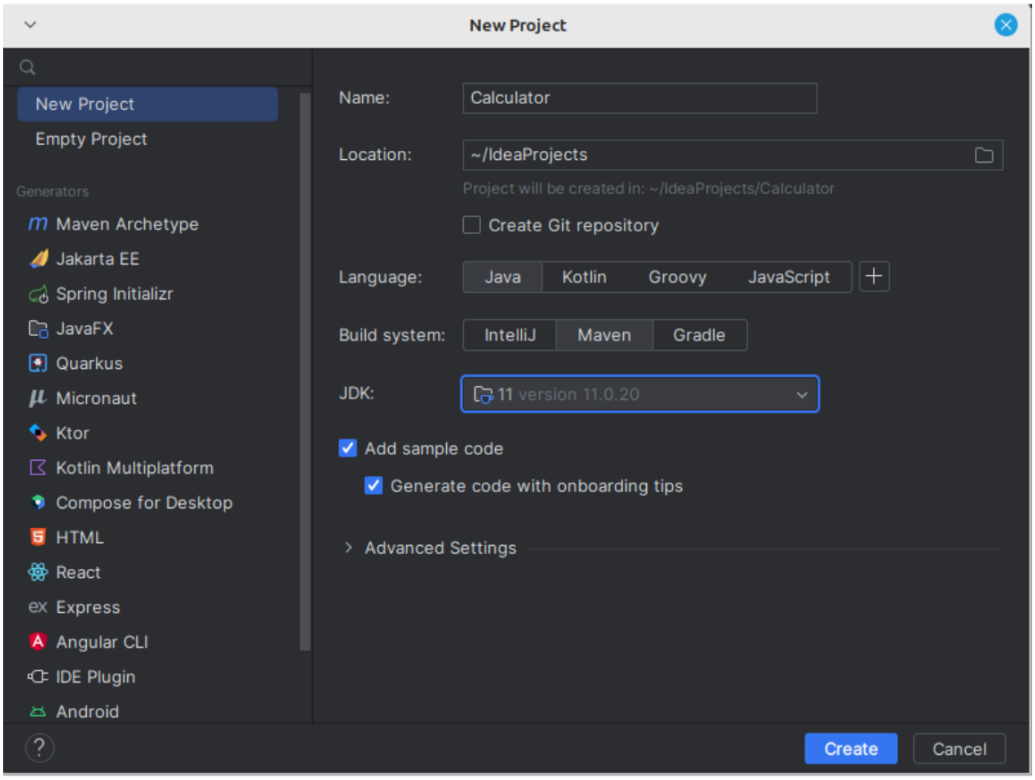

Figure 1: Creating a new project in IntelliJ IDEA.

6.2 Running Maven Commands

To install Maven on your system, use the following command:

```
sudo apt install maven
```

Maven is a build automation tool that is essential for managing dependencies and building Java projects. Installing Maven is the first step before you can compile and install your project.

After installing Maven, navigate to your project directory and run the following commands:

```
mvn clean
mvn compile
mvn install
```

- **mvn clean**: This command removes the `target` folder, ensuring that the next compilation uses the latest files.
- **mvn compile**: This compiles the project and runs the test cases to check for errors.
- **mvn install**: This command generates the `JAR` file for the project, making it ready for deployment or further usage.

The clean, compile, and install process is essential for maintaining a consistent build process and ensuring that the project is free of errors before deployment.

6.3 Modifying the pom.xml File for Custom JAR Naming

When Maven generates a JAR file, it typically defaults to a name that includes the project name along with the keyword "SNAPSHOT." However, to ensure consistency and allow for easy deployment, we will modify the 'pom.xml' file to generate a JAR with a custom name.

By editing the 'pom.xml' file and adding specific XML tags within the 'project .../project ' section, we can control the exact name of the output JAR file. These modifications are essential to ensure that the JAR file remains the same across different builds, which simplifies the deployment process.

After making the necessary changes, the 'pom.xml' file will look somewhat like the following:

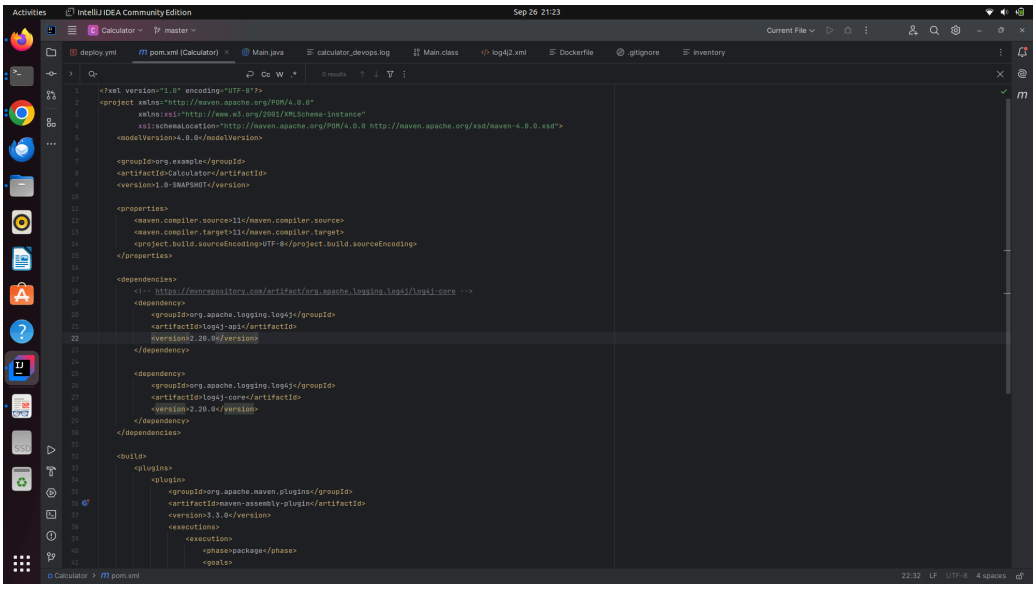

Figure 2: Sample 'pom.xml' file after modifications for custom JAR naming.

In this example, the 'build ' and 'finalName ' tags have been added to ensure that the output JAR has a constant name.

Additionally, further configurations can be made as shown below:

Make sure that these changes are saved in the 'pom.xml' file before proceeding with the build.

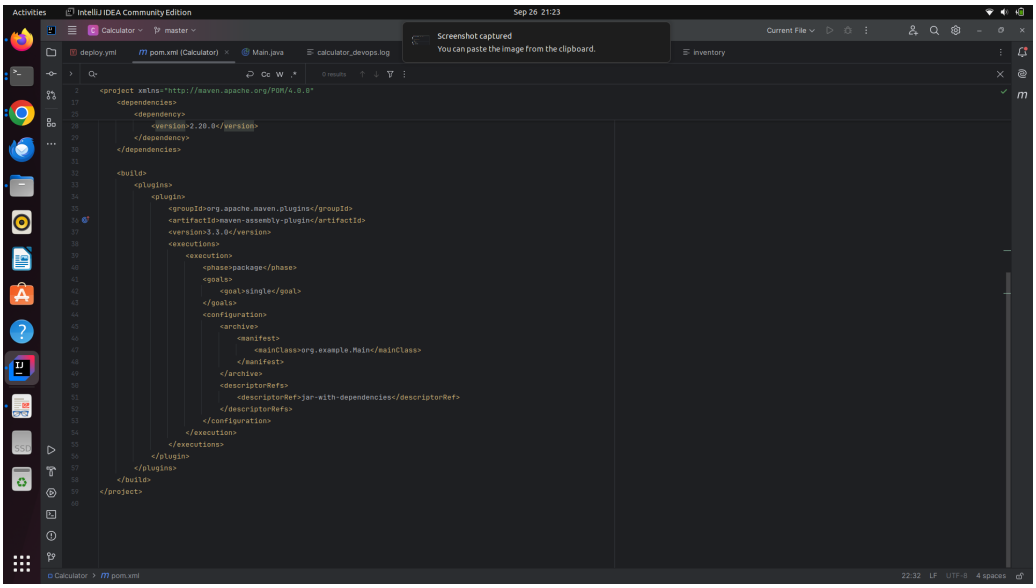

Figure 3: Additional configuration in the ‘pom.xml‘ file.

6.4 Creating a Remote GitHub Repository and Connecting with Local Git

Once we have our project set up locally, the next step is to create a remote repository on GitHub and connect it to the local repository. This allows us to push our local changes to a central repository for version control and collaboration.

Follow these steps:

1. First, create a new repository on GitHub. Navigate to GitHub, log into your account, and select the `New` repository button.
2. Name your repository `Calculator`, choose whether it should be public or private, and then create the repository. GitHub will provide you with a remote URL that you'll use to connect your local repository.
3. Next, initialize Git in your local project directory (if it hasn't been initialized already):

   ```
   git init
   ```

4. Add all project files to the staging area:

   ```
   git add .
   ```

5. Commit your changes:

   ```
   git commit -m "Initial commit"
   ```

6. Now, add the GitHub repository as the remote origin:

   ```
   git remote add origin https://github.com/your-username/Calculator.git
   ```

7. Push your local changes to the remote GitHub repository:

   ```
   git push -u origin master
   ```

After completing these steps, your local Git repository will be linked to your remote GitHub repository. Any further changes can be committed locally and pushed to GitHub using the same workflow.

Using Personal Access Tokens for GitHub Authentication

In the event GitHub requires you to authenticate, especially after August 2021, GitHub has replaced password authentication for Git operations with **Personal Access Tokens (PAT)**.

1. To generate a Personal Access Token, go to GitHub and navigate to `Settings > Developer Settings > Personal Access Tokens`.
2. Click on `Generate New Token`, and specify the scope for the token (e.g., access to repositories). For basic use, select `repo`.
3. Copy the generated token and store it securely. Once generated, you won't be able to view it again.
4. Now, when you push changes to GitHub, you'll be prompted for authentication. Instead of entering your GitHub password, use the token as the password:

   ```
   git push -u origin master
   ```

   During this step, you will be prompted to enter your GitHub username and the newly generated token as the password.

Using tokens enhances security and ensures that your credentials remain protected during interactions with GitHub.

7 Initial Jenkins Configuration

1. **Login and Dashboard Access**: After logging in, navigate to the dashboard and click on "New Item" to create a new pipeline.
2. **Pipeline Creation**: Name the new item "Calculator" and select the "Pipeline" option, then click "OK" to proceed.
3. **Pipeline Configuration**: On the configuration screen, save the default settings to access the pipeline dashboard.
4. **Add Pipeline Syntax**: Return to the pipeline dashboard to add the pipeline syntax for each stage, noting that there will be unmet dependencies.
5. **Manage Plugins**: Go to Dashboard ¿ Manage Jenkins ¿ Manage Plugins to resolve unmet dependencies by installing required plugins.
6. **Necessary Plugins**: Search for and ensure the following plugins are installed: Ansible, Docker, Docker Pipeline, Git Client, Git plugin, GitHub, and JUnit.
7. **Installation Completion**: If any required plugins are not installed, proceed to install them before continuing with pipeline setup.

8 Jenkins Pipeline Overview

8.0.1 1. Cloning Project from GitHub

Create a new pipeline in Jenkins. Add the first stage to clone the GitHub repository.

```
pipeline {
    agent any
    stages {
        stage('Git Clone') {
            steps {
                git branch: 'master', url: 'https://github.com/your-repo/Calculator.git'
            }
        }
    }
}
```

8.0.2 2. Building the Project with Maven

Add a stage to clean, compile, and install using Maven.

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

8.0.3 3. Creating a Docker Image

Add a Dockerfile to your project and use it to build the Docker image.

```
stage('Build Docker Image') {
    steps {
        script {
            docker_image = docker.build "your-dockerhub-username/calculator:latest"
        }
    }
}
```

8.0.4 4. Pushing Docker Image to Docker Hub

Push the Docker image to your Docker Hub repository.

```
stage('Push Docker Image') {
    steps {
        script {
            docker.withRegistry('', 'DockerHubCred') {
                docker_image.push()
            }
        }
    }
}
```

8.0.5 5. Cleaning Up Docker Images

Remove unused Docker images and containers.

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

8.0.6 6. Deploying with Ansible

Add an Ansible playbook to pull and deploy the Docker image.

```
stage('Ansible Deployment') {
    steps {
        ansiblePlaybook inventory: 'Deployment/inventory', playbook: 'Deployment/deploy.yml'
    }
}
```

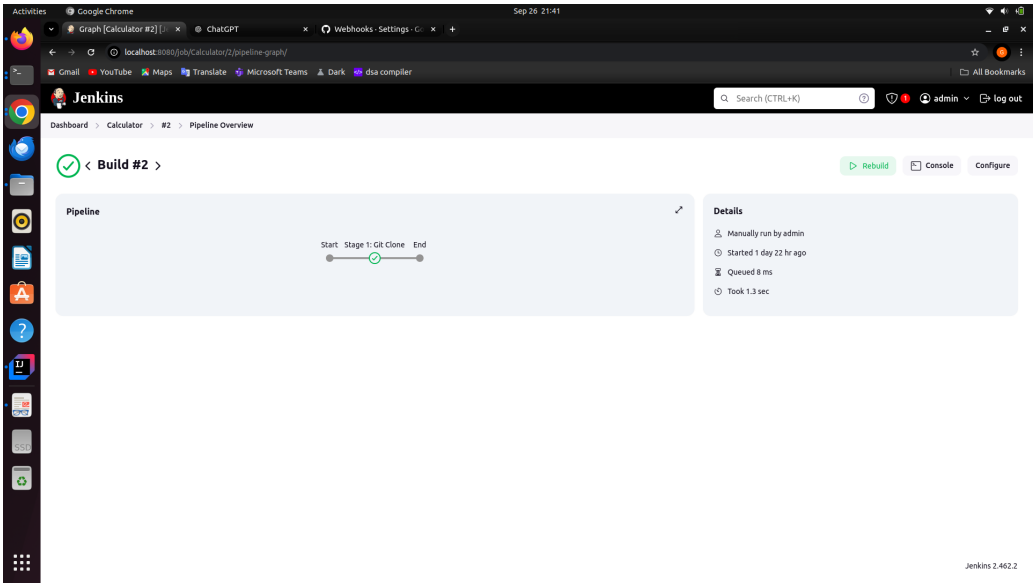

Figure 4: Cloning the GitHub repository

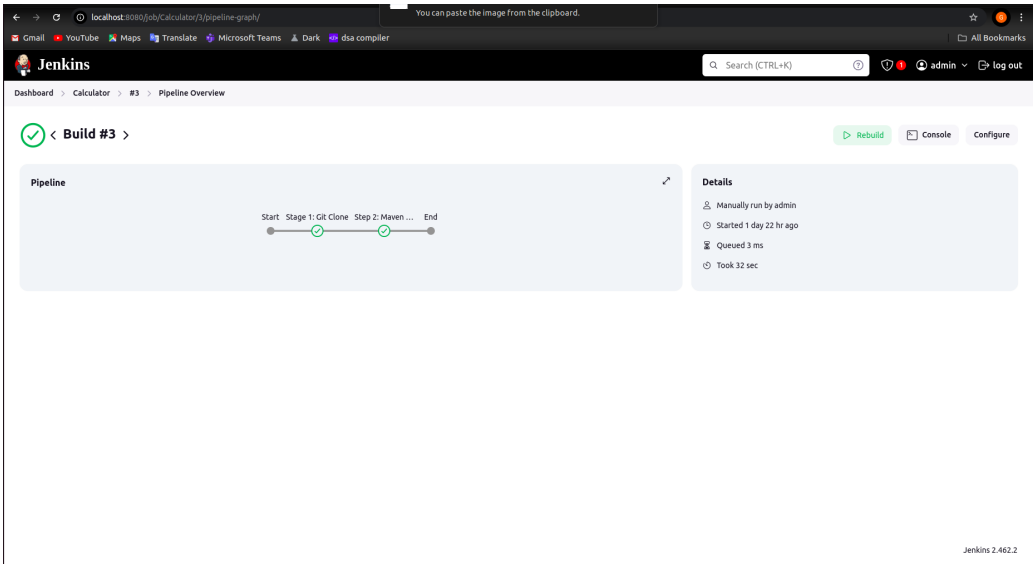

Figure 5: Building the project with Maven

9 Running a container

To see the list of containers, we can use the following command:

```
sudo docker container ls --all
```

We can see that the container status is `Exited`. To run the container, we use the following command:

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

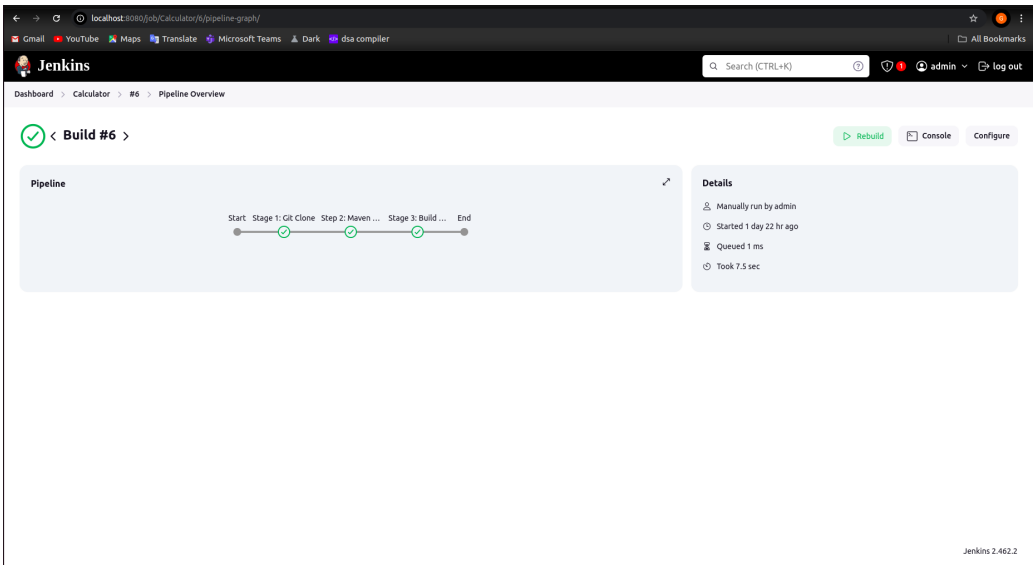

Figure 6: Creating a Docker image

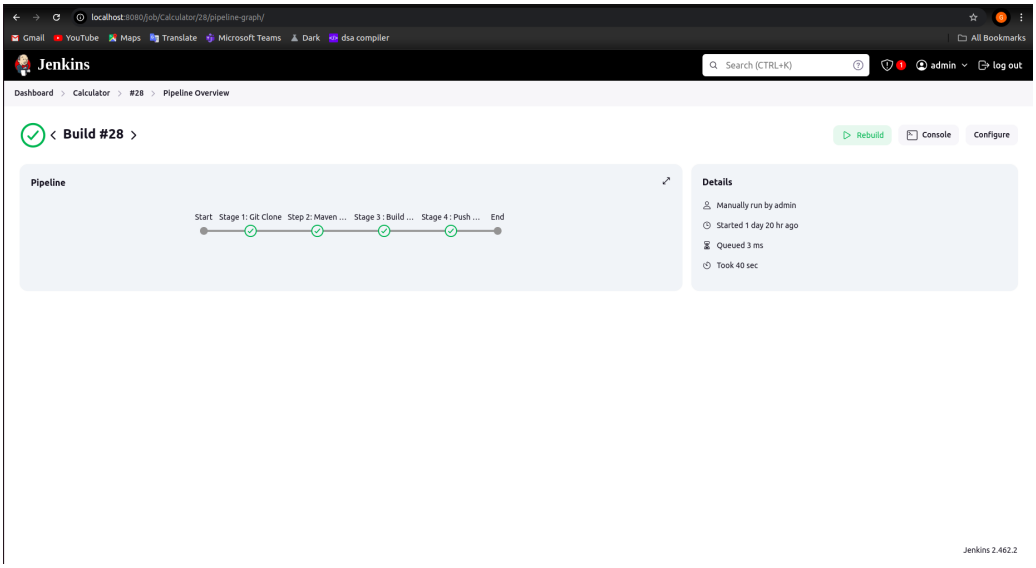

Figure 7: Pushing Docker image to Docker Hub

10 Ngrok Integration

Now the last step is to integrate with ngrok. We start by running ngrok on port 8080 with the following command:

```
ngrok http 8080
```

The forwarding URL is `https://c551-103-156-19-229.ngrok-free.app`. Note that this will change every time we close and launch ngrok (since this is the free version). So, make sure not to close this for now.

Next, we go to our project repository in GitHub ¿ Settings ¿ Webhooks option of our repository. In the webhooks section, we click on "Add webhook" and then specify the following details:

- **Payload URL**: `<ngrok forwarding url>/github-webhook/`

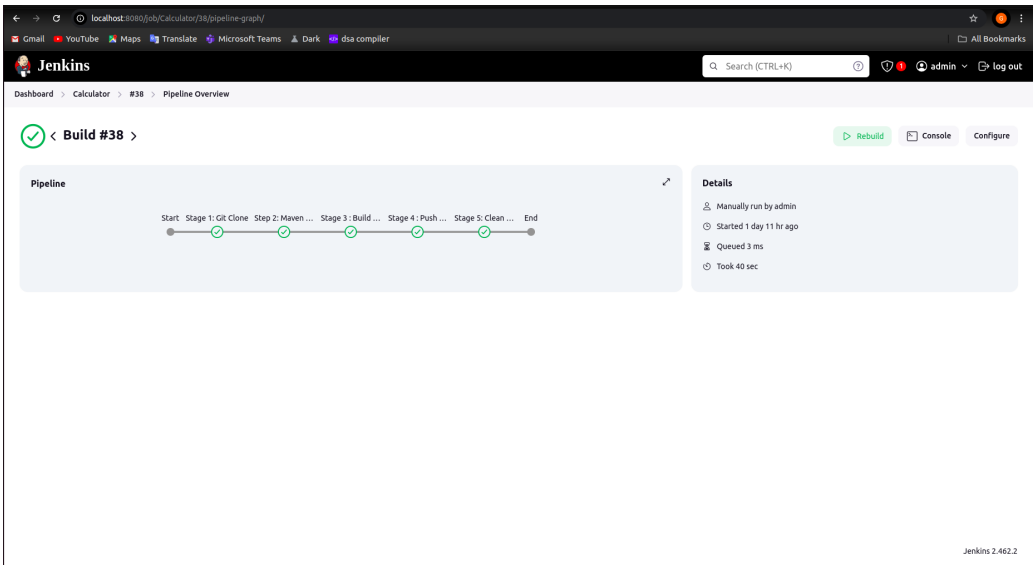

Figure 8: Cleaning up Docker images

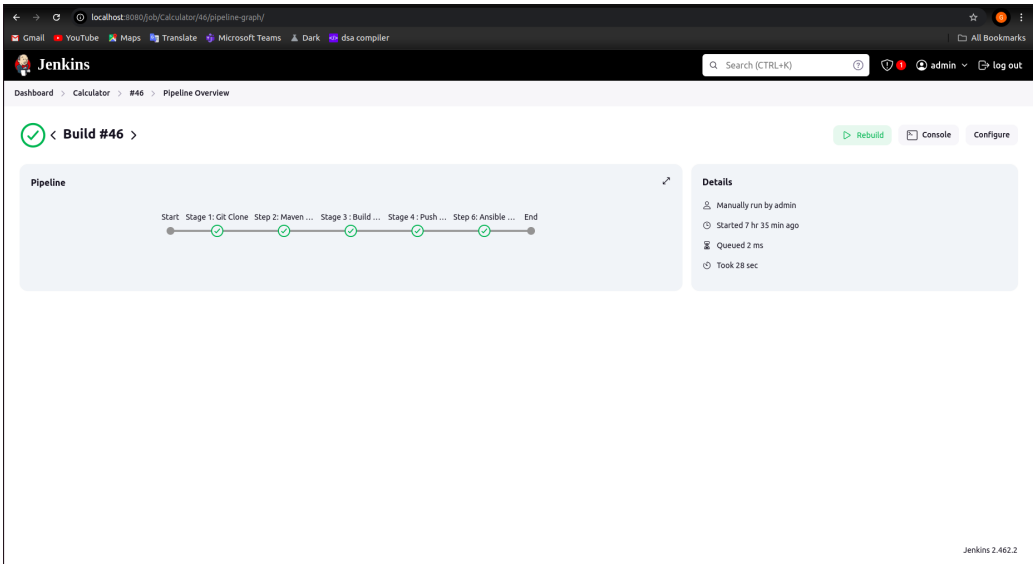

Figure 9: Deploying with Ansible

11 Special Files in the Code

1. **DockerFile**
   - Defines a Docker image for the Calculator application, specifying the base image as OpenJDK 11.
   - Copies the compiled JAR file (`Calculator-1.0-SNAPSHOT-jar-with-dependencies.jar`) into the Docker image.
   - Sets the working directory and specifies the command to run the main class of the application when the container starts.
2. **Log4j2.xml**
   - Configuration file for Log4j 2, specifying how logging should be handled in the application.
   - Defines appenders for console and file outputs, with a specific pattern layout for formatting log messages.
   - Sets the root logger level to debug and specifies that log messages should be written to `calculator_devops.log`.

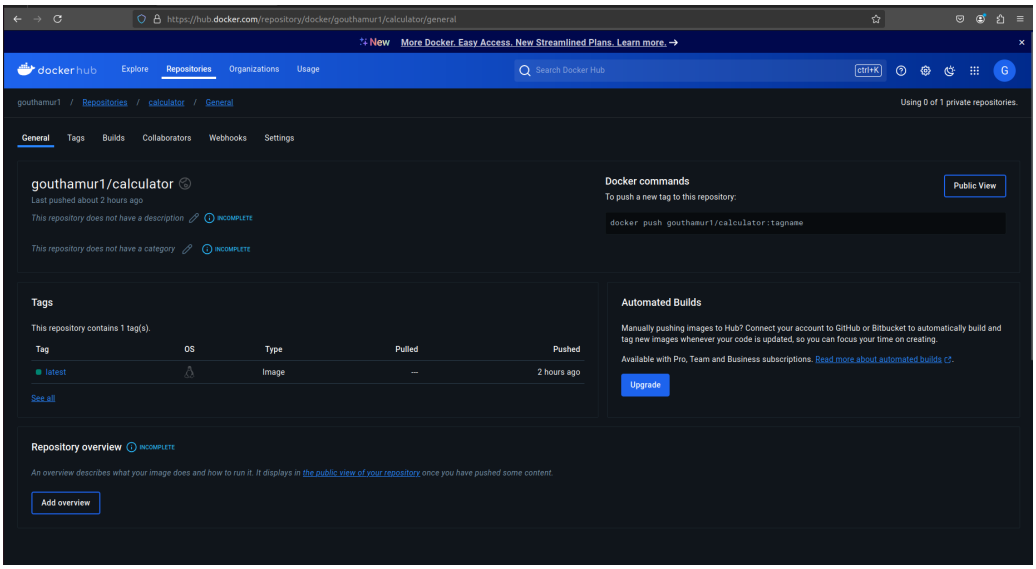

Figure 10: Docker Image on Docker Hub

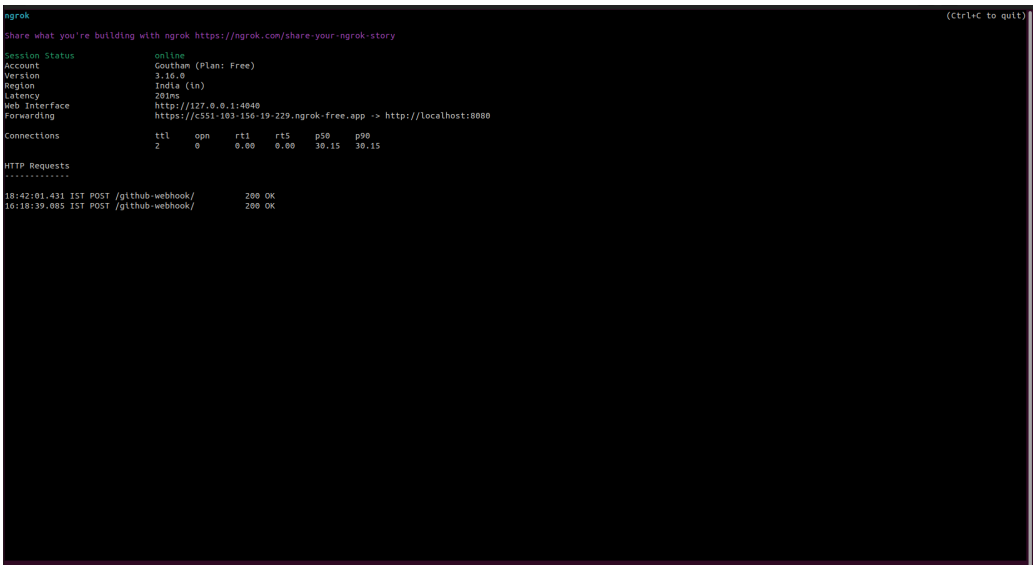

Figure 11: Ngrok

3. **Calculator_devops.log**
   - A log file that records runtime events and information generated by the Calculator application.
   - Contains log entries that indicate the execution of mathematical operations, such as calculating the square root and computing exponentiation.
   - Follows the format specified in the `Log4j2.xml` configuration, providing timestamps, thread information, and log levels.
4. **deploy.yml**
   - Ansible playbook that automates the deployment of the Calculator Docker image.
   - Includes tasks to pull the latest Docker image, stop and remove any existing containers, start the Docker service, and run a new container from the image.
   - Ensures the environment is ready for the application to run by managing Docker containers effectively.
5. **inventory**
   - An Ansible inventory file that defines the hosts where the playbook will be executed.

- Specifies the localhost as the target host for running the Ansible tasks, with local connection settings.
- Indicates the user (`gouthamur`) who will execute the Ansible commands on the localhost.

12 Main.java

1. **Purpose**: The `Main` class serves as the entry point for the Calculator application, providing a command-line interface for users to perform various mathematical operations.
2. **Logging**: Utilizes Log4j for logging important operations and user inputs, aiding in debugging and tracking application behavior (e.g., logging square root calculations, factorial calculations, and more).
3. **User Interaction**: Implements a menu-driven interface using a loop that continuously prompts the user to select a mathematical operation (square root, factorial, natural logarithm, power, or exit) until the user chooses to exit.
4. **Mathematical Operations**: Contains functionality for computing square roots, factorials, natural logarithms, and powers. Each operation is executed based on user input, with appropriate validation checks (e.g., factorials are only computed for non-negative integers).
5. **Factorial Method**: Includes a separate method to compute the factorial of a number, demonstrating modular programming practices. The method uses an iterative approach to calculate the factorial, enhancing efficiency for larger numbers.