

IIT Bangalore
CS 816 Software Production Engineering
Mini Project

Scientific Calculator
using DevOps



Project Report

G Sri Harsha (IMT2019030)

Project Report

G Sri Harsha (IMT2019030)

March 21, 2023

Contents

1	Problem Statement	3
2	Codebase	3
3	What is DevOps?	3
4	Why DevOps?	4
5	Tools used	4
6	Development	5
6.1	Creating, Implementing and Writing JUnits	5
6.2	SCM using Github	8
6.3	Containerization using Dockerfile	9
6.4	Configuration management using Ansible	10
6.5	Development pipeline using Jenkins	11
6.6	Automatic triggers using Github webhooks	17
6.7	Monitoring logs using Elastic	18

Abstract

This report summarizes the process of building a calculator program using DevOps principles. The development process involved using various tools such as GitHub for version control, Jenkins for continuous integration and continuous deployment, Docker for containerization, and Ansible for infrastructure automation.

The report provides an overview of the development process, including the steps involved in setting up the project and configuring the various tools. It also highlights the benefits of using DevOps practices, such as improved collaboration, increased efficiency, and faster delivery of software products.

The report further discusses the challenges faced during the development process and the strategies employed to overcome them. It also provides an evaluation of the overall effectiveness of the DevOps practices employed in the project, as well as recommendations for future improvements.

Overall, the report demonstrates the value of DevOps principles in software development and how they can be effectively applied to build high-quality software products.

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 Codebase

Github Repository: <https://github.com/GSri30/ScientificCalculator>

Docker Hub Image: <https://hub.docker.com/r/gsri30/scientific-calculator>

3 What is DevOps?

DevOps is a software development methodology that emphasizes collaboration, automation, and continuous delivery. It involves bringing together development (Dev) and operations (Ops) teams to work together throughout the software development lifecycle. DevOps aims to increase the efficiency, reliability, and quality of software development by breaking down silos and streamlining processes.

The key principles of DevOps include:

- Continuous integration and delivery (CI/CD)
- Infrastructure as code (IaC)
- Automated testing and deployment
- Monitoring and feedback loops
- Collaboration and communication across teams

DevOps also involves the use of various tools and technologies to automate tasks, manage code, and deploy applications quickly and reliably.

4 Why DevOps?

DevOps is important for several reasons:

- **Speed and agility:** DevOps enables organizations to develop, test, and deploy software faster and more efficiently, reducing time to market and enabling rapid innovation.
- **Quality and reliability:** By automating testing and deployment processes, DevOps reduces the risk of errors and downtime, ensuring that software is reliable and meets user expectations.
- **Collaboration and communication:** DevOps brings together development and operations teams, promoting collaboration and communication, which leads to better outcomes and increased job satisfaction.
- **Scalability and flexibility:** DevOps allows organizations to scale their software development processes and infrastructure up or down quickly, depending on demand.
- **Competitive advantage:** Adopting DevOps practices can give organizations a competitive advantage by enabling them to innovate faster, improve customer experiences, and reduce costs.

Overall, DevOps is a powerful methodology that helps organizations deliver software more efficiently, with higher quality and reliability, while promoting collaboration and agility across teams.

5 Tools used

- **Source control management:** **Git** is a popular distributed version control system designed to handle everything from small to very large projects with speed and efficiency. With Git, developers can track changes to their code, collaborate with others, and easily roll back changes when necessary.
- **Testing:** **JUnit** is a popular testing framework for Java that allows developers to write and run unit tests for their code. With JUnit, developers can ensure that their code behaves as expected in different scenarios, catch and fix bugs early, and improve overall code quality.
- **Build:** **Apache Maven** is a popular build automation tool primarily used for Java projects to manage dependencies, build and deploy applications, and perform other project-related tasks. Maven helps developers automate the entire build process, making it easier to manage and maintain their projects.
- **Continuous Integration:** **Jenkins** is a popular open-source tool used for continuous integration and continuous delivery (CI/CD) in software development. With Jenkins, developers can automate the build, test, and deployment processes, allowing them to catch and fix issues early in the development cycle.
- **Containerize:** **Docker** is a popular platform used for containerization, which is the process of packaging an application and its dependencies into a lightweight, portable container. With Docker, developers can easily deploy their applications across different environments, from development to production, without worrying about compatibility issues or dependencies.
- **Configuration management:** **Ansible** is an open-source automation tool used for deployment, configuration management, and orchestration of IT infrastructure. With Ansible, developers can automate the deployment of applications and the configuration of infrastructure components, including servers, network devices, and databases.

- **Monitoring:** The **ELK stack** is a collection of open-source tools used for log management and analysis. ELK stands for Elasticsearch, Logstash, and Kibana, which are the three primary components of the stack. The ELK stack allows developers to collect, store, and analyze log data from different sources, providing valuable insights into the performance and behavior of their applications.
- **Triggering actions:** **GitHub webhooks** allow developers to receive real-time notifications about events that occur within their GitHub repositories. With webhooks, developers can automate the build, test, and deployment processes, trigger notifications, or integrate with other tools and services.
- **Tunneling:** **Ngrok** is a popular tool used to create secure tunnels to expose local servers or services to the internet. With Ngrok, developers can test their applications on different devices or share their local work with others without deploying to a public server.
- **Logging:** **Log4j2** is a popular logging framework for Java applications that provides a flexible and efficient logging mechanism. With Log4j2, developers can log messages at different levels of severity, filter and format log data, and configure the logging behavior of their applications.

6 Development

The following are the various steps involved (each is explained further in detail) for the development of the scientific calculator:

1. Creating a new maven project.
2. Implementing calculator in Java.
3. Writing JUnit tests for unit testing.
4. Source code management using Github.
5. Containerization using Dockerfile.
6. Configuration management using Ansible playbook.
7. Development pipeline using Jenkins.
 - (a) Stage-Git
 - (b) Stage-Maven build
 - (c) Stage-Docker build
 - (d) Stage-Dockerhub
 - (e) Stage-Ansible
8. Automatic triggers using Github Webhooks (along with ngrok tunneling)
9. Monitoring logs using ELK stack.

6.1 Creating, Implementing and Writing JUnits

I have used IntelliJ to create the maven project. [fig. (1)]

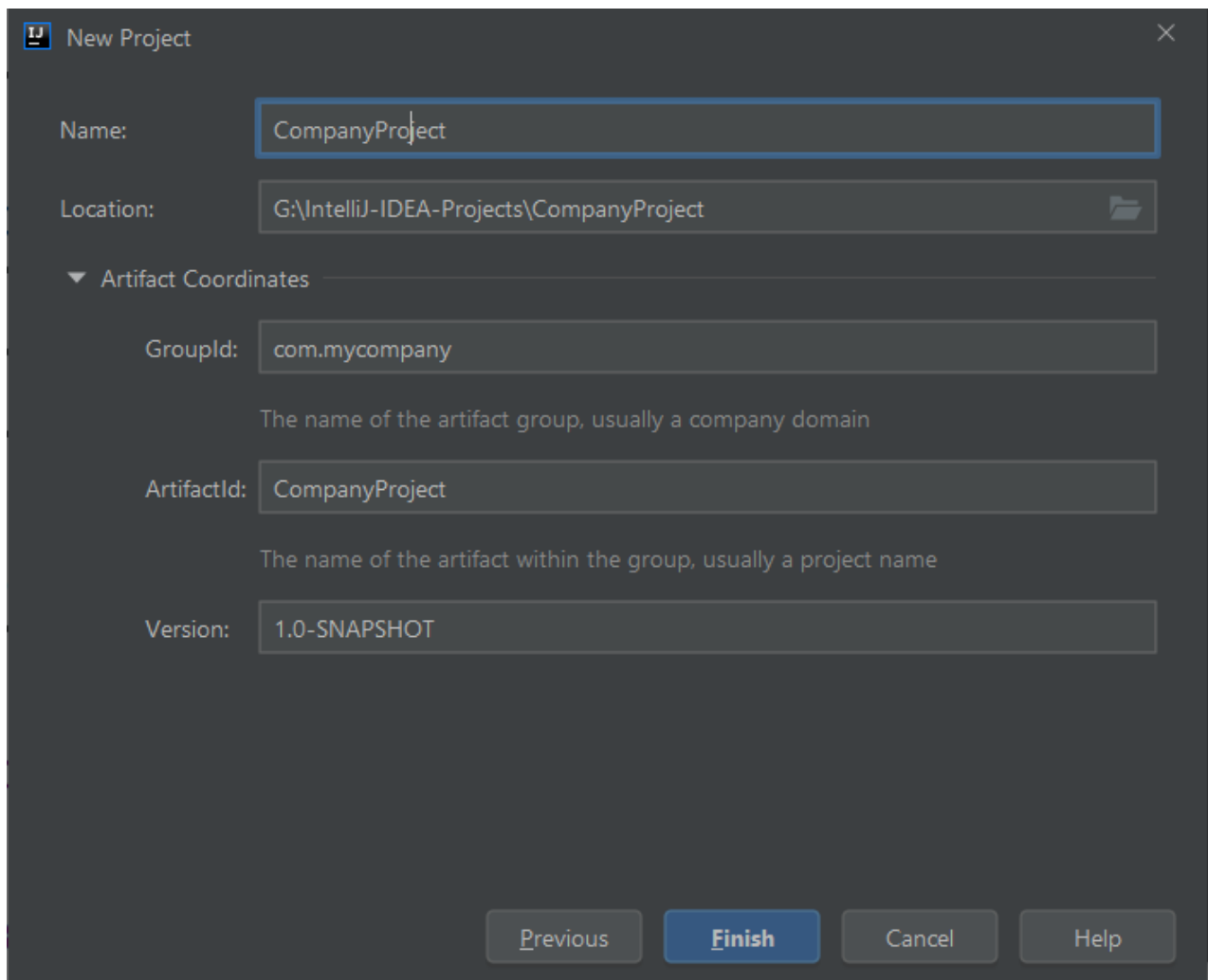
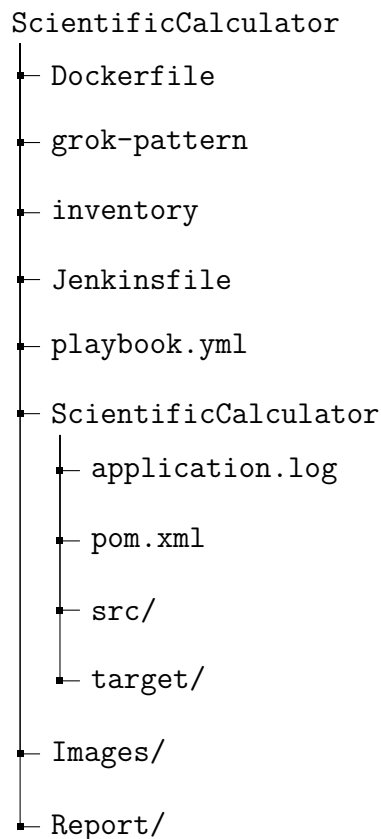
The image shows the 'New Project' dialog in IntelliJ IDEA. It has a dark theme. At the top, there's a title bar with the IntelliJ logo and the text 'New Project' and a close button. Below the title bar, there are several input fields. The 'Name' field is highlighted with a blue border and contains 'CompanyProject'. The 'Location' field contains 'G:\IntelliJ-IDEA-Projects\CompanyProject' and has a folder icon on the right. Below these is a section titled 'Artifact Coordinates' with a dropdown arrow. Under this section, there are three more input fields: 'GroupId' with 'com.mycompany', 'ArtifactId' with 'CompanyProject', and 'Version' with '1.0-SNAPSHOT'. Each of these fields has a descriptive text below it: 'The name of the artifact group, usually a company domain' for GroupId, 'The name of the artifact within the group, usually a project name' for ArtifactId, and no text for Version. At the bottom, there are four buttons: 'Previous', 'Finish' (highlighted in blue), 'Cancel', and 'Help'.

Figure 1: IntelliJ-Maven Project Create

The following is the project directory structure:



- **Dockerfile:** It contains a set of instructions used to build an openjdk:17 Docker image.
- **grok-pattern:** Contains the grok pattern of the logs generated. This pattern is used by elastic to visualize the logs once uploaded.
- **inventory:** The managed servers which are controlled by Ansible are defined in this inventory file.
- **Jenkinsfile:** It contains the definition of our Jenkins Pipeline. We have specifically 6 stages defined in this Jenkinsfile.
- **playbook.yml:** These contain a set of plays that are run in a specific order. We have specifically 3 plays defined in this playbook.
- **ScientificCalculator/src/:** Contains the source code to the implementation of the scientific calculator. It also contains the JUnit tests for the same program.
- **pom.xml:** This file contains the main config of the Java maven project, which is used to build and test the project.

After successfully writing the code for the required functionalities, we need to generate the JAR file for the project. For this, run [fig. (2)]

```
$ mvn clean install
```

After running this, the maven framework builds the project and generates a folder names 'target'. The JAR file can be found at

```
target / ScientificCalculator -1.0-SNAPSHOT-jar-with-dependencies.jar
```

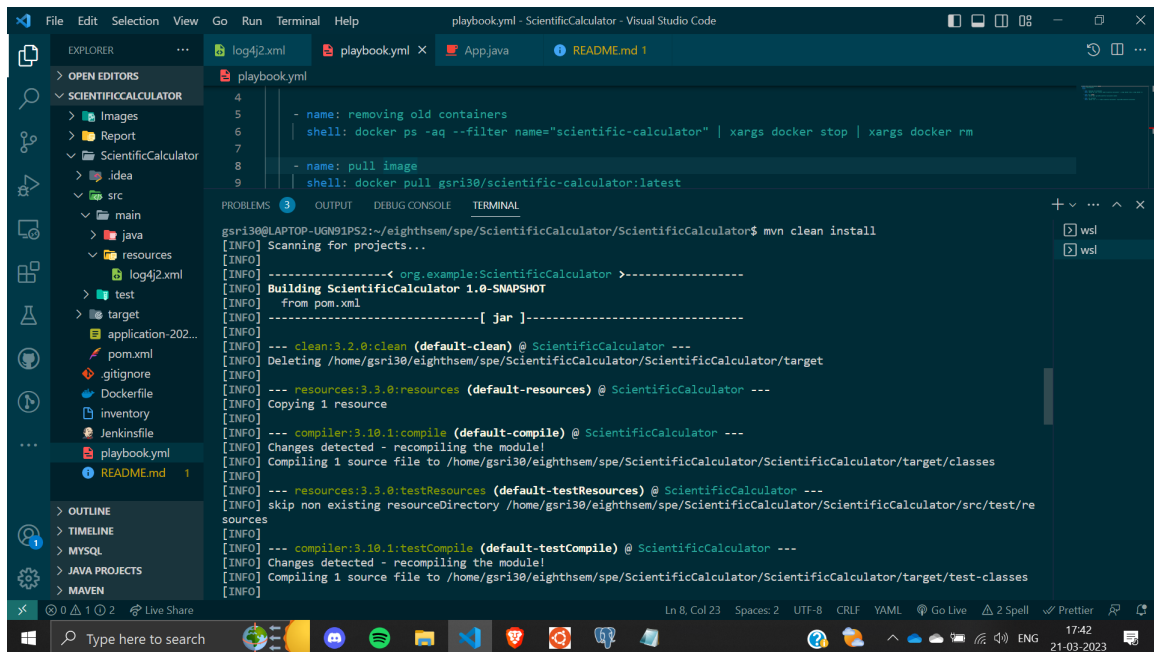


Figure 2: Command: mvn clean install

6.2 SCM using Github

- Source code management (SCM) is the process of tracking and managing changes to software code. SCM is essential for collaboration, version control, and maintaining the integrity and history of the codebase, ensuring the development process is efficient, reliable, and secure.
- Git is one of the most popular SCM tools that allow developers to track changes to their codebase, manage different versions of their code, and collaborate with other team members.
- With Git, developers can work on their code locally, create branches for different features or bug fixes, and push their changes to a central repository for other team members to review and merge.
- Git's distributed nature, ease of use, and powerful branching and merging capabilities make it a popular choice for managing source code in modern software development.
- In the current flow, a GitHub repository is created. Also, the locally created maven project is made git controlled and added this repository as an origin remote repository.
- All the local changes can be vied using the command [fig. (3)]

```
$ git status
```

- These changes can be made permanent by making a commit. The following are the series of commands for the same:

```
$ git add .
$ git commit -m "commit-message"
$ git push origin main
```

- All the different commits can be viewed in github in 'commits' section. [fig. (4)]


```

gsri30@LAPTOP-UGN91PS2:~/eighthsem/spe/ScientificCalculator$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ScientificCalculator/src/main/java/org/example/App.java
        modified:   ScientificCalculator/src/test/java/org/example/AppTest.java

no changes added to commit (use "git add" and/or "git commit -a")
gsri30@LAPTOP-UGN91PS2:~/eighthsem/spe/ScientificCalculator$

```

Figure 3: Command: git status

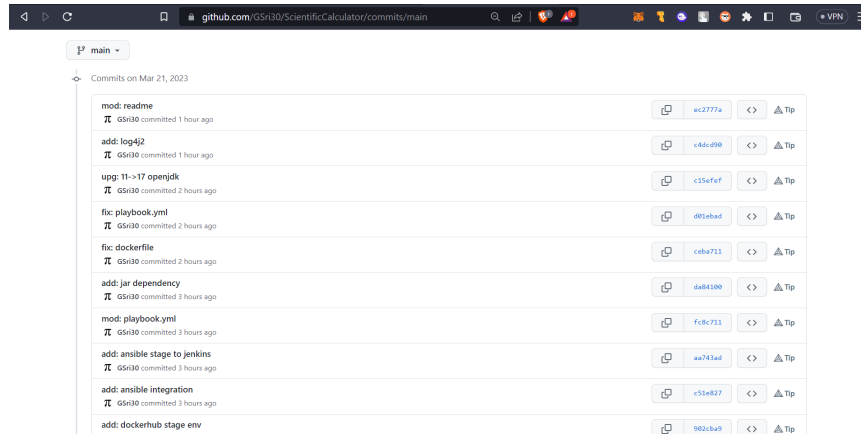


Figure 4: Git commits of the project

- Github branches can also be used to separate out the logic (main, test, dev, prod etc). I have used two branches, main and test. The test branch is extensively used to update the unit tests of the project. [fig. (5)]

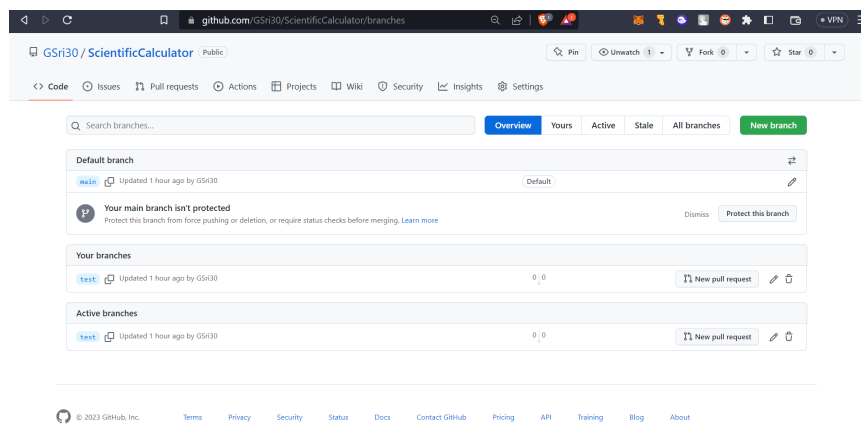


Figure 5: Git branches of the project

6.3 Containerization using Dockerfile

- Containerization is a process of packaging an application and its dependencies into a lightweight, standalone executable package called a container, that can run consistently across different computing environments.

- Dockerfile is a script used to define the contents, configuration, and behavior of a Docker container. It specifies the steps to create an image that can be used to instantiate container instances.

- The following is the configuration used

- OpenJDK:17 as the base image.

```
FROM openjdk:17-jdk-slim
```

- Change the working directory

```
WORKDIR ./
```

- Copy contents of target folder to working directory

```
ADD <target-path> ./
```

- Set the entrypoint to the JAR file in target folder

```
ENTRYPOINT [ "java", "-jar", "<JAR-file-path>" ]
```

6.4 Configuration management using Ansible

- Configuration management is the process of managing and maintaining the configuration of software systems, including hardware, software, and network components, to ensure that they function optimally and meet the desired state.
- Ansible is an open-source configuration management tool that automates the provisioning, configuration, and deployment of software and infrastructure.
- A playbook.yml is an Ansible script that defines a set of tasks to be executed on a target system, along with the necessary configuration parameters and variables. It describes the desired state of the system and instructs Ansible on how to achieve it.
- The following is the configuration used

- Task 1: Remove the currently running containers from host.

```
$ docker ps -aq --filter name=<name> \
| xargs docker stop \
| xargs docker rm
```

- Task 2: Pull the latest image from dockerHUB to host.

```
$ docker pull gsri30/<image:latest>
```

- Task 3: Running the latest pulled image inside a container.

```
$ docker run -it -d <image>\
--name <container-name>
```

6.5 Development pipeline using Jenkins

- Continuous Integration (CI) and Continuous Deployment (CD) are two essential practices in modern software development that involve automating the build, testing, and deployment of software applications.
- Continuous Integration is the process of frequently merging code changes from multiple developers into a shared repository, testing them to ensure they are working as expected, and then building and packaging the application.
- Continuous Deployment is the process of automating the deployment of the software to a production environment once it has been tested and verified. This is typically done using a series of automated scripts and tools that ensure the deployment process is fast, reliable, and consistent.
- Jenkins is an open-source automation server that is commonly used for implementing CI/CD pipelines. It allows developers to automate the entire software development process, from building and testing to deployment and monitoring. Jenkins supports a wide range of plugins and integrations, making it a flexible and customizable tool for teams of all sizes.

After successfully installing Jenkins, perform the following:

- Go to <http://localhost:8080/> and complete the initial configuration. [fig. (6)]

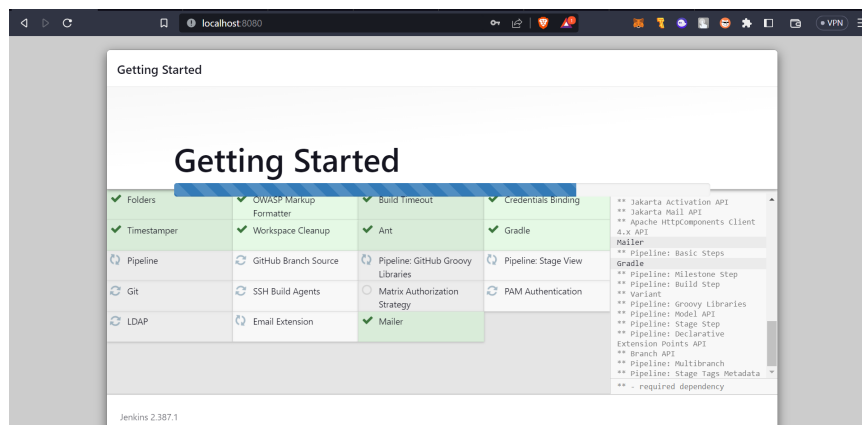


Figure 6: Jenkins getting started

- Go to
Manage Jenkins -> Plugin Manager -> Available Plugins
and enable the plugins mentioned in [fig. (7)].
- Then go to
Manage Jenkins -> Global Tool Configuration
and configure 'git' and 'maven'.
- Then go to
Manage Jenkins -> Manage Credentials
and configure the Github PAT and Docker HUB credentials. [fig. (8)]

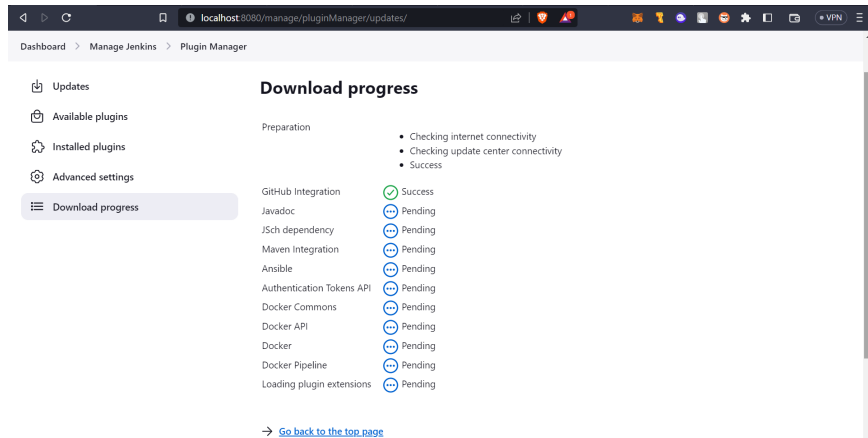


Figure 7: Jenkins plugins

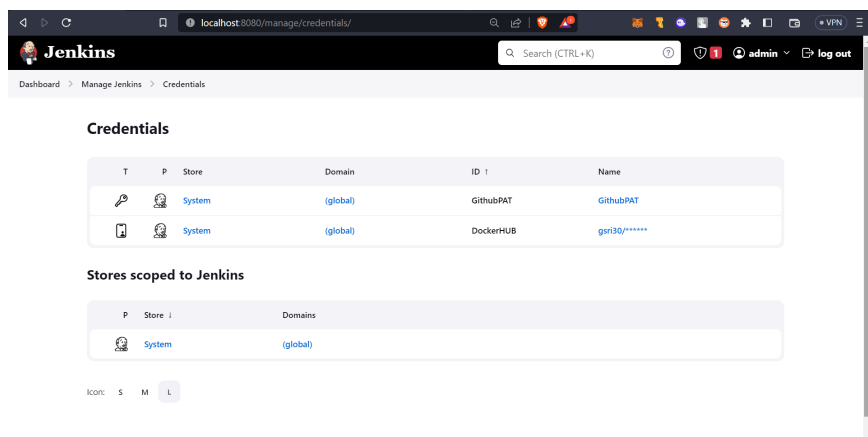


Figure 8: Jenkins credentials

- Then go to

Manage Jenkins -> Configure system

and configure Github server with the Github PAT credential created in the previous step.

Now after successfully setting up Jenkins, write a 'Jenkins' file specifying the different stages required. Our Jenkins file contains the following stages:

- Stage 1: Pulling the source code from github.
- Stage 2: Building the source code (using maven). This generates the JAR file in the 'target' folder.
- Stage 3: Building a new docker image. This builds a new docker image from the context provided by the Dockerfile in the repo pulled from Github in Stage 1.
- Stage 4: Pushing the updated image to Docker HUB. This publishes the generated image to your DockerHUB.
- Stage 5: Remove the docker image locally to avoid redundancy.
- Stage 6: Executing the ansible playbook.

Now after successfully setting up the Jenkins file, perform the following:

- Configure the project in Jenkins by going to

Configuration -> General -> Github project

and setting up the Github URL.

- Then go to [fig. (9)]

Configuration -> Pipeline -> Pipeline script from SCM

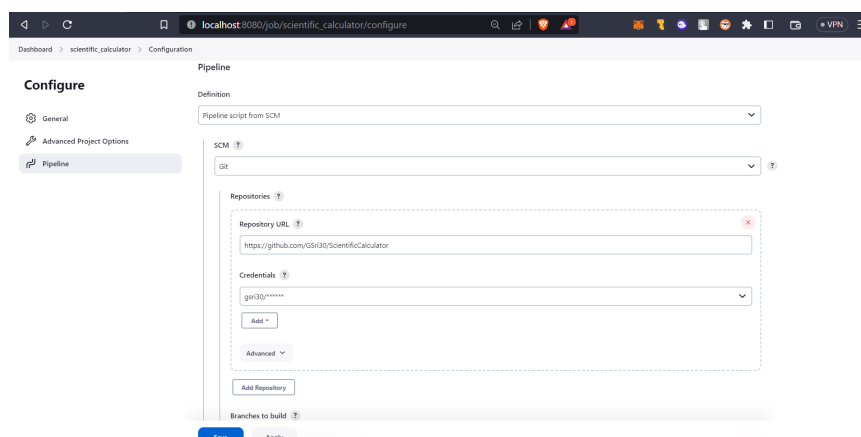


Figure 9: Jenkins pipeline from script SCM

- Then finally 'build' the project by clicking on

Build Now

in the side nav bar.

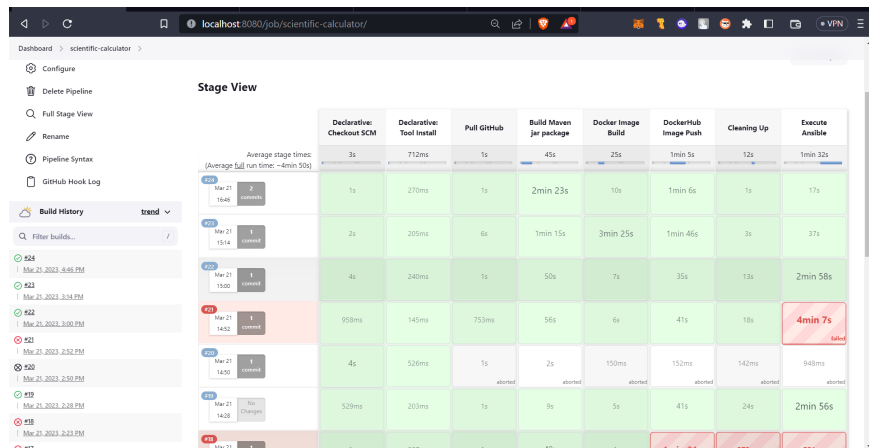


Figure 10: Jenkins Stage View

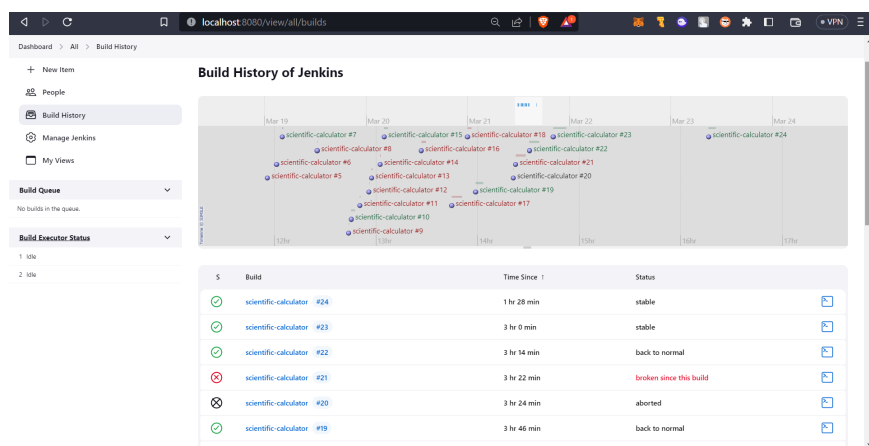


Figure 11: Jenkins Build History

- We get this Stage View after a series of builds [fig. (10)]
- We get this Build History [fig. (11)]
- We get this Build Time Trend [fig. (12)]

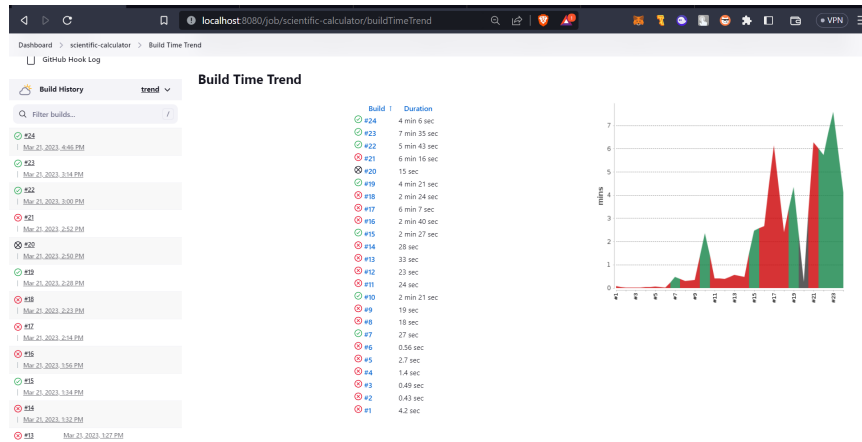


Figure 12: Jenkins Build Time Trend

After these steps, the following happen

- A docker image will be pushed to your DockerHUB registry. [fig. (13)]

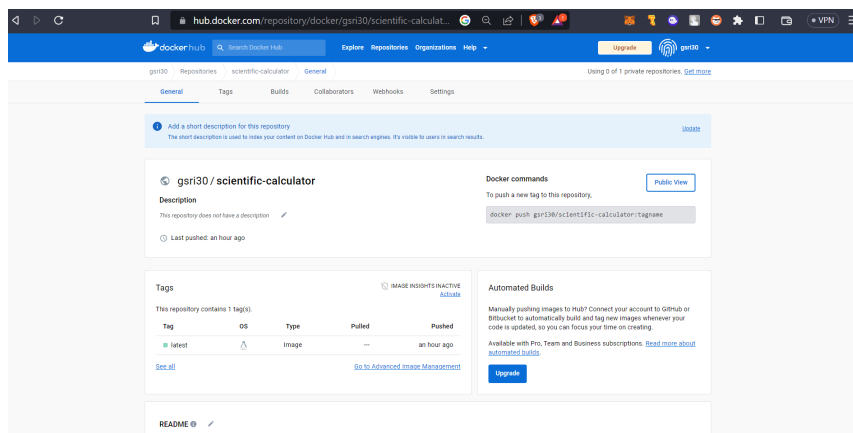


Figure 13: DockerHUB

- Docker container will be up locally running. It can be checked using the command

```
$ docker ps -a
```

- Alternatively, windows users can use a special tool called 'Docker Desktop' to access a GUI for the same.

Finally, the application can be run using the command

```
$ docker attach <container-name>
```

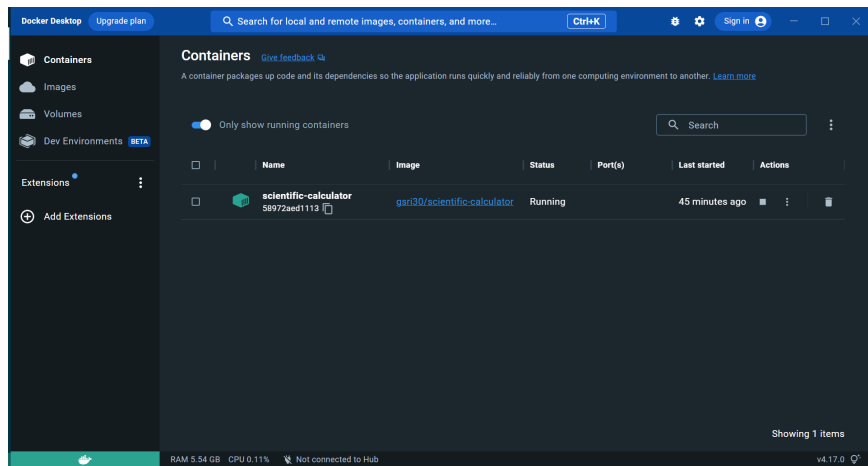


Figure 14: Docker Desktop: Docker containers

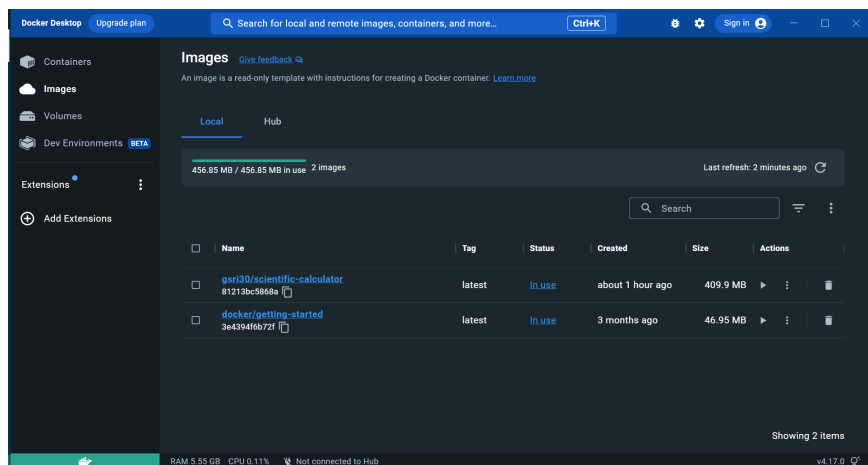


Figure 15: Docker Desktop: Docker Images

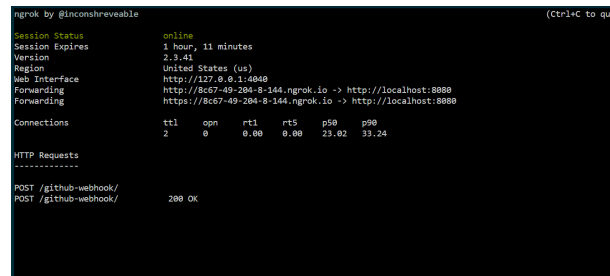
```
Scientific Calculator
1. Square root
2. Factorial
3. Logarithm (base e)
4. Power
0. Exit
Please enter your option:
4
Enter the numbers to find a power b :
3 5
19:54:32.357 [main] INFO org.example.App - Power -> Input: 3.0 5.0 -> Output: 243.0
Power:
Input: 3.0, 5.0
Output: 243.0
```

Figure 16: Calculator application

6.6 Automatic triggers using Github webhooks

- GitHub webhooks are automated HTTP callbacks triggered by events that occur within a GitHub repository.
- Ngrok is a secure tunneling service that exposes your local host to the internet, allowing external users to access it. It can be used for testing, demos, and other development scenarios.
- Hence, using ngrok, we tunnel our localhost to internet using the command [fig. (17)]

```
$ ngrok http 8080
```



```
ngrok by @inconshreveable (Ctrl+C to quit)
Session Status      online
Session Expires     1 hour, 11 minutes
Version             2.9.41
Region              United States (us)
Web Interface        http://127.0.0.1:4040
Forwarding           http://8c67-49-204-8-144.ngrok.io -> http://localhost:8080
                    https://8c67-49-204-8-144.ngrok.io -> http://localhost:8080

Connections
  ttl  opn  rt1  rt5  p50  p99
   2    0   0.00  0.00  23.02  33.24

HTTP Requests
-----
POST /github-webhook/
POST /github-webhook/      200 OK
```

Figure 17: ngrok

- Go to
Manage Jenkins → Configure system
and add the localhost exposed https url as Jenkins URL.
- Then go to
Configure → Build Triggers
and select
GitHub hook trigger for GITScm polling
- Then, add the GitHub webhook (using the exposed URL and your Github PAT)

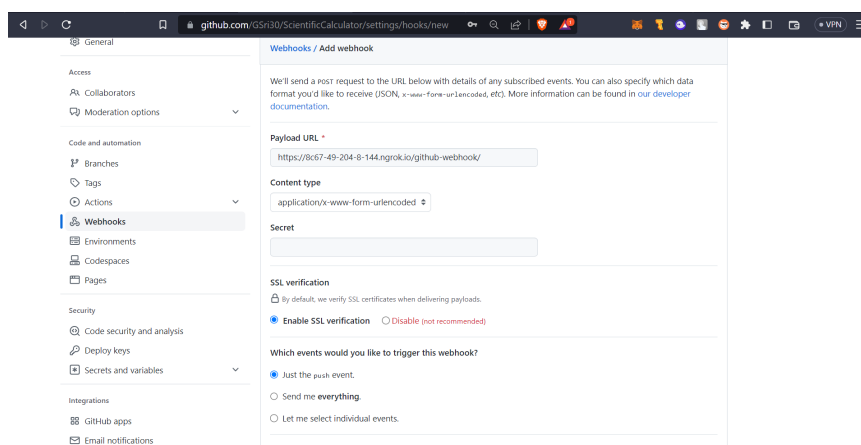


Figure 18: Github Webhook

6.7 Monitoring logs using Elastic

- ELK stack is a popular open-source log management and analysis platform consisting of three components:
 - Elasticsearch
 - Logstash
 - Kibana
- Some key points on ELK stack are:
 - It provides real-time visibility into the health and performance of complex distributed systems by centralizing logs and metrics from multiple sources.
 - It supports a wide range of log formats and protocols, making it easy to integrate with different types of applications and systems.
 - It allows users to search, filter, and analyze logs and metrics using a powerful query language and visualization tools.
 - It supports alerting and notification mechanisms for proactive monitoring and troubleshooting.
 - It can be extended with additional components and plugins to enhance its functionality, such as Beats for lightweight data shippers, and Logstash plugins for custom data processing.
- First, we need to have an account in www.elastic.co.
- Then, create your first deployment in elastic.
- After that, get your logs from the docker container running and upload them in

Deployment -> Analytics

-> Machine Learning -> Data Visualizer

-> File

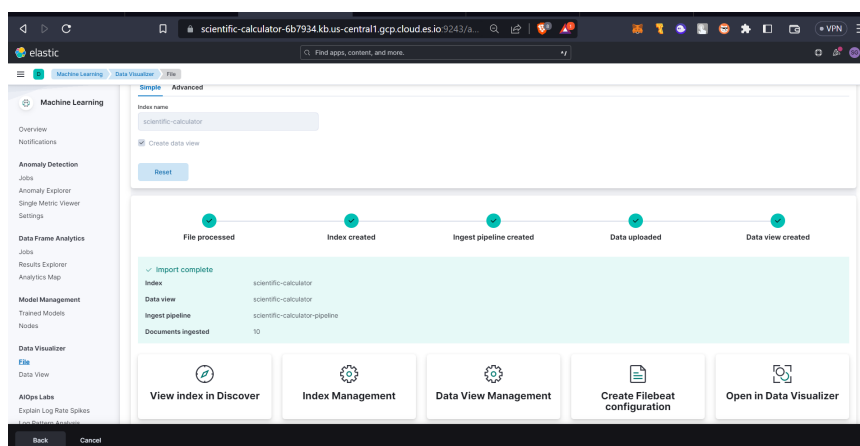


Figure 19: Elastic: Data Visualization Steps

- Change the 'grok-pattern' by overriding the settings [fig. (20) and fig. (21)]
- Click on Import.
- The following are the visualizations obtained.

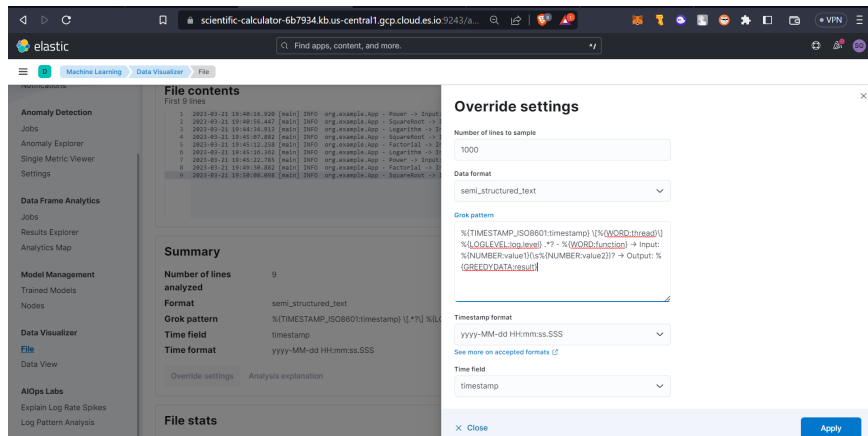


Figure 20: Elastic: grok pattern for logs

```
ScientificCalculator > application.log
1 2023-03-21 19:48:16.920 [main] INFO org.example.App - Power -> Input: 3.0 4.0 -> Output: 61.0
2 2023-03-21 19:48:16.947 [main] INFO org.example.App - SquareRoot -> Input: 180.0 -> Output: 3.1622776601683795
3 2023-03-21 19:48:16.973 [main] INFO org.example.App - Logarithm -> Input: 100.0 -> Output: 4.605170185988092
4 2023-03-21 19:48:16.999 [main] INFO org.example.App - SquareRoot -> Input: 180.0 -> Output: 10.0
5 2023-03-21 19:48:17.025 [main] INFO org.example.App - Factorial -> Input: 5.0 -> Output: 120.0
6 2023-03-21 19:48:17.051 [main] INFO org.example.App - Logarithm -> Input: 15.0 -> Output: 2.70805020110221
7 2023-03-21 19:48:17.077 [main] INFO org.example.App - Power -> Input: 4.0 2.0 -> Output: 16.0
8 2023-03-21 19:48:17.103 [main] INFO org.example.App - Factorial -> Input: 9.0 -> Output: 6.0
9 2023-03-21 19:48:17.129 [main] INFO org.example.App - SquareRoot -> Input: 6.0 -> Output: 2.449489742783178
10 2023-03-21 19:48:17.155 [main] INFO org.example.App - Power -> Input: 2.0 5.0 -> Output: 32.0
11 2023-03-21 19:48:17.181 [main] INFO org.example.App - SquareRoot -> Input: 10.0 -> Output: 3.1622776601683795
12 2023-03-21 19:48:17.207 [main] INFO org.example.App - Logarithm -> Input: 9.0 -> Output: 2.1972245773362196
13 2023-03-21 19:48:17.233 [main] INFO org.example.App - Power -> Input: 3.0 5.0 -> Output: 243.0
14
```

Figure 21: Elastic: logs

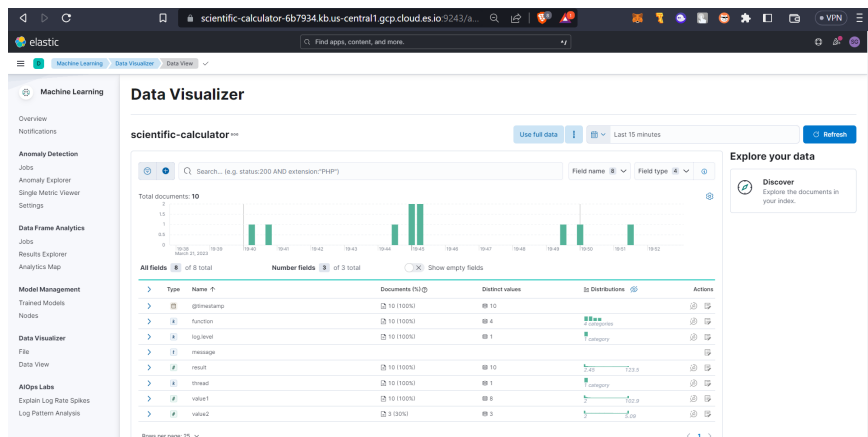


Figure 22: Elastic: Data Visualization

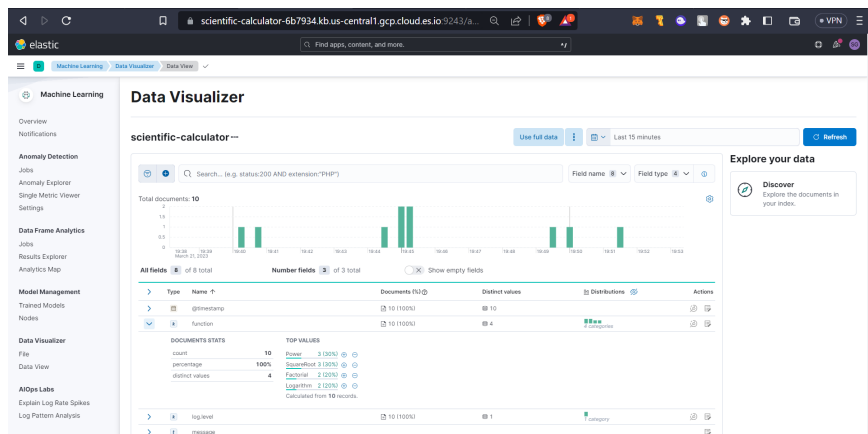


Figure 23: Elastic: function attribute visualization