Report

Scientific Calculator

with DevOps

Software Production Engineering Mini Project

Student: Divyam Agrawal

Roll number: IMT2019028

Course: CS 816

Contents

1. Problem Statement

Create a scientific calculator program with user menu driven operations

- Square root function - $\sqrt{x}$
- Factorial function - $x!$
- Natural logarithm (base e) - ln(x)
- Power function - x^b

2. Implementation

- GitHub Repository: https://github.com/divyamagwl/Calculator_DevOps
- DockerHub: https://hub.docker.com/repository/docker/divyamagwl/calculator

3. DevOps

3.1 What is DevOps?

- DevOps is a cultural philosophy that emphasizes on team empowerment, cross-team communication and collaboration, and technology automation.
- It is a set of methods and tools that automate and integrate processes between software development team and operation teams, allowing enterprises to produce applications and services at high velocity and improve products faster. This allows companies to better serve their customers and compete in the market.
- DevOps is built around four core principles that govern the efficacy and efficiency of application development and deployment -
    1. **Automation of the software development lifecycle:** This involves automating testing, builds, releases, provisioning of development environments, and other manual operations that can slow down or bring human error into the software delivery process.
    2. **Collaboration and communication:** Collaboration and communication are essential to facilitate the rapid and continuous delivery of software, and to ensure that everyone is working towards the same goals.
    3. **Continuous improvement and minimization of waste:** DevOps teams are constantly looking for ways to enhance their processes, from automating repetitive operations to monitoring performance data for ways to minimise release delays or mean-time-to-recovery.
    4. **Hyperfocus on user needs with short feedback loops:** DevOps teams may take a step back and focus on what real users want and how to offer it to them through automation, greater communication and cooperation, and continuous improvement.

3.2 Why DevOps?

- **Maximizes Efficiency with Automation:** DevOps maximizes efficiency and streamlines software development and deployment processes through automation. This reduces human error and accelerates software delivery. Automation allows DevOps teams to focus on high-level tasks like improving software quality, collaborating with stakeholders, and continuous improvement.
- **Optimizes the Entire Business:** DevOps compels organizations to prioritize the optimization of the entire system, rather than just individual IT silos, in order to enhance the overall business. This involves being more adaptive and utilizing data-driven approaches to align with the needs of customers and the business as a whole.
- **Improves Speed and Stability of Software Development and Deployment:** By breaking down the traditional silos between software development and IT operations, DevOps teams can work collaboratively to streamline the software delivery pipeline. This leads to faster and more frequent releases, allowing organizations to respond quickly to changing market conditions and customer demands.

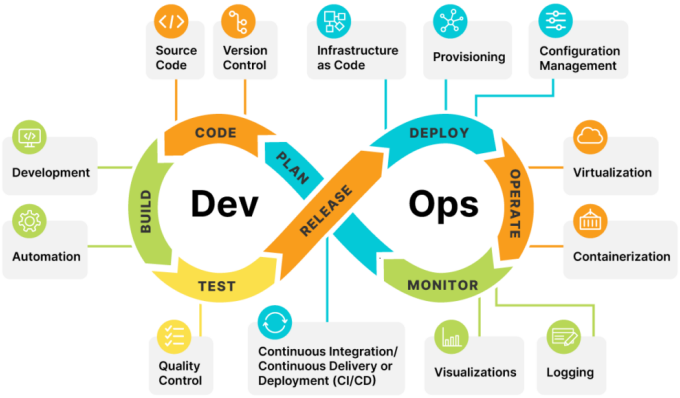


Figure 1: DevOps Cycle

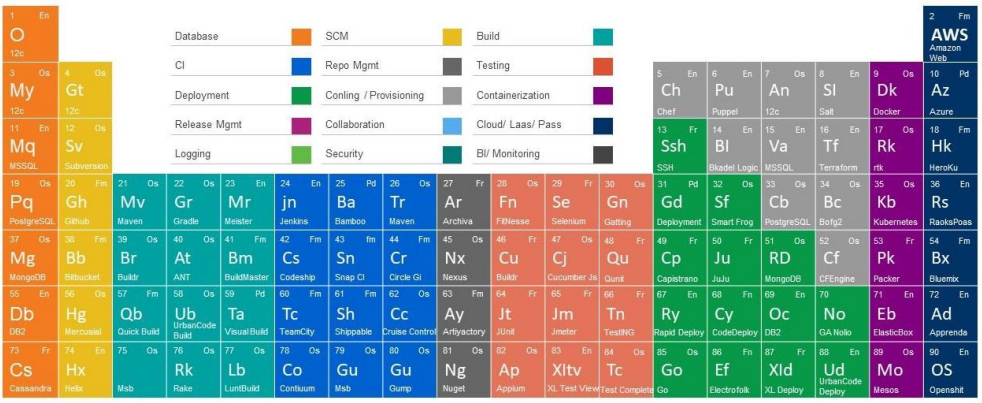


Figure 2: DevOps Tools Periodic Table

4. Tools Utilized

- **Git:** a *source code management* tool that allows developers to collaborate on code and track changes.
- **JUnit**: an open-source *testing framework* for Java that enables developers to write and run unit tests.
- **Maven:** a *build automation* tool that helps manage dependencies and build Java-based projects.
- **Jenkins:** a *continuous integration and continuous delivery (CI/CD)* tool that automates the build, test, and deployment processes.
- **GitHub Webhooks:** a tool that *trigger automated actions* when specific events occur in a GitHub repository.
- **Docker:** a *containerization platform* that enables developers to package applications and dependencies into portable, lightweight containers.
- **Ansible:** a *configuration management* tool that automates the deployment and management of infrastructure and applications.
- **ELK:** an acronym for Elasticsearch, Logstash, and Kibana, which are open-source tools used for *centralized logging and log analysis*.
- **ngrok:** a tool that creates *secure tunnels* to expose local servers to the public internet, useful for testing and development purposes.

5. Development and Deployment Process

Before starting, **install** the following tools:

- Git : https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- Java: https://www.oracle.com/in/java/technologies/downloads/#java11
- IntelliJ IDE: https://www.jetbrains.com/idea/download/#section=mac
- Maven: https://maven.apache.org/install.html
- Jenkins: https://www.jenkins.io/doc/book/installing/
- Docker: https://docs.docker.com/engine/install/
- Ansible: https://docs.ansible.com/ansible/latest/installation_guide/index.html
- ngrok: https://ngrok.com/download

Create repositories on these platforms:

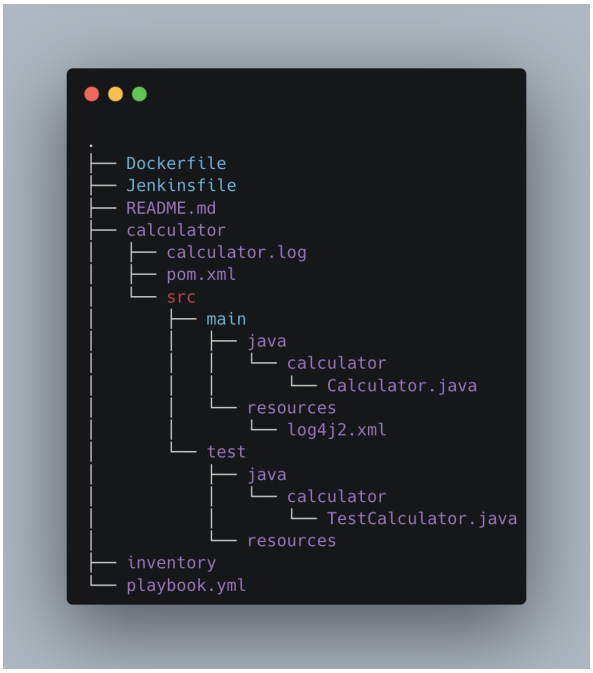
- GitHub: https://github.com/new
- DockerHub: https://hub.docker.com/repository/create

We will cover the following steps in detail in the upcoming sections:

1. Writing Java code for a Scientific Calculator program.
2. Writing unit tests using JUnit.
3. Committing and pushing changes to your GitHub repository.
4. Writing a pipeline script in Jenkins for
   (a) Git Pull
   (b) Maven Build
   (c) Docker Image Creation
   (d) Pushing Image to DockerHub
   (e) Ansible Deployment
5. Attaching the container to your terminal.
6. Uploading the log file to ELK.

5.1 Code, Build and Test

- Tools used -
  - Programming Language - Java (version - 11)
  - Logging - Log4j
  - Testing - JUnit
  - Build - Maven
- Directory structure -



- Description of files -
  - **Calculator.java** - Contains the main function including the menu and the following functions -
    - Square root function - $\sqrt{x}$
    - Factorial function - $x!$
    - Natural logarithm (base e) - ln(x)
    - Power function - x^b
  - **TestCalculator.java** - Contains JUnit test cases for the functions described above. Each function has two types of test cases: true positive and false positive.

Figure 3: Example Test case

  - **log4j2.xml** - Includes basic configurations for log4j, such as which appender to use (e.g., console appender, file appender) and other settings like pattern and log level.

Figure 4: Log4j configurations

– **pom.xml** - Contains information about the project and configuration details used by Maven to build the project.

Figure 5: Added Dependencies for JUnit and Log4j

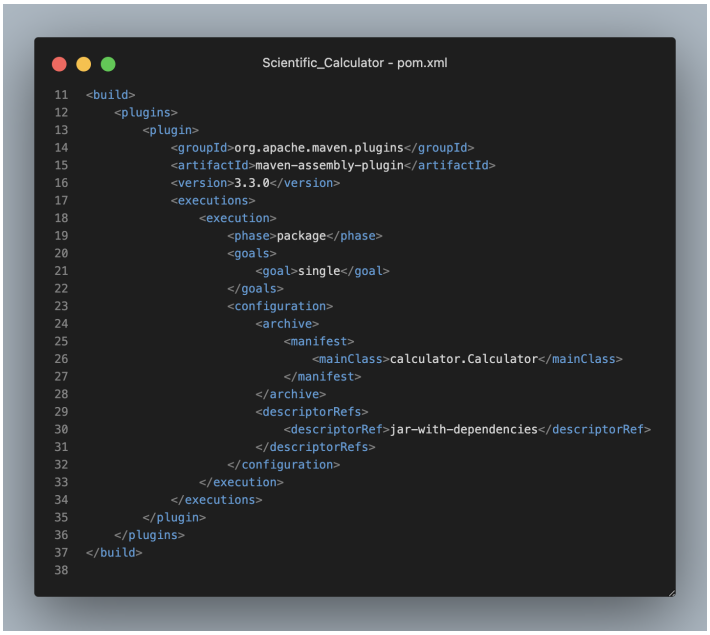


Figure 6: Maven Assembly Plugin to generate jar file with dependencies

- To generate a JAR file with dependencies, run the following command

  ```
  $ mvn clean install
  ```

  Maven will build the project and check all test cases. Once completed, a ”target” directory will be created in the current directory, which will contain the JAR file.

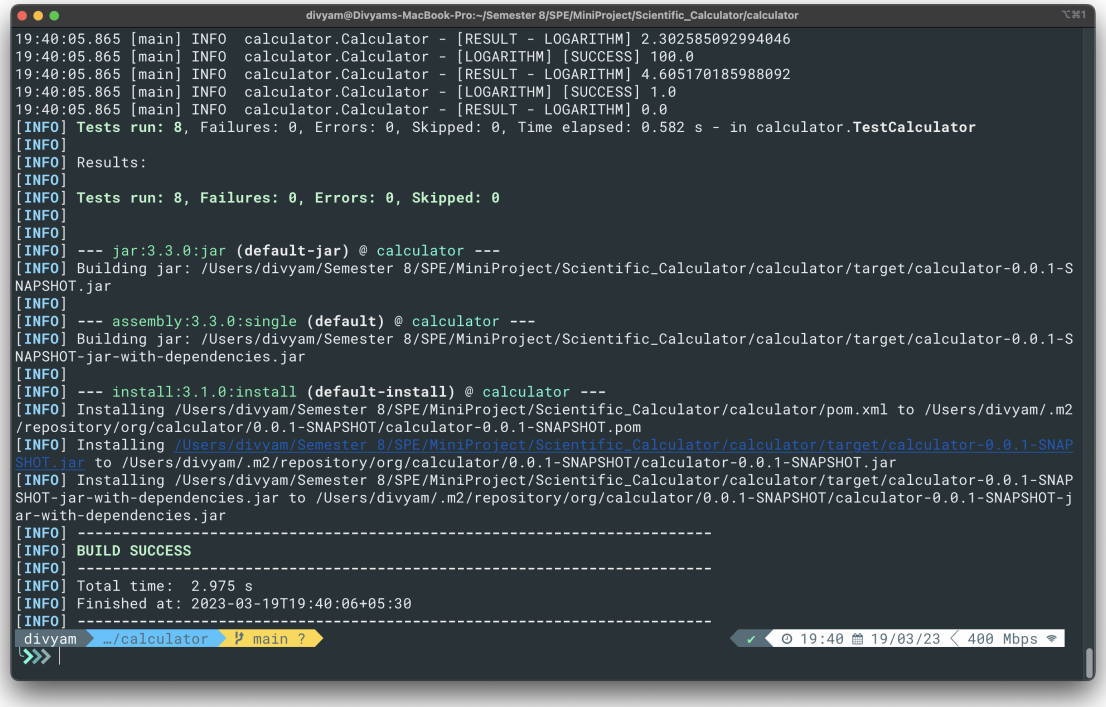


Figure 7: Console Output

5.2 Source Code Management

- SCM tools allow developers to collaborate on code, track changes and maintain version control of their codebase.
- SCM tools also provide features such as branching and merging, which allow developers to work on different versions of the codebase and merge changes together.
- SCM Tool Used - Git and GitHub
- Create a new GitHub repository
- To push changes to the repository, use the following commands

```
$ git init
$ git add .
$ git commit -m "Message"
$ git branch -M main
$ git remote add origin <GitHub repo URL>
$ git push origin master
```

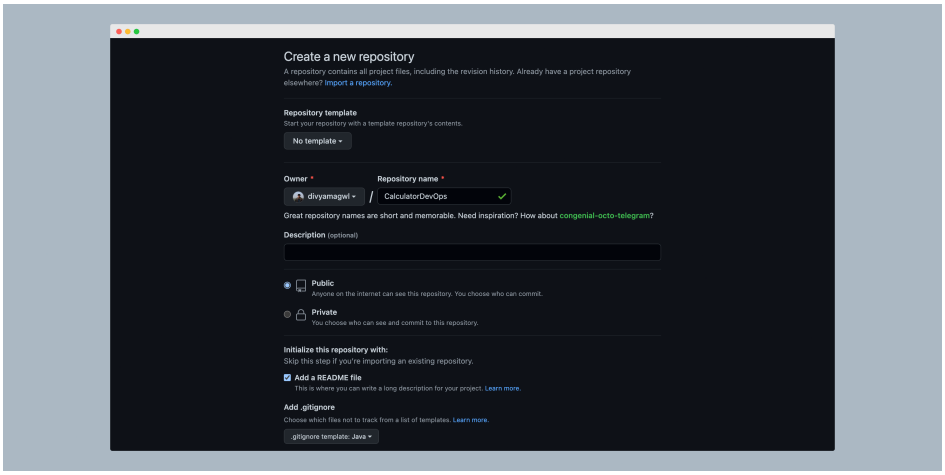


Figure 8: Creating a new repository on GitHub

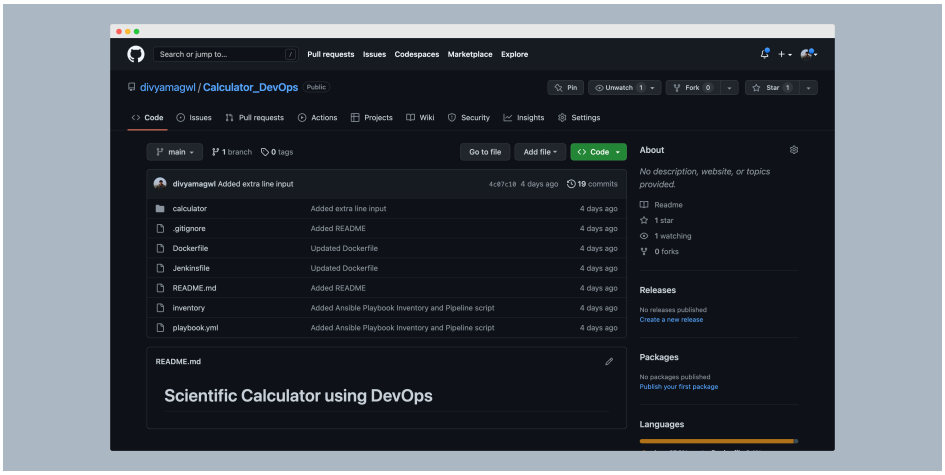


Figure 9: My GitHub repository

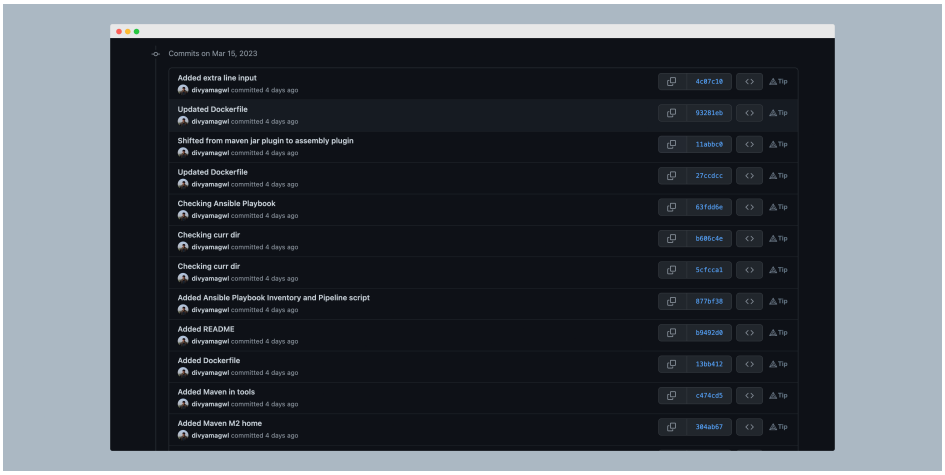


Figure 10: My GitHub commits

5.3 Continuous Integration / Continuous Deployment

- Jenkins is an open-source automation tool written in Java with plugins built for continuous integration.
- Jenkins is utilized to continuously build and test software projects, simplifying the process for developers to integrate changes, and allowing users to obtain up-to-date builds with ease.
- Additionally, Jenkins enables continuous delivery of software by integrating with a wide range of testing and deployment technologies.
- We utilize Jenkins Pipeline for our project. Jenkins Pipeline is a suite of plugins that allows developers to create and manage continuous delivery pipelines as code. It enables developers to define the entire software delivery process in a single pipeline script.
- After Jenkins is installed, browse to http://localhost:8080 (or whichever port you configured for Jenkins when installing it)

5.3.1 Jenkins Plugins

Browse to Manage Jenkins → Plugin Manager → Available Plugins and install the following plugins -

- Git plugins & GitHub plugins
- Maven Integration
- Docker plugin & Docker pipeline
- Ansible plugin

Browse to Manage Jenkins → Global Tool Configuration and use the following configurations -

Figure 11: Git Installations config

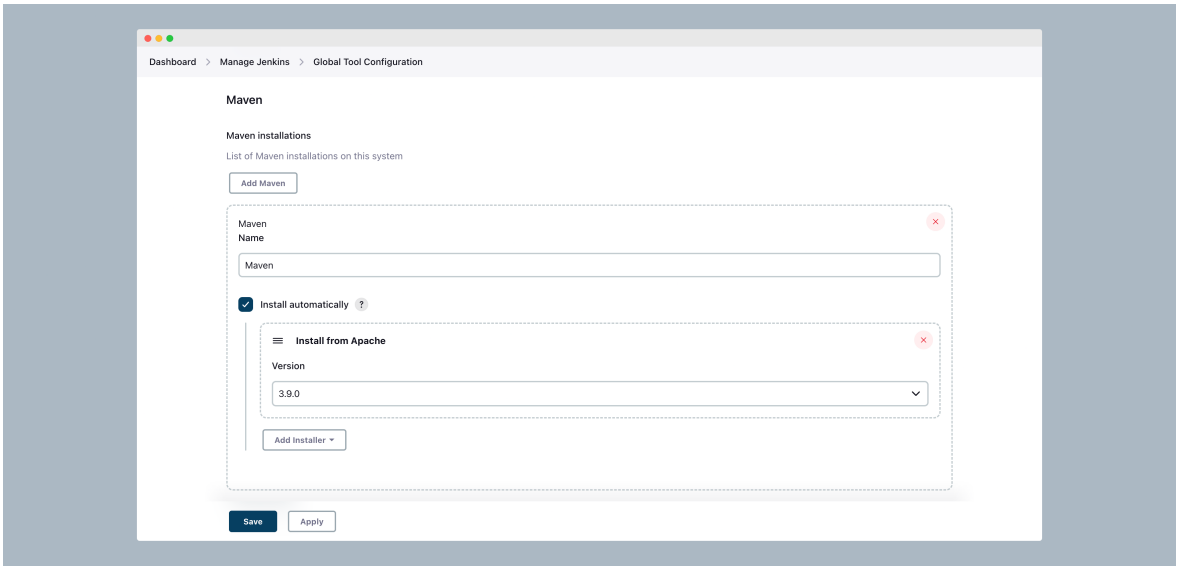


Figure 12: Maven Installations config

MacOS users will need to follow these specific steps in order to use Docker and Ansible with Jenkins. This involves adding the local installation paths for both tools to Jenkins. Please refer to the following Stack Overflow question for more details.

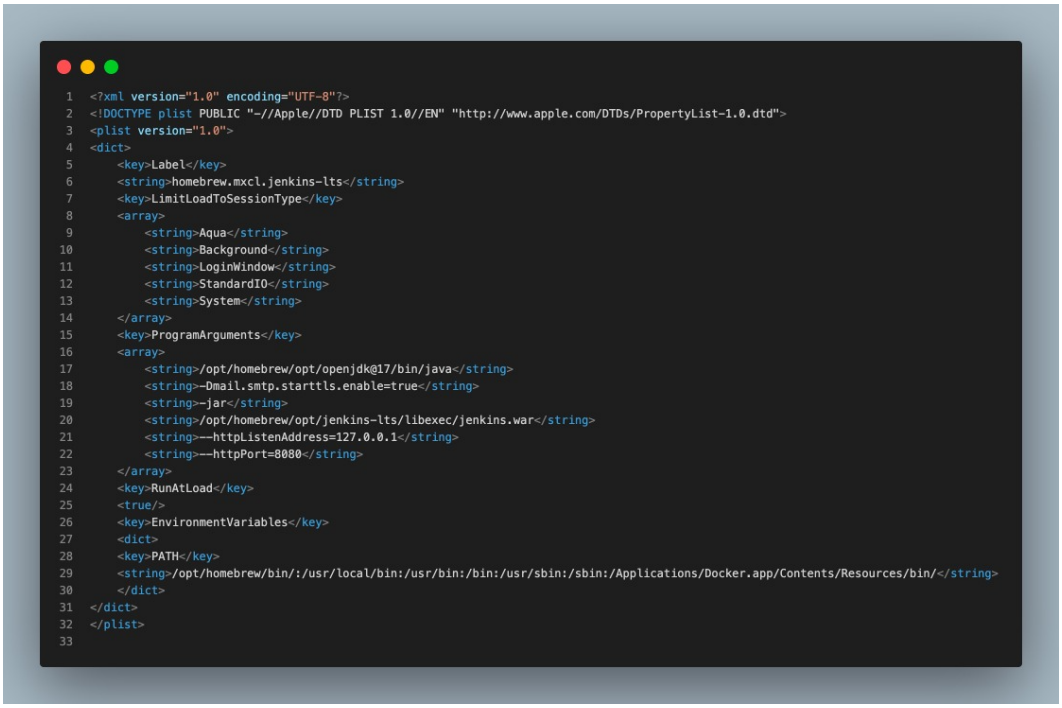
- Go to the following directory

```
$ cd /opt/homebrew/Cellar/jenkins-lts/<jenkins-
    version>
```

- Open homebrew.mxcl.jenkins-lts.plist file.
- Add the following code snippet after RunAtLoad and <true> :

```
<key>EnvironmentVariables</key>
<dict>
<key>PATH</key
<string>/opt/homebrew/bin/:/usr/local/bin:/usr/bin
    :/bin:/usr/sbin:/sbin:/Applications/Docker.app/
    Contents/Resources/bin/</string>
</dict>
```

- We have included `/opt/homebrew/bin/` as the installation path for Ansible.
- The remaining path `/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/` has been added as the installation path for Docker.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>homebrew.mxcl.jenkins-lts</string>
    <key>LimitLoadToSessionType</key>
    <array>
        <string>Aqua</string>
        <string>Background</string>
        <string>LoginWindow</string>
        <string>StandardIO</string>
        <string>System</string>
    </array>
    <key>ProgramArguments</key>
    <array>
        <string>/opt/homebrew/opt/openjdk@17/bin/java</string>
        <string>-Dmail.smtp.starttls.enable=true</string>
        <string>-jar</string>
        <string>/opt/homebrew/opt/jenkins-lts/libexec/jenkins.war</string>
        <string>--httpListenAddress=127.0.0.1</string>
        <string>--httpPort=8080</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>EnvironmentVariables</key>
    <dict>
    <key>PATH</key>
    <string>/opt/homebrew/bin/:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/</string>
    </dict>
</dict>
</plist>

```

Figure 13: homebrew.mxcl.jenkins-lts.plist

Browse to Manage Jenkins → Manage Credentials and add your DockerHub credentials and GitHub Personal Access token.

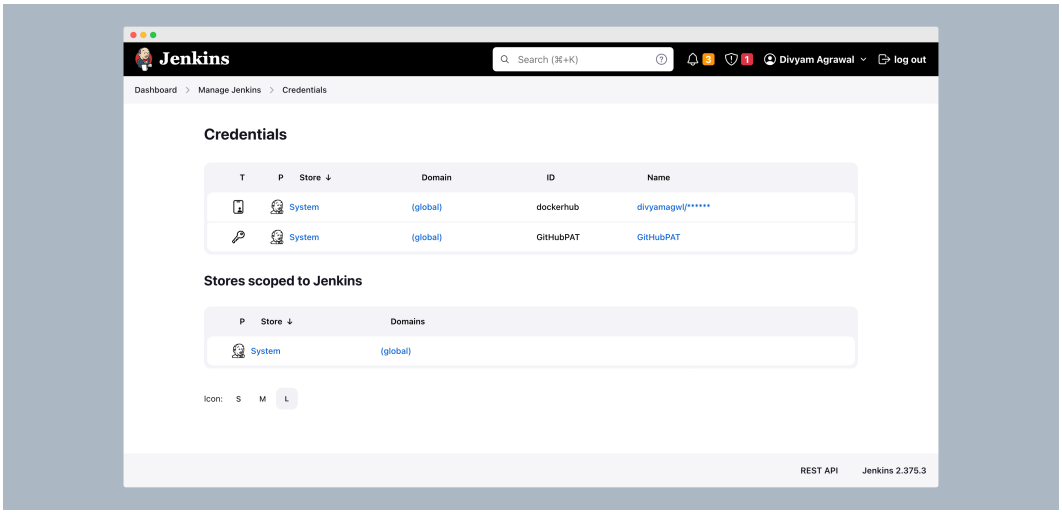


Figure 14: Credentials stored in Jenkins

Browse to Manage Jenkins → Configure system and add GitHub server with the following configuration.

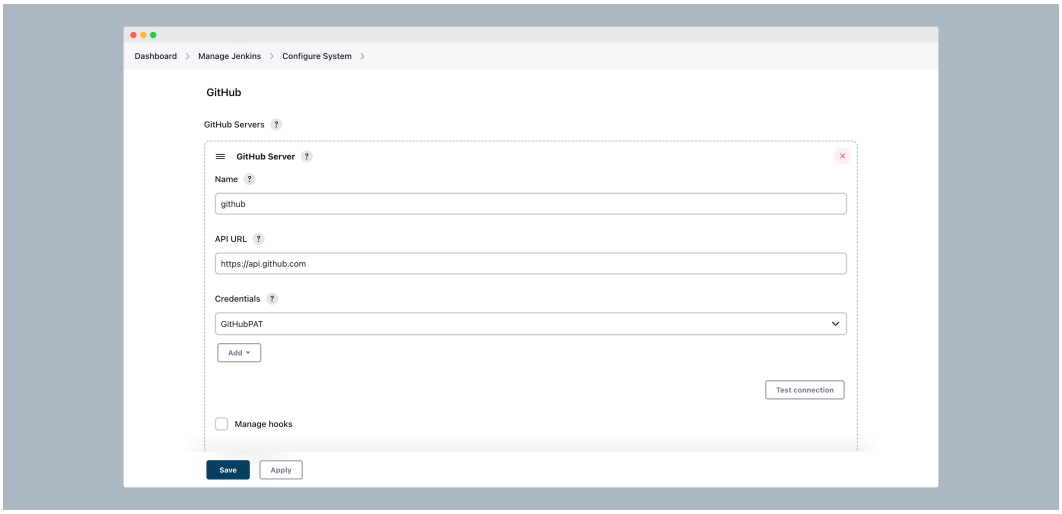


Figure 15: GitHub Server configuration in Jenkins

5.3.2 Jenkins Pipeline

- Create a Jenkinsfile which will contain the stages of the Jenkins pipeline -
  - **Pull GitHub:**

```
stage('Pull GitHub') {
    steps {
        git branch: 'main', url: 'https://github
           .com/divyamagwl/Calculator_DevOps'
    }
}
```

    Pulls the remote repository from GitHub using Jenkins.

  - **Build Maven jar package:**

```
stage('Build Maven jar package') {
    steps {
        dir("calculator/") {
            script{
                sh 'mvn clean install'
            }
        }
    }
}
```

    Generates a JAR file with all required dependencies, while also deleting the old target folder and installing all dependencies again for the new target folder.

– **Docker Image Build:**

```
environment
{
    registry = "divyamagwl/calculator"
    registryCredential = "dockerhub"
    dockerImage = ""
}

stage('Docker Image Build') {
    steps {
        script {
            dockerImage = docker.build(registry
               + ":latest")
        }
    }
}
```

The docker build command builds Docker Images from a Dockerfile and a "context". A build's context is the set of files located in the specified PATH or URL.

– **DockerHub Image Push:**

```
stage('DockerHub Image Push') {
    steps {
        script {
            docker.withRegistry('',
               registryCredential) {
                dockerImage.push()
            }
        }
    }
}
```

The docker push command shares your image to the DockerHub registry.

– **Removing Docker Image:**

```
stage('Cleaning Up') {
    steps {
        sh "docker rmi $registry:latest"
    }
}
```

The docker rmi command removes one or more images from the host node.

– **Execute Ansible:**

```
stage('Execute Ansible') {
    steps {
        ansiblePlaybook colorized: true,
        installation: 'Ansible',
        inventory: 'inventory',
        playbook: 'playbook.yml'
    }
}
```

Runs Ansible Playbook.

- Browse to Add Item and select Pipeline project.

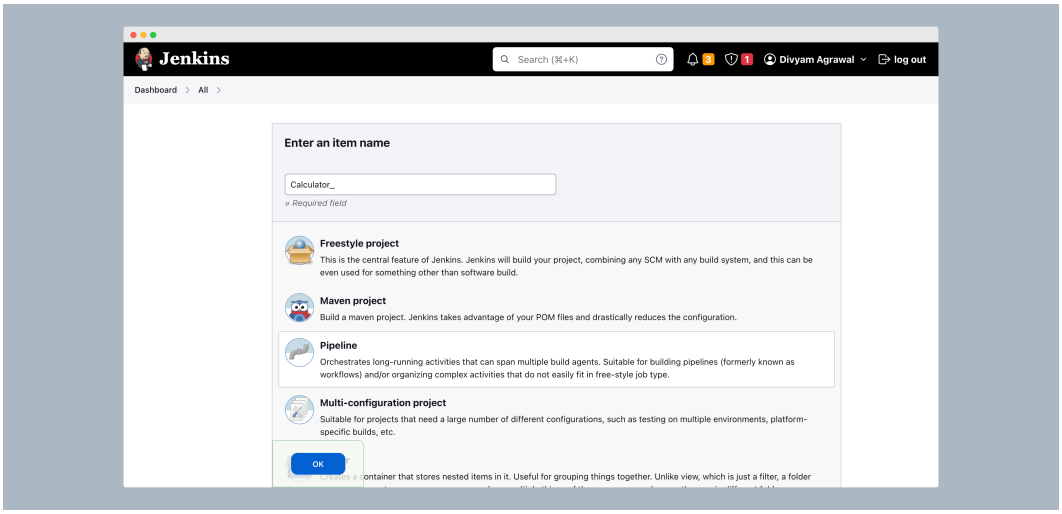


Figure 16: Creating a new Jenkins Project

- For the job to work with a GitHub repository, use the following configuration -

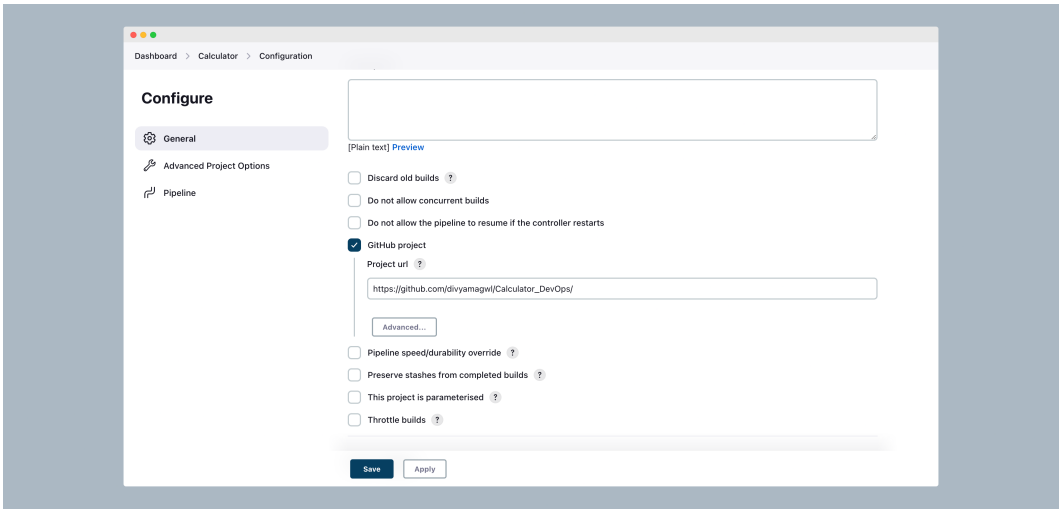


Figure 17: Selecting GitHub project

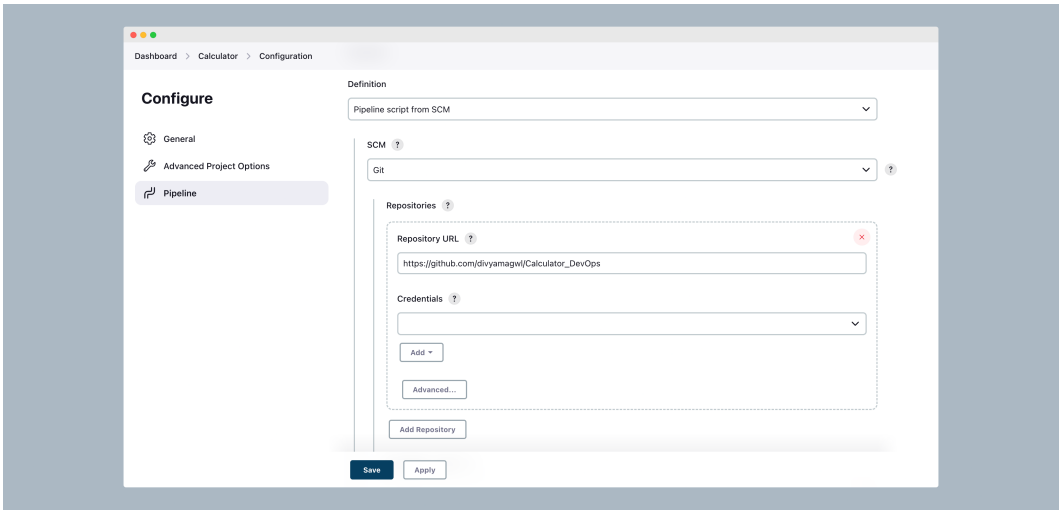


Figure 18: Pipeline script from SCM

- My Jenkins Project

Figure 19: Pipeline Execution

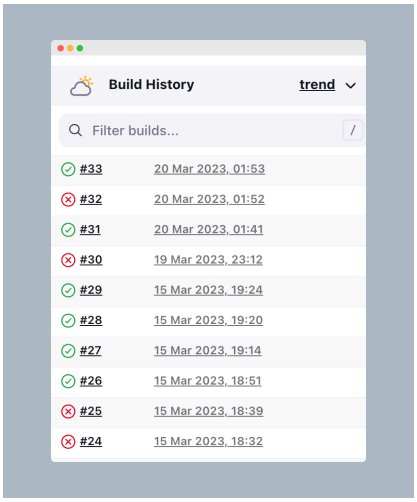


Figure 20: Build History

- Upon a successful build, the Docker Image and running container will be visible on the host node. We can attach the running container to our terminal using the `docker attach` command.

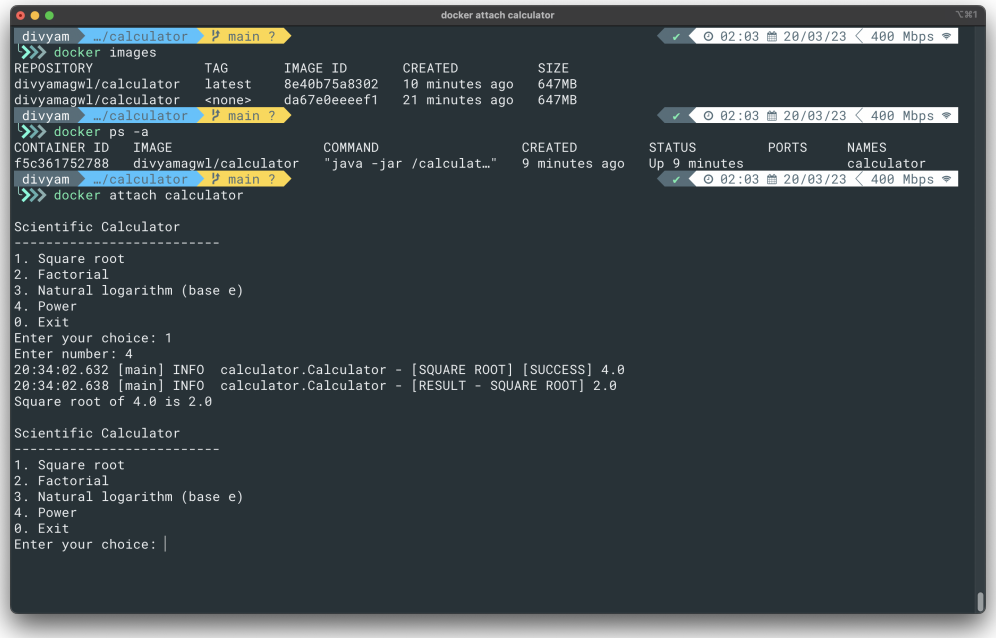


Figure 21: Docker attach running container

5.3.3 GitHub Webhook

- A Webhook is a mechanism to automatically trigger the build of a Jenkins project in response to a commit pushed to a Git repository.
- We will use Ngrok to expose Jenkins local web server running on port 8080 to the internet. Ngrok is a tool that enables developers to create secure tunnels from a public endpoint to a locally running web service.
- After installing Ngrok, execute the following command-

```
$ ngrok http 8080
```

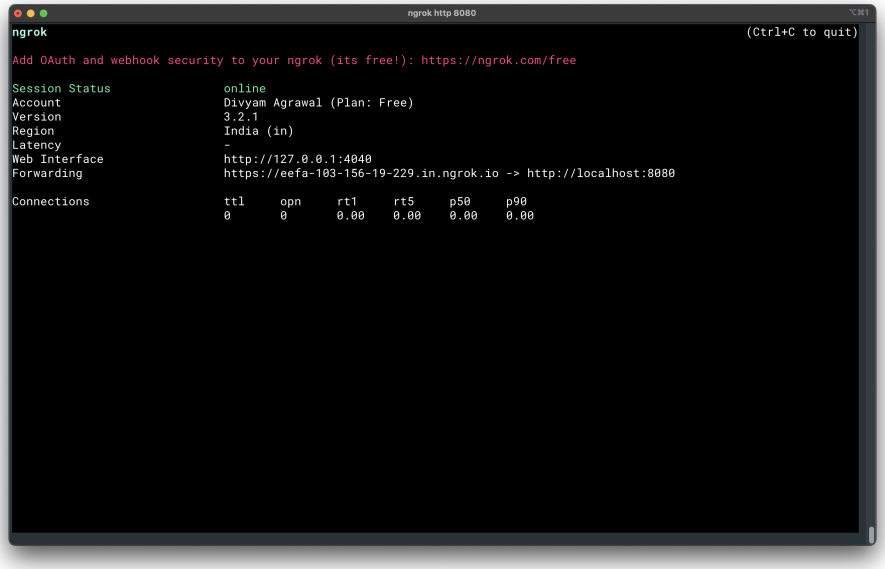


Figure 22: Ngrok secure tunnel

- Browse to Manage Jenkins → Configure system and add Ngrok Forwarding URL to Jenkins URL.

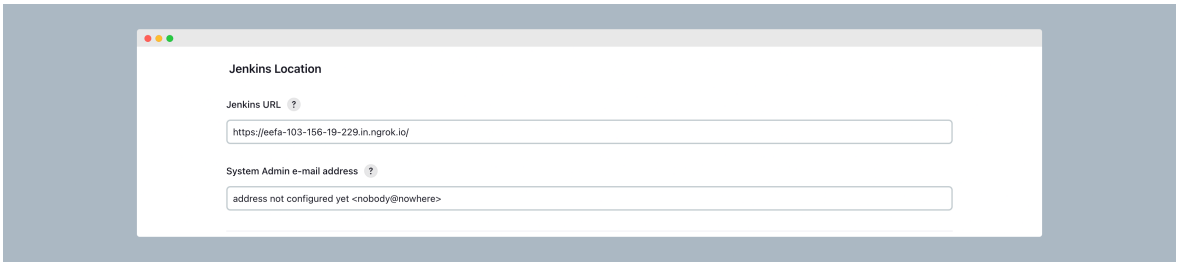


Figure 23: Add Ngrok URL in Jenkins URL

- Browse to Your Jenkins Project → Configure → Build Triggers and select `GitHub hook trigger for GITScm polling`

- In your GitHub repository, browse to Settings → Webhooks and add Ngrok Forwarding URL appended with "github-webhook/" in payload URL and Personal Access token in Secret.

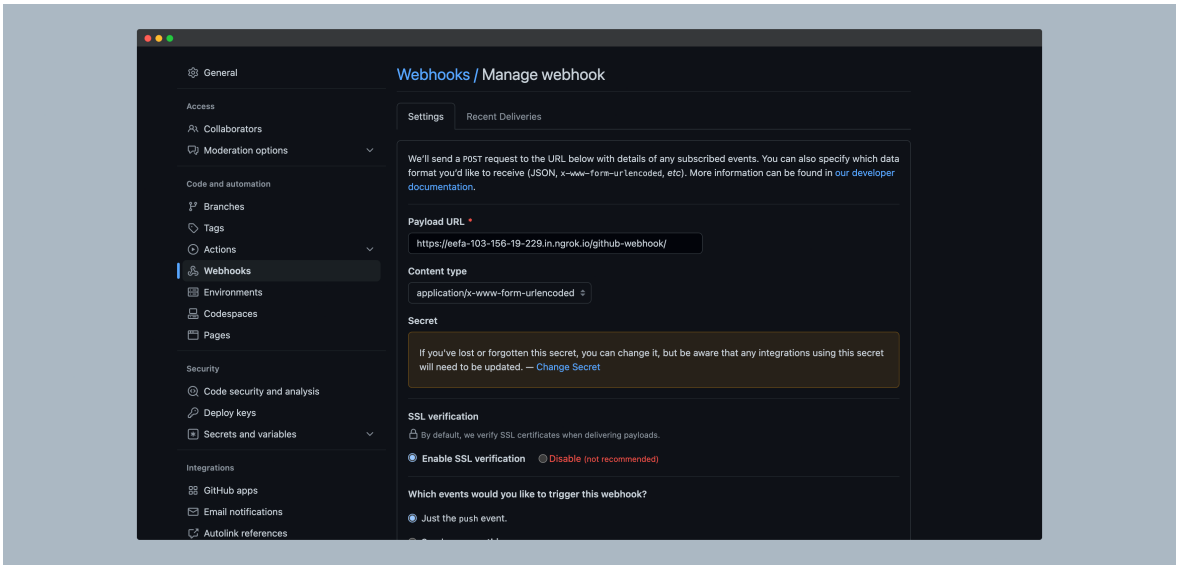


Figure 24: Add Payload URL and Secret

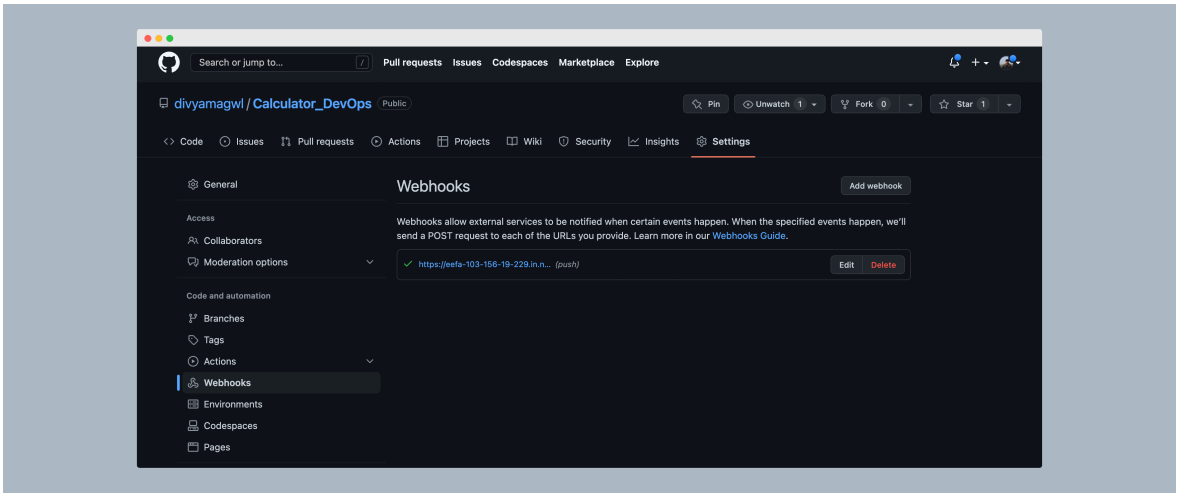


Figure 25: Successful Delivery

5.4 Containerization

- Containerization is a software deployment process that bundles an application's code with all the files and libraries it needs to run on any infrastructure.
- Containers are lightweight, portable, and self-contained environments that enable developers to package an application with all its dependencies, libraries, and configuration files, ensuring that it runs consistently across different environments.

- We will use Docker to create containers. Docker is an open-source tool that enables developers to build, package, and deploy applications in a containerized environment.
- Create a Dockerfile

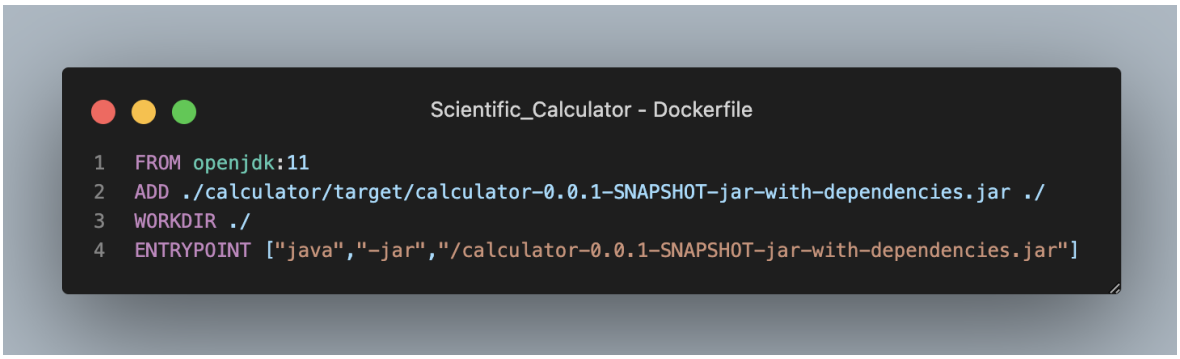


Figure 26: Dockerfile

  - **FROM:** We use the OpenJDK 11 base image to build our image.
  - **ADD:** Copy jar file from source on the host machine into the container's file system.
  - **WORKDIR:** Changes the current working directory.
  - **ENTRYPOINT:** Specify the command that should be run when a container based on the image is started.
- The Jenkins Pipeline script has been configured to include the Dockerfile and Docker commands, automating the process of building and pushing the Docker Image to DockerHub.

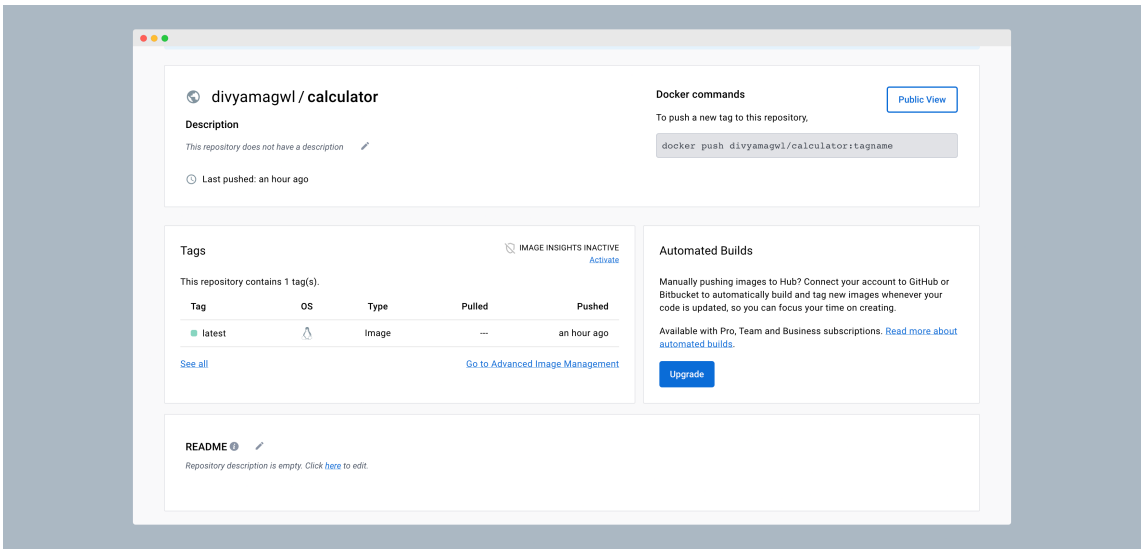


Figure 27: DockerHub public repository

5.5 Deployment

- We will use Ansible for local deployment. Ansible is a suite of software tools that enables infrastructure as code.
- In Ansible, managed hosts or servers which are controlled by the Ansible control node are defined in a host inventory file. The Ansible inventory file defines the hosts and groups of hosts upon which commands, modules, and tasks in a playbook operate.
- Create an inventory file where we explicitly define our localhost in the inventory file for local deployment.

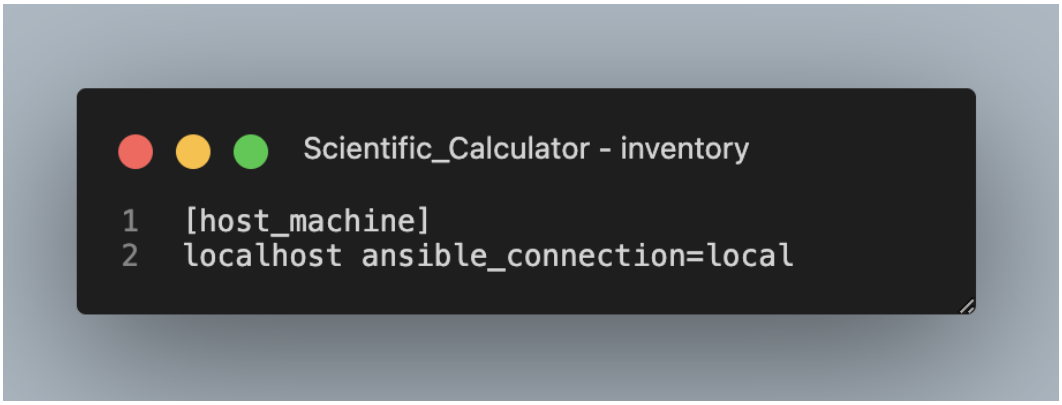


Figure 28: Ansible Inventory

- Ansible Playbooks offer a repeatable, re-usable, simple configuration management.
- Playbooks consist of one or more plays run in a particular order. A play is an ordered set of tasks run against hosts chosen from your inventory. Plays define the work to be done. Each play contains a set of hosts to configure, and a list of tasks to be executed.

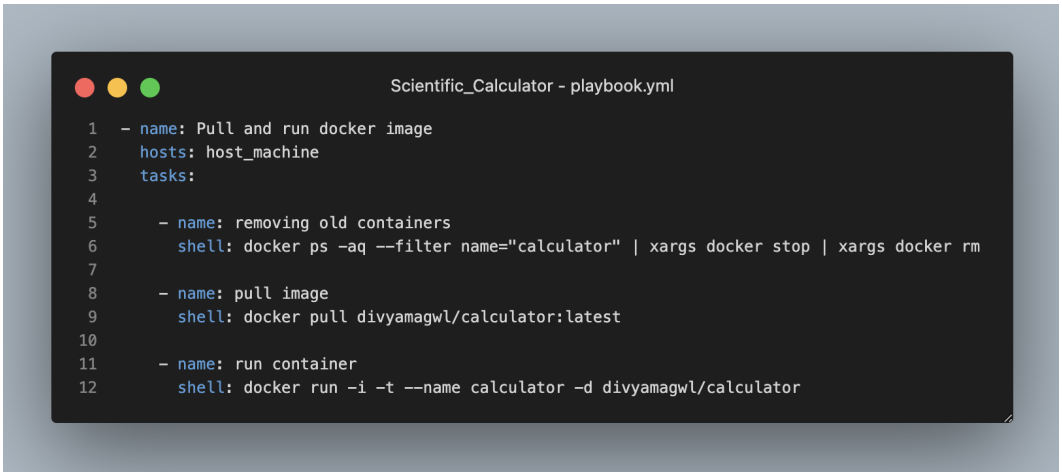


Figure 29: Ansible Playbook

– Filtering out all the containers with the name ”calculator”, then stopping the running containers and removing them from the host machine.

```
- name: removing old containers
  shell: docker ps -aq --filter name="calculator
     " | xargs docker stop | xargs docker rm
```

– Pulling our docker image from DockerHub to the host machine.

```
- name: pull image
  shell: docker pull divyamagwl/calculator:
     latest
```

– Creating a writeable container layer over our specified image and then starting it.

```
- name: run container
  shell: docker run -i -t --name calculator -d
     divyamagwl/calculator
```

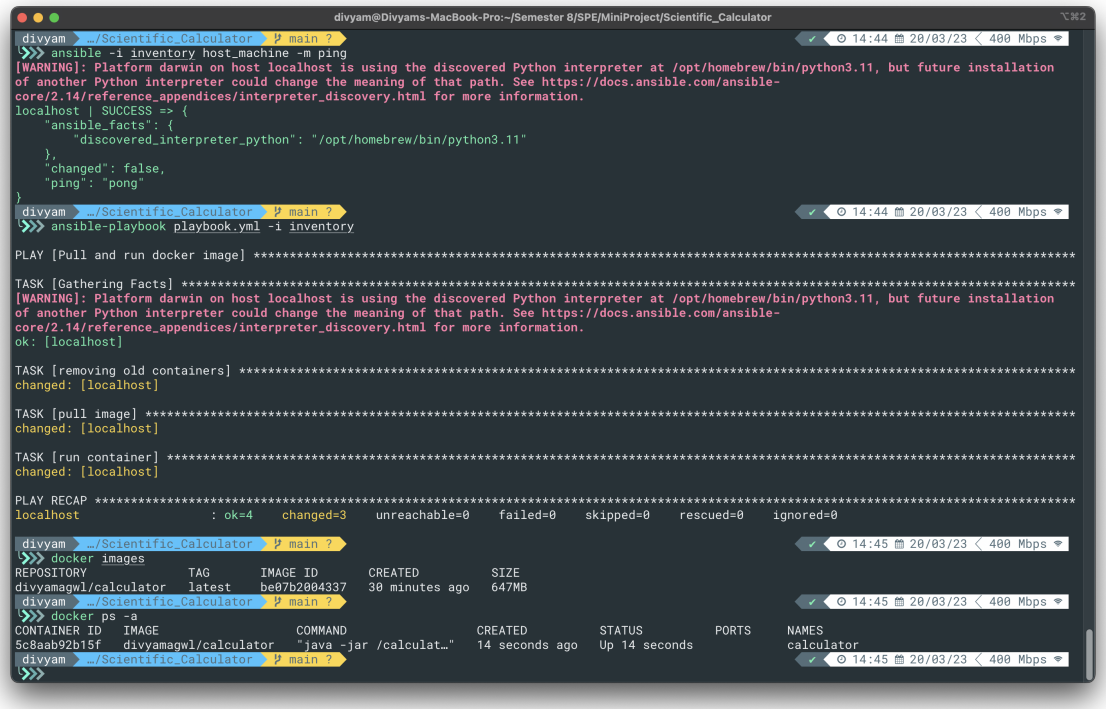


Figure 30: Pinging host machine and running playbook manually

- The Jenkins Pipeline script has been configured to execute the Ansible Playbook automatically.

5.6 Monitoring

- ELK stack is an open-source toolset that is used for log management and analysis. The ELK stack consists of three main components:
  - Elasticsearch: A distributed, RESTful search and analytics engine designed to store, search, and analyze large volumes of data in real-time.
  - Logstash: A data processing pipeline that collects, parses, and enriches log data from various sources and sends it to Elasticsearch for storage and analysis.
  - Kibana: A web-based user interface for visualizing and analyzing data stored in Elasticsearch, including metrics, logs, and application performance data.
- Go to https://www.elastic.co/cloud/elasticsearch-service/signup and sign up using a temporary email address.
- Follow the instructions provided by the platform to create a deployment and wait a few minutes for the deployment process to complete.

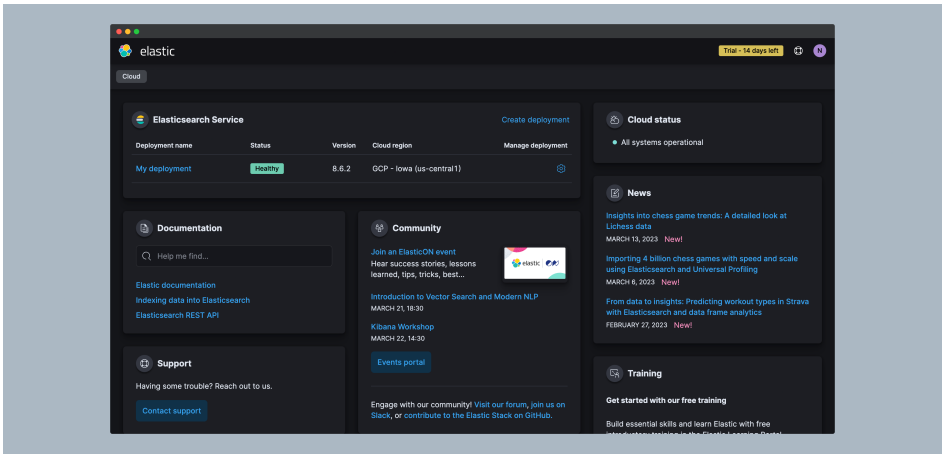


Figure 31: Elastic Cloud Home

- To obtain the log file from your Docker container, use the following command:

```
$ docker cp calculator:/calculator.log ../
  monitoring
```

- Now, navigate to Your Deployment → Analytics → Machine Learning → Data Visualizer → File
- Upload the log file to the dashboard.

- Click on ”Override settings” and update the grok pattern with the following:

```
%{TIMESTAMP_ISO8601:timestamp} \[%{WORD:thread}\]
   %{LOGLEVEL:log.level} .*? - \[%{WORD:function}\]
    - %{NUMBER:value1}(\s%{NUMBER:value2})? - \[
   RESULT\] - %{GREEDYDATA:result}
```

  This grok pattern is based on the log outputs that were written during the code development.

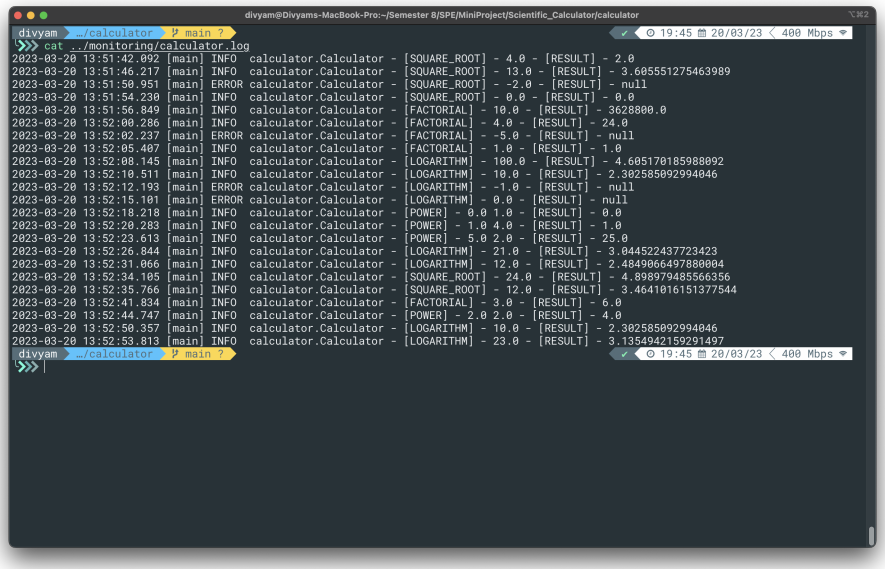


Figure 32: My Log file

- Finally, import the file and specify an index name.

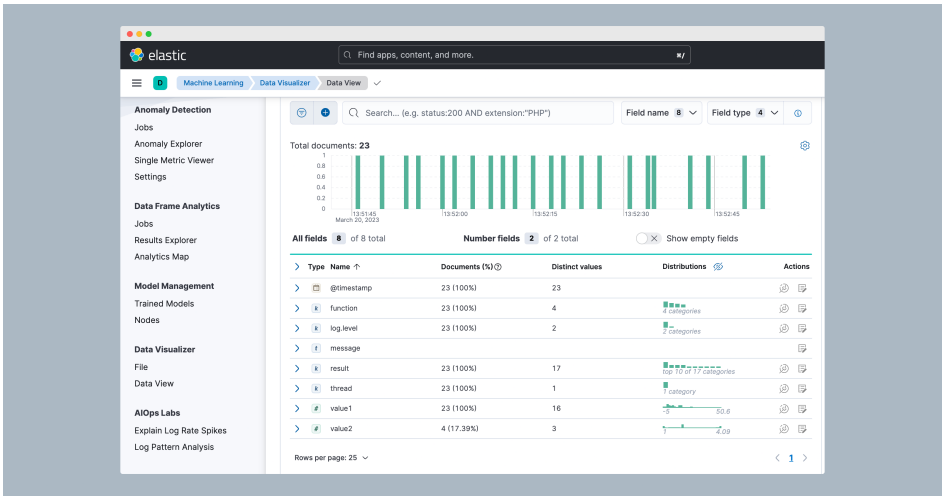


Figure 33: Data View

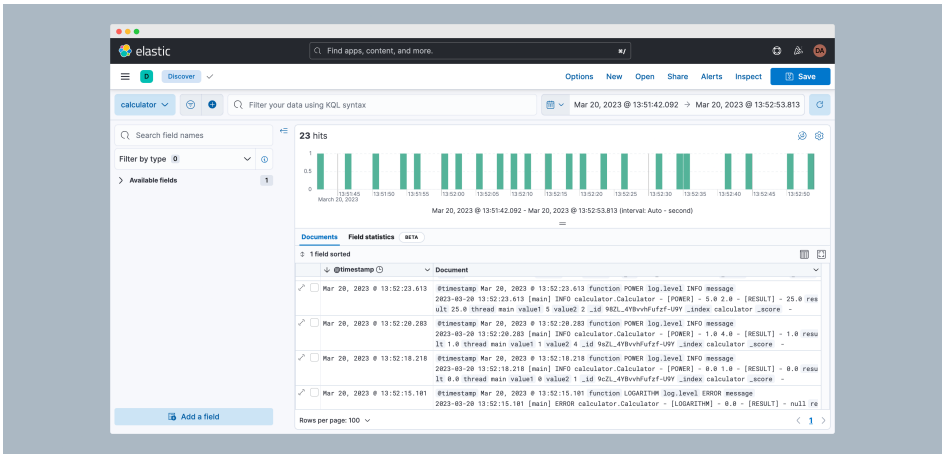


Figure 34: Discover View with documents from log

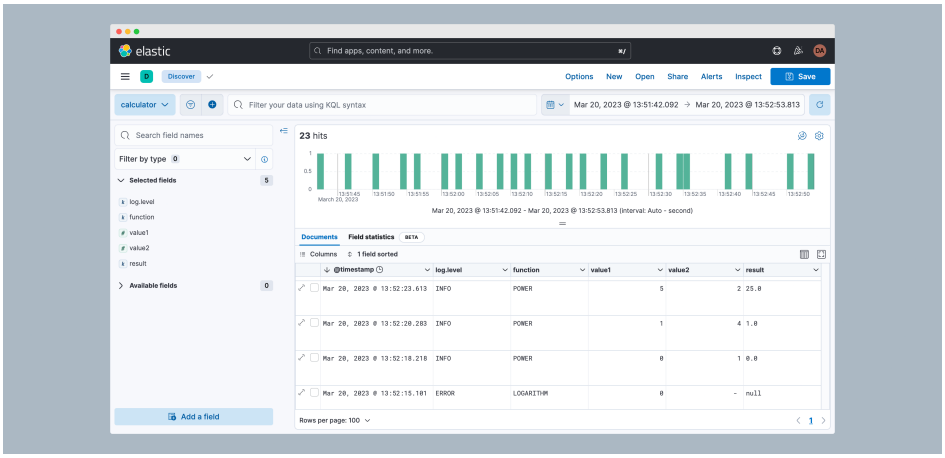


Figure 35: Discover View with selected fields from log

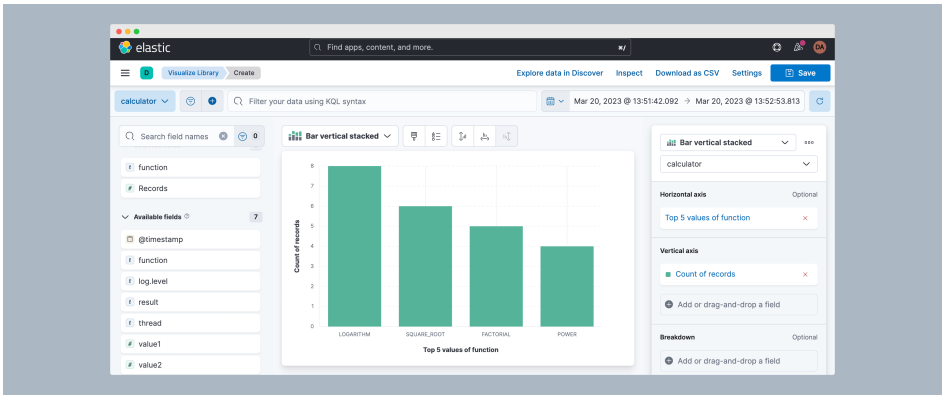


Figure 36: Function call count visualization

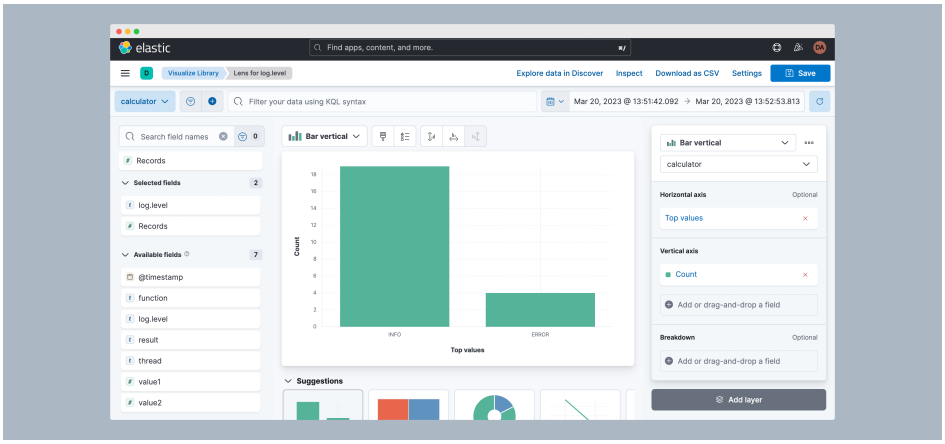


Figure 37: Log Level count visualization

5.7 Scientific Calculator Outputs

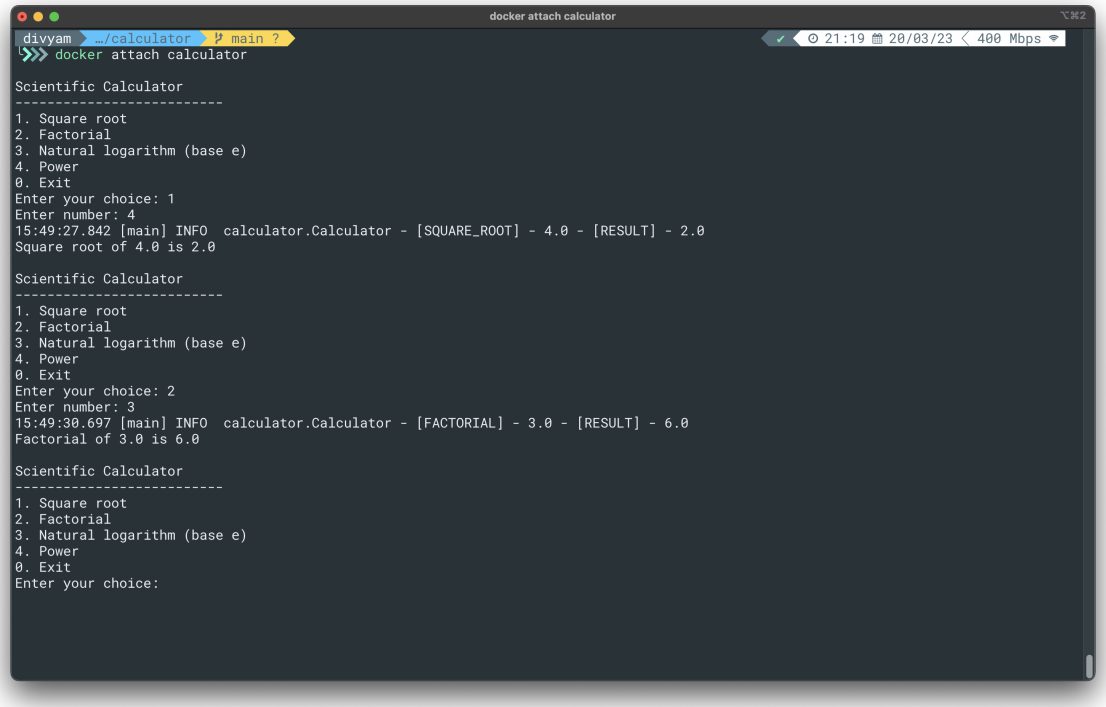


Figure 38: Square root and Factorial functions

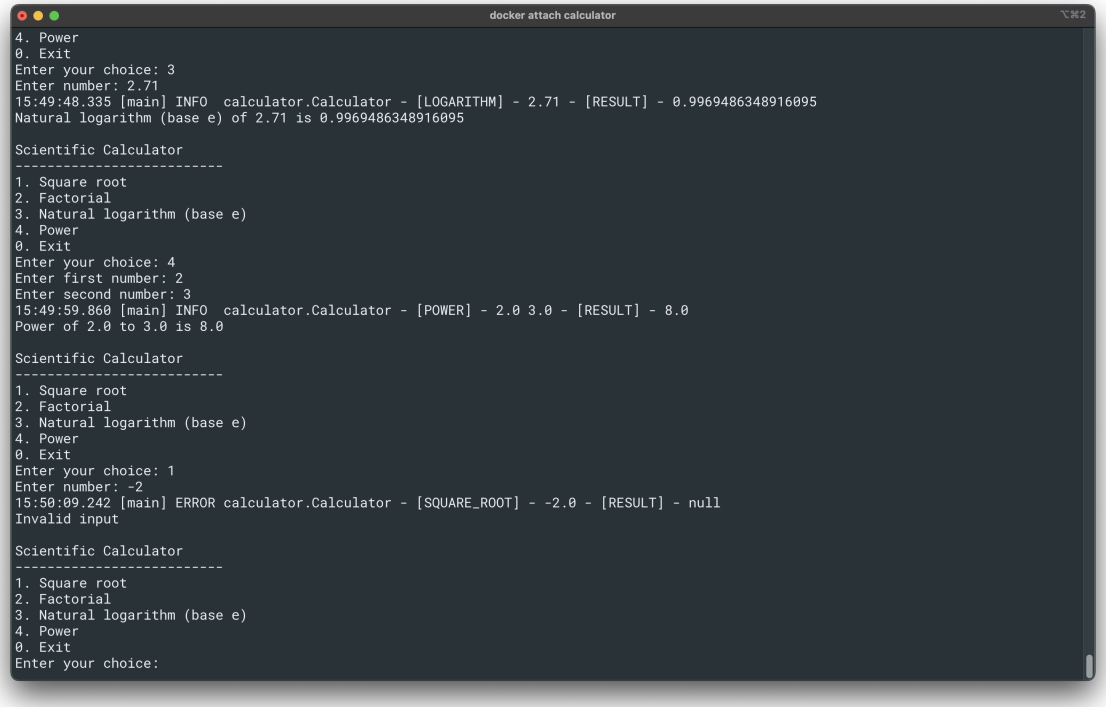


Figure 39: Natural log and Power functions

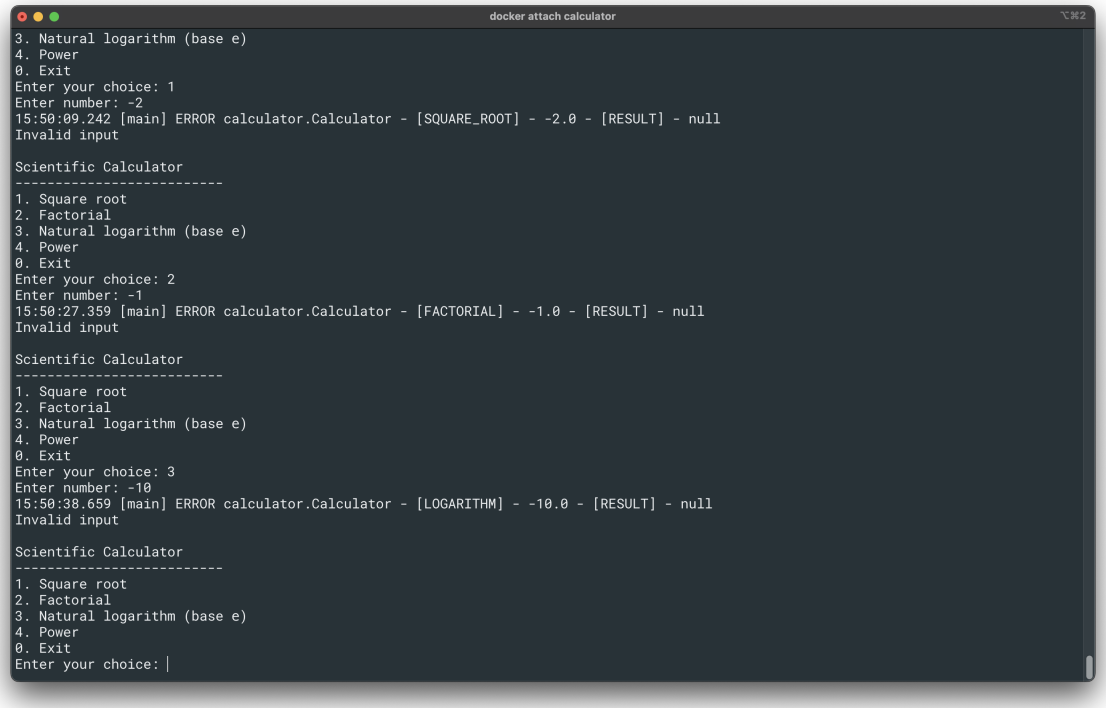


Figure 40: Invalid inputs