CS 816 - Software Production Engineering

Mini Project - Scientific Calculator with DevOps

Sivani Channamsetty
IMT2019020

CONTENTS

Problem Statement

Create a scientific calculator program with user menu driven operations

- Square root function - √x
- Factorial function - x!
- Natural logarithm (base e) - ln(x)
- Power function - x^b

Introduction(What and Why of Devops)

DevOps is a combination of culture, methods, and tools that improves an organization's capacity to release applications more quickly. This model breaks the wall between the development team and the operaton team. The project's objective is to develop a calculator using deployment and integration automation tools. Source code management (SCM), continuous integration, continuous deployment, and configuration management for the entire source code are all included in the automation. This is achieved by putting into practise some of the devops tools that assist in the automation of the mentioned processes. The project entails developing a scientific calculator program that contains mathematical operations like square root calculation, factorial, natural logarithm, and power operations utilising devops tools like github, maven, junit, jenkins, ansible, and docker. The main objective of the project is to build a Jenkins pipeline while learning the CI/CD/CM devops concepts.

Frequently, there is friction between developers and operations teams because of the inherent differences in their roles. While developers aim to introduce new features to the application through changes, operations teams are cautious of any changes that could potentially harm the system, leading to application downtime. Organizations that embrace DevOps can enhance and expedite the evolution of software products more effectively than those using traditional software development approaches. DevOps serves as a bridge between development and operations within a company with the ultimate objective of boosting overall productivity. The DevOps team plays a critical role in bridging the gap between development and operations, thereby enabling testers and operators to work more efficiently.

Advantages of devops

- Improved deployment frequency
- Faster time to market
- Lower failure rate of new release
- Shortened lead time between fixes
- faster mean time to recovery in the event of a new release crashing

Tools used for

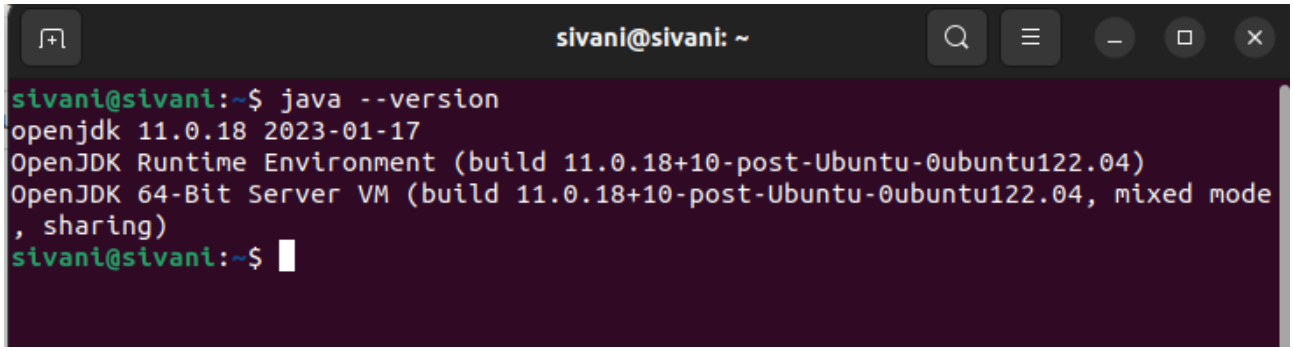
- *Source control management* : Git and GitHub
- *To perform WebHooks :* Ngrok
- *Automate the build process :* Github WebHooks
- *Code editor* : IntelliJ
- *Building and packaging the code* : Maven
- *Continuous Integration* : Jenkins

- *Containerising the code*: Docker
- *Configuration Management and Continuous Deployment* : Ansible
- *Monitoring* : ELK-stack(Elasticsearch)

Setting up tools

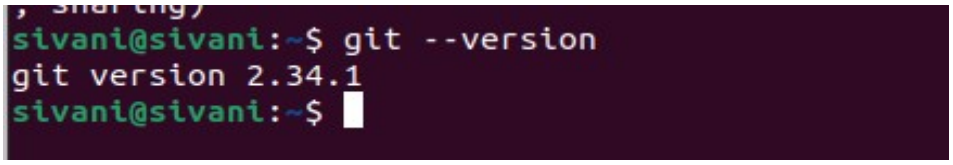
Java *:*

- *sudo apt-get update*
- *sudo apt install openjdk-11-jre-headless*
- *java --version*



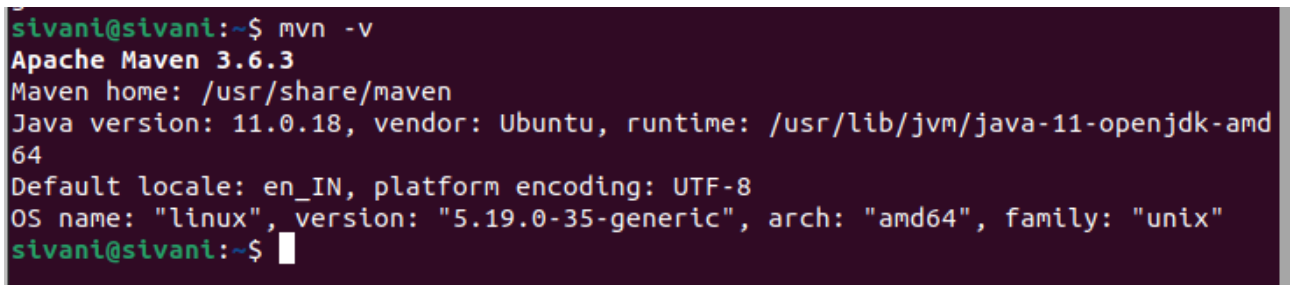
Git and Github*:*

- *sudo apt-get update*
- *sudo apt-get install git*
- *git –version*



Maven :

- *sudo apt-get update*
- *sudo apt-get install maven*
- *mvn -v*

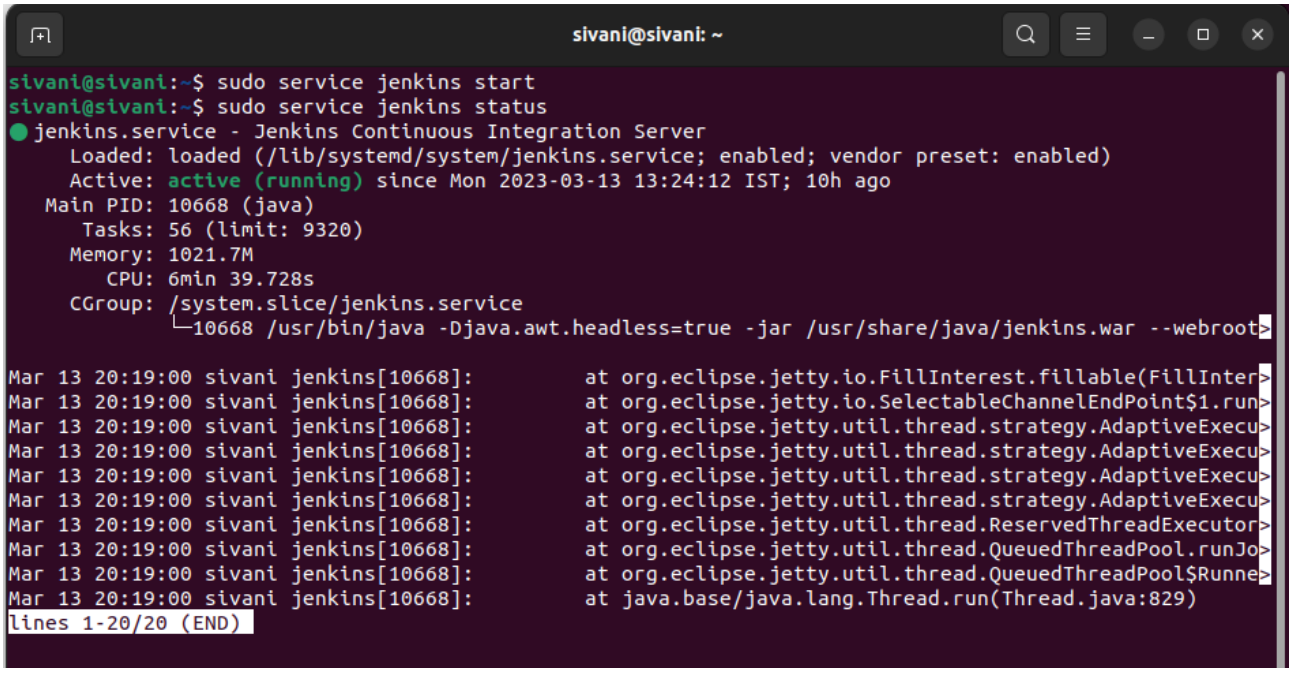


Intellij :

- For stable : *sudo snap install intellij-idea-ultimate --classic*
- For EAP builds : *sudo snap install intellij-idea-community --classic –edge*

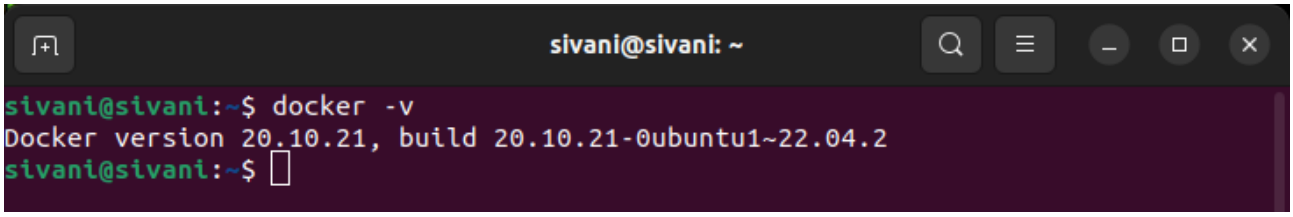
Jenkins :

- *wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -*
- *sudo sh -c 'echo deb https://pkg.jenkins.io/debian binary/ >/etc/apt/sources.list.d/jenkins.list'*
- *sudo apt install ca-certificates*
- *sudo apt-get update*
- *sudo apt-get install jenkins*
- *sudo service jenkins start*
- *sudo service jenkins status*
- *sudo cat /var/lib/jenkins/secrets/initialAdminPassword* (copy jenkins password)
- Go to *http://localhost:8080/* to use jenkins
- Paste the password there and start jenkins



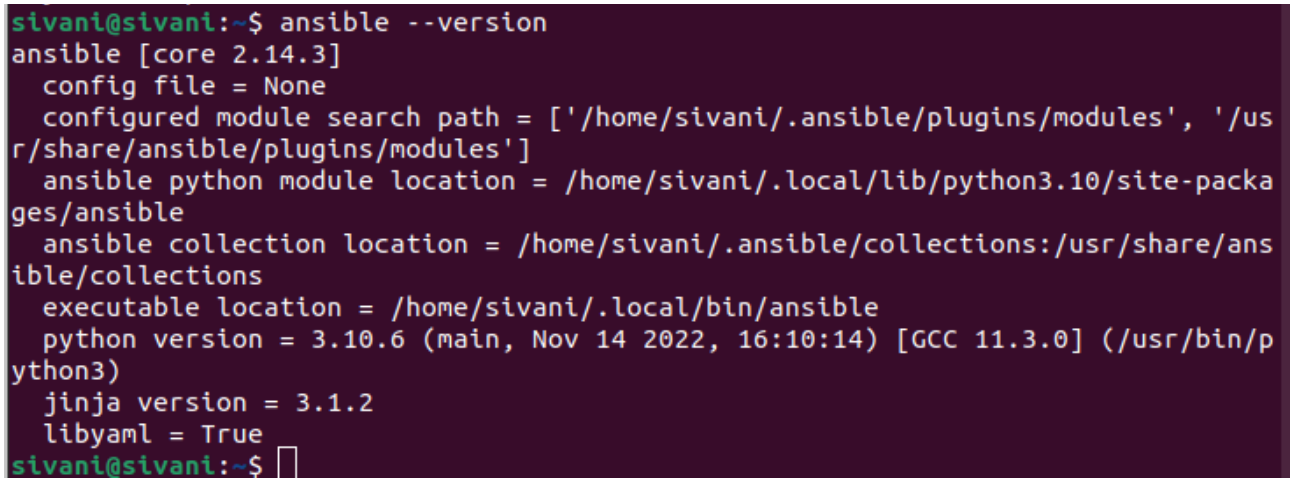
Docker :

- *sudo apt-get install docker.io*
- *sudo apt install apt-transport-https ca-certificates curl*
- *sudo apt-get install software-properties-common*
- *curl -fsSL https://get.docker.com -o get-socker.sh*
- *bash get-docker.sh*
- *docker -v*



Ansible :

- *sudo apt-get update*
- *sudo apt install openssh-server*
- *ssh-keygen -t rsa*
- *ssh-copy-id <username>@<localhost IP>*
- *sudo apt-get install ansible*
- *pip3 install docker*
- *pip3 install ansible*
- *ansible --version*



Ngrok :

- Sign up in *https://ngrok.com/*
- Download ngrok from*: https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.tgz*
- Then extract ngrok from the terminal*: sudo tar xvzf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin*
- Copy Authtoken from*: https://dashboard.ngrok.com/get-started/your-authtoken*
- Add Authtoken*: $ngrok authtoken <token>* in terminal
- Execute *$ngrok http 8080* in terminal*;* copy the public ip address for your local host.

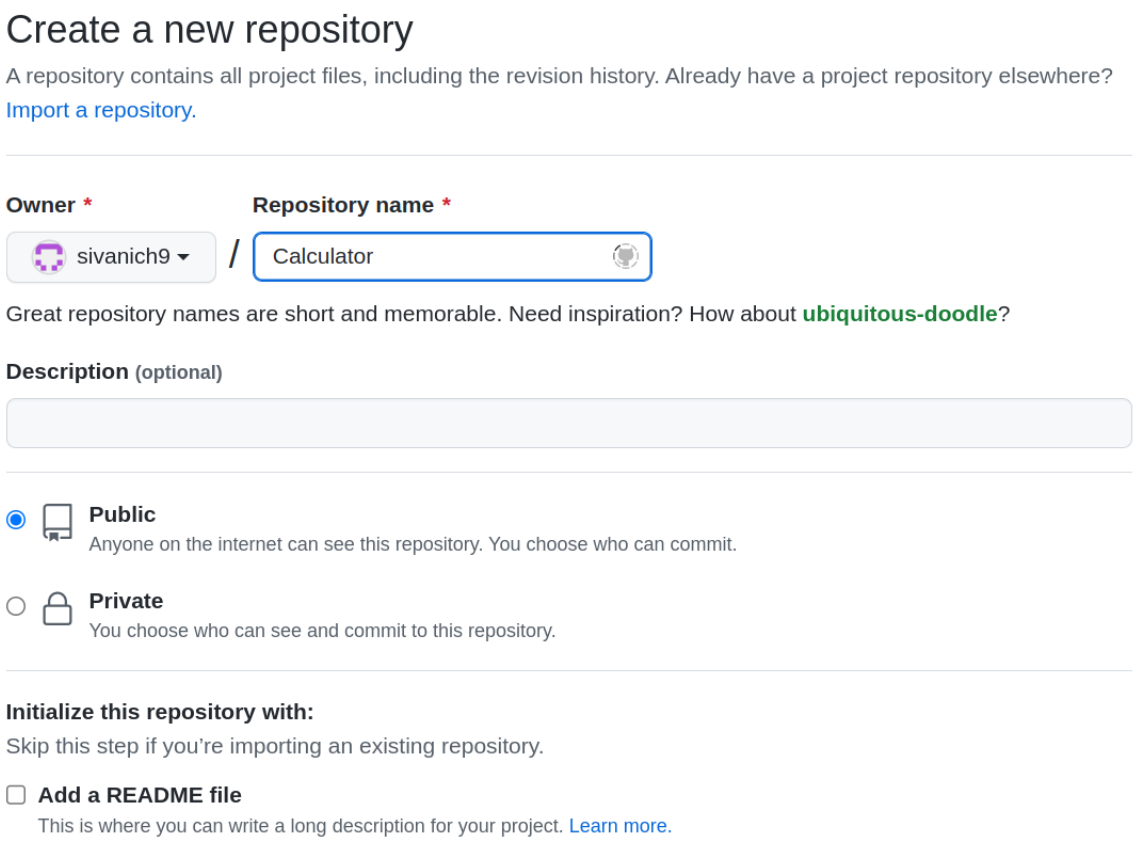
Project Workflow :

I am working this project on local repository(Calculator) directory. So no need to add paths in commands. All the commands are without path of the directory.

Git and Github :

For version control system we use git and github. Github helps in automation through jenkins integration. Git is an invaluable tool for implementing version control systems, enabling the preservation of all code versions and facilitating easy access to the required version when necessary. To leverage git in any project folder, the folder must first be initialized as a git repository. While git can be used locally, we push the code to a cloud-based management platform such as Github. This platform allows anyone with access, subject to the repository's privacy settings, to make modifications and contribute to the code. First create a github repository and clone that repository to local repository. We add code and make changes to the code in local repository. For this we use *Intellij* code editor. After that we add code into the staging area and then commit changes. Finally we push the code to the remote repository.

- To initialize repository in local : *git init*
- To see status of the repository : *git status*
- Clone remote repository into local *: git clone <remote repository url> <where to clone>*
- After adding files in local repository, we add to the staging area : *git add -A*
- Commit the changes : *git commit -m "msg"*
- To check log like who made changes and in which branch changes are made : *git log*
- Push to the remote repository : git push
- Add the link to the current local repository with the use of git remote add – *git remote add origin <link to github repo>*



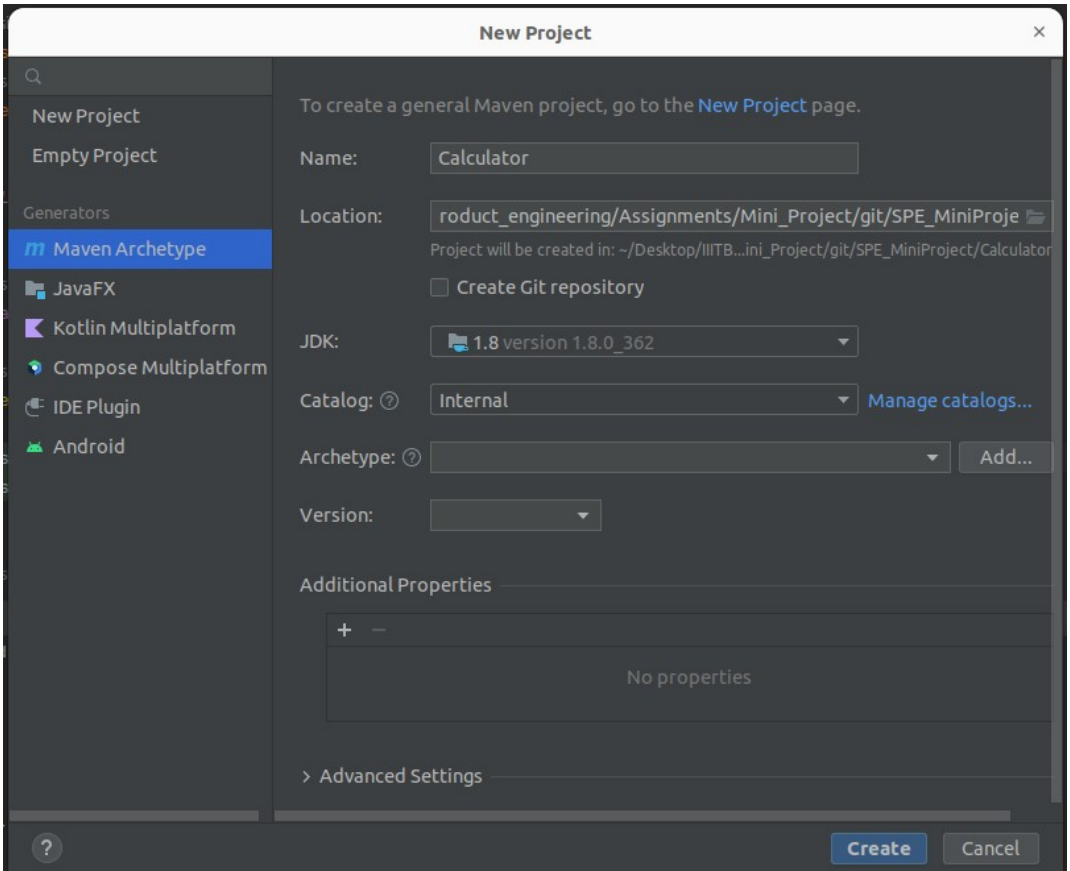
Intellij , Maven and JUnit :

We use java to build our project as it is system independent and we use IDE Intellij. Maven is a popular build automation tool that's mostly used with Java projects. This tool is based on Java and is used for developing applications. It allows us to include dependencies and create a jar file, which is a snapshot of our project that can be executed on any machine. By controlling project dependencies, handling compilation, packaging the code into a distributable format, and producing documentation, it simplifies the building process. Maven defines the project structure, dependencies, and build process using a Project Object Model (POM). The project's name, version, description, and dependencies are among the details found in the POM file (POM.xml). It is simple to obtain and manage project dependencies, including third-party libraries, using Maven's dependency management tool. We add dependencies of our project in pom.xml file. Maven simplifies the management of dependencies and assists users in transforming a project into a JAR file that can be easily transported and executed on different systems. Developers can carry out a variety of operations with the help of Maven's plugins, including deploying the program, creating code documentation, and performing tests.

We test our code using JUnit. The project uses JUnit, a user-friendly unit testing tool. In my project, I've defined three tags: @before, @after, and @test. The @test tag is used for the annotation to specify that a method is a test method. Every time one of the @Test methods is called, @Before is done first. Following each @Test method, an @After annotation is executed. With this, a Test setup may be repeatedly set up and cleaned up after each test. To determine whether the expected value and the actual value are the same, we utilise assertEquals for true positive test cases. If they are not same, we have messages in assertEquals statement which gives messages that will be displayed if the test fails. Such messages help in identifying and resolving the failures quickly. Similarly we use assertNotEquals for false positive test cases.

- *mvn clean install* – To build the project
- The build project is stored in target folder of maven project
- *java -cp <jar-name> <class-path>* - To build the jar file
- *mvn clean test* (or) *maven test* - Maven runs the JUnit tests

Create new maven project in Intellij



Add JUnit dependencies in pom.xml file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>Calculator</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
        </dependency>
    </dependencies>
```

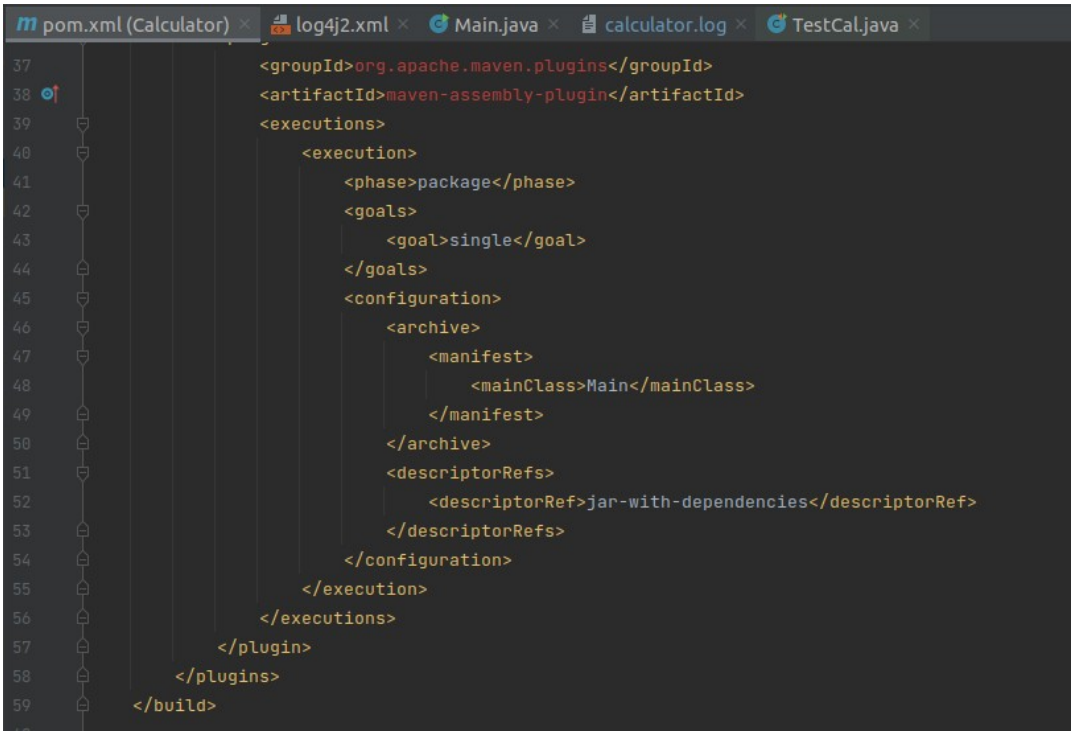
Add log4j dependencies in pom.xml file

logging is very important as using logs we can see if any any undesirable behaviour is occurring, for backup purposes and lot more. For this feature we will be using Log4J with our code. Log4j is a Java-based logging utility that is widely used in software development to log information at various levels of severity. In Maven, Log4j can be included as a dependency in the project's pom.xml file, which allows developers to easily configure logging settings and output locations. By using Log4j with Maven, developers can efficiently debug and troubleshoot their applications, as well as monitor performance and diagnose errors.

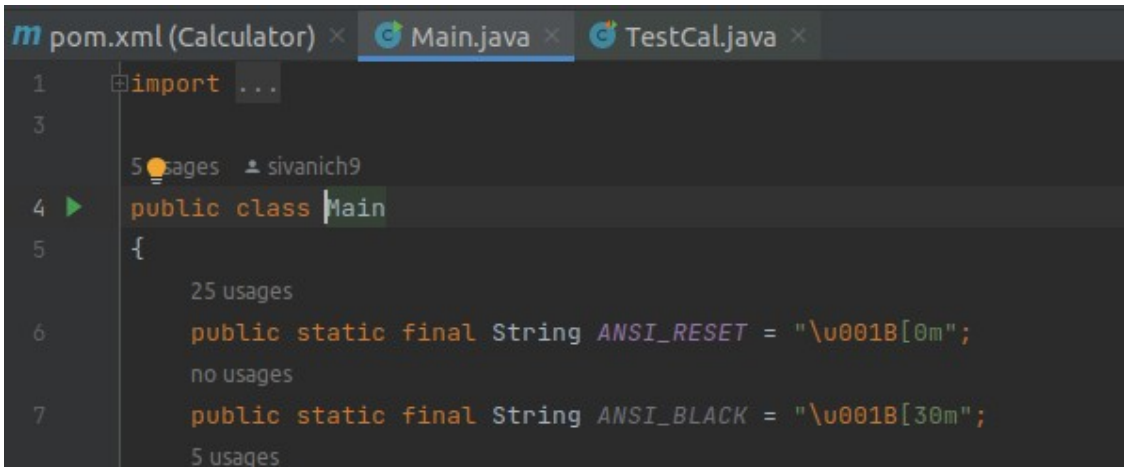
```xml
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.14.0</version>
        </dependency>
    </dependencies>
```

In essence, in the following step we are instructing the process to include all the dependencies within the jar file during its creation, eliminating the need to separately install these dependencies

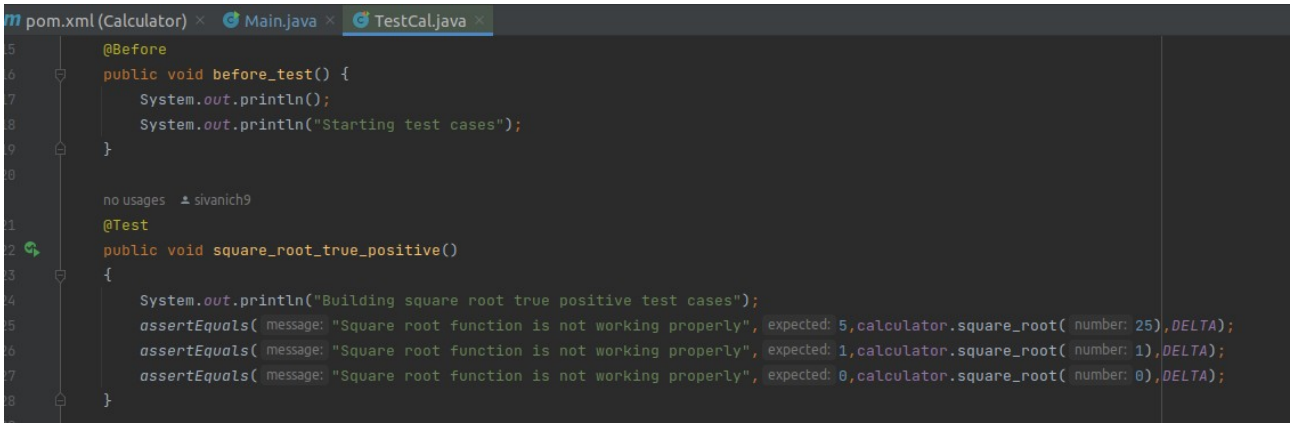
on the system where the jar file will be executed. We are accomplishing this by utilizing the Maven Assembly Plugin.



Code in main.java file



Code in the test.java file



we have to create **“**log4j.xml” in **“**<project-folder>/src/main/resources/log4j2.xml”.

pom.xml (Calculator) | log4j2.xml | Main.java | calculator.log | TestCal.java

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="ConsoleAppender" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS}{IST} [%F] %-5level %logger{36} - %msg%n"/>
        </Console>
        <File name="FileAppender" fileName="calculator.log" immediateFlush="false" append="true">
            <PatternLayout pattern="%d{dd/MM/yyy:HH:mm:ss SSS}{IST} [%F] [%level] %logger{36} %msg%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="ConsoleAppender"/>
            <AppenderRef ref="FileAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

In *Appenders*, in *PatternLayout* we have format like *%d{dd/MM/yyy:HH:mm:ss SSS}{IST} [%F] [%level] %logger{36} %msg%n*

Denoting

1. First part is date
2. [%F] denotes the class which calls it
3. “%level” denotes type of log like INFO, ERROR
4. “%logger{36}” means in which class the logger object was created
5. “%msg” is the message passed to the logger while logging.

Add logger method in main.java and logger.info in methods of main.java

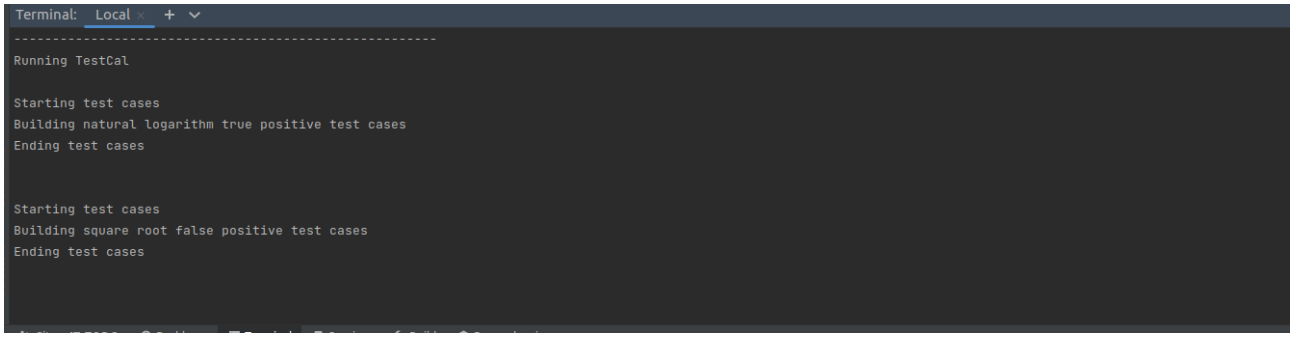
pom.xml (Calculator) | log4j2.xml | Main.java | calculator.log | TestCal.java

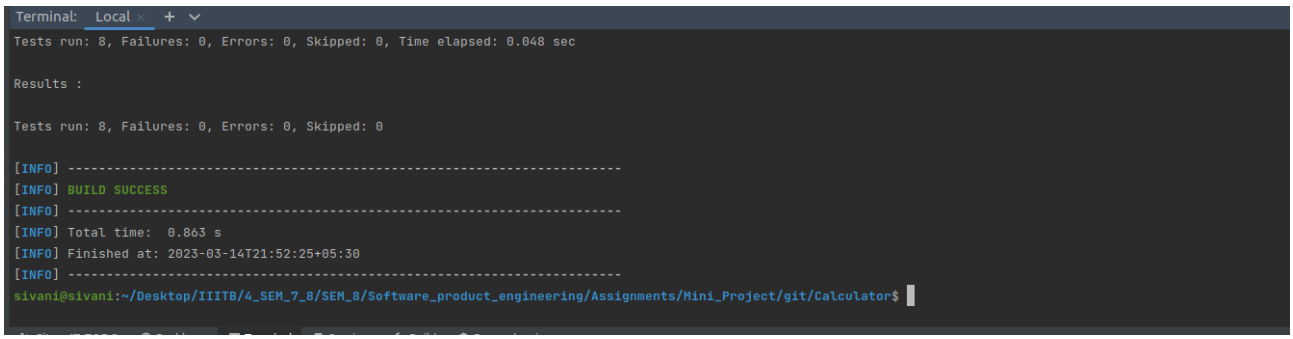
```
12 usages
private static final Logger logger = LogManager.getLogger(Main.class);
```

n.xml (Calculator) | log4j2.xml | Main.java | calculator.log | TestCal.java

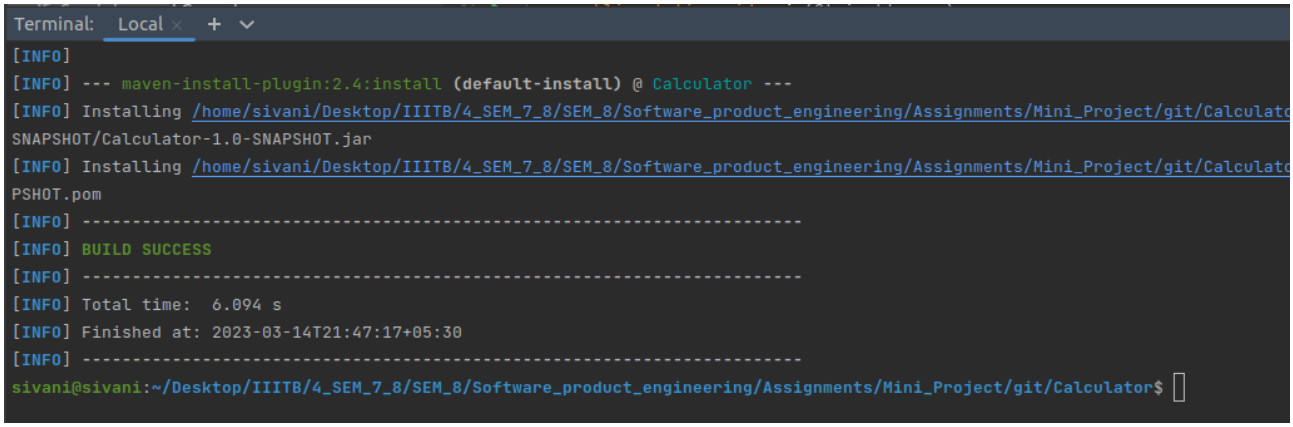
```
}

7 usages  sivanich9
public double square_root(double number)
{
    try
    {
        logger.info("[SQRT] - " + number);
        if(number < 0)
        {
            logger.error("[ERROR - SQRT] - Square root function is given -ve number");
            throw new IllegalArgumentException(ANSI_RED+"Cannot calculate square root of negative number"+ANSI_RESET);
        }
        logger.info("[RESULT - SQRT] - " + Math.sqrt(number));
        return Math.sqrt(number);
    }
    catch (IllegalArgumentException e)
    {
        System.err.println(e.getMessage());
        return Double.NaN;
    }
}
```

Run test cases using *mvn test*

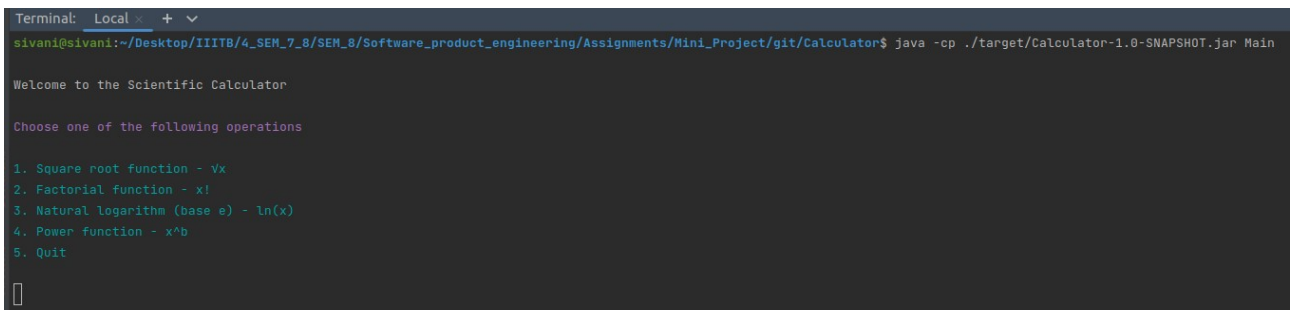




Build the project using *mvn clean install*



Run the jar file



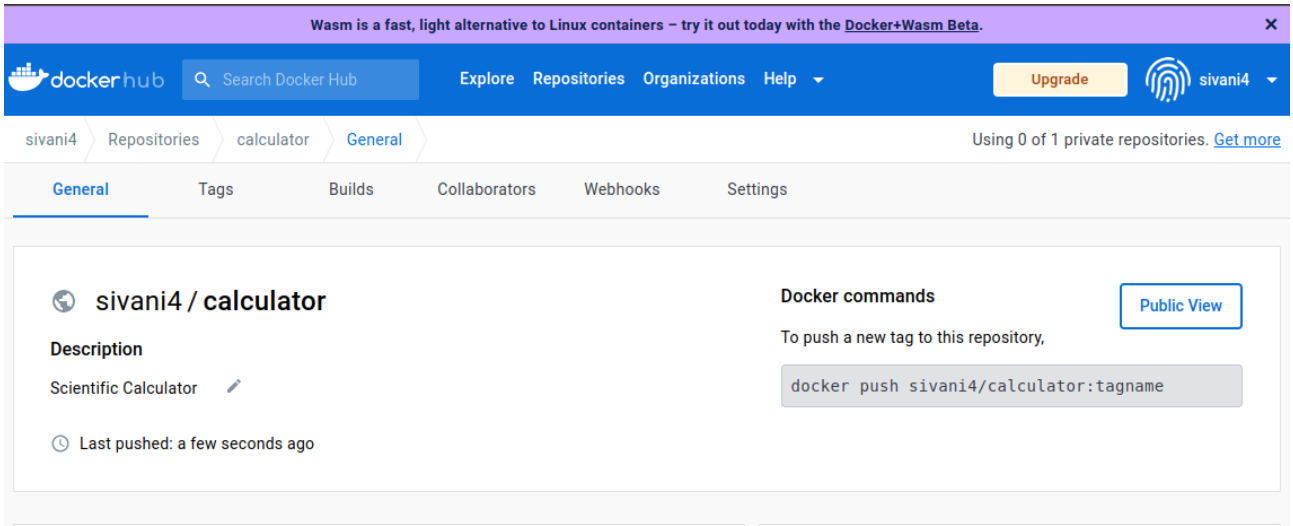
Add changes to the github repository

```
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ git add -A
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ git commit -m "added changes"
[main 0853222] added changes
 7 files changed, 10 insertions(+), 8 deletions(-)
 rewrite target/Calculator-1.0-SNAPSHOT.jar (83%)
 rewrite target/classes/Main.class (85%)
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ git push
Username for 'https://github.com': sivanich9
Password for 'https://sivanich9@github.com':
Enumerating objects: 33, done.
Counting objects: 100% (33/33), done.
Delta compression using up to 8 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (17/17), 6.15 KiB | 899.00 KiB/s, done.
Total 17 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 7 local objects.
To https://github.com/sivanich9/Calculator.git
   ee012eb..0853222  main -> main
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$
```

Docker :

Docker is a popular open-source containerization platform that allows developers to package an application and its dependencies into a single, portable container that can run consistently across different computing environments.(e.g., development, testing, and production). Because of the many dependencies and versions of dependencies that must be supported, there will be a number of issues to be resolved when the product is delivered. The clients will find it very difficult to resolve. Docker is helpful in this situation since it enables us to correctly install all of the dependencies and then compress them into an image. As the image was uploaded to DockerHub for a similar reason as github, anyone can now download the image and use it for their own projects. Docker is used to make images through containerization.

First we sign in to the dockerhub and create one repository named calculator



This is the docker file. On building this docker file, we will get docker image. Building this file, it helps us to install all dependencies and compress them into an image.

The following is the meaning of the Dockerfile commands:

- FROM: Imports the base image openjdk11 to create a new image.
- COPY: Copies a file from the current directory of the Dockerfile into the root directory of the image.
- WORKDIR: Changes the current working directory within the image.

- CMD: Runs a command inside the image.

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

We build docker file to get a docker image. Command to build is : *sudo docker build -t <docker-username/reponame>* .

```
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ sudo docker build -t sivani4/calculator .
Sending build context to Docker daemon  337.4kB
Step 1/4 : FROM openjdk:11
 ---> 47a932d998b7
Step 2/4 : COPY ./target/Calculator-1.0-SNAPSHOT.jar ./
 ---> Using cache
 ---> 6260bf113dc4
Step 3/4 : WORKDIR ./
 ---> Using cache
 ---> 75f0084decd5
Step 4/4 : CMD ["java", "-cp", "Calculator-1.0-SNAPSHOT.jar", "Main"]
 ---> Using cache
 ---> 812ed1e5214d
Successfully built 812ed1e5214d
Successfully tagged sivani4/calculator:latest
```

By using *sudo docker images*, we can see docker image sivani4/calculator is created.

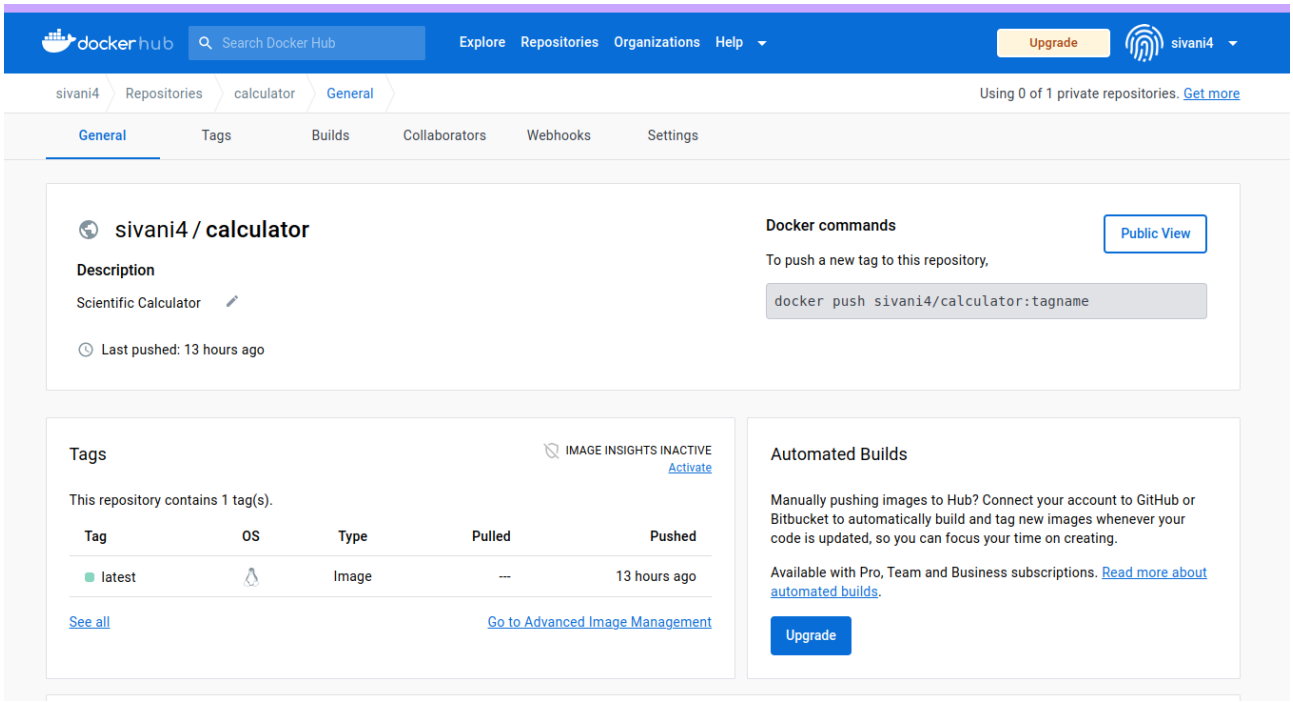
```
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ sudo docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
calculator          latest    812ed1e5214d   5 minutes ago   654MB
sivani4/calculator  latest    812ed1e5214d   5 minutes ago   654MB
my-java-app         latest    b884a62f7595   12 hours ago    654MB
<none>              <none>    3010c52d172d   12 hours ago    526MB
<none>              <none>    fdf8d3e36db8   13 hours ago    526MB
openjdk             8         b273004037cc   7 months ago    526MB
openjdk             11        47a932d998b7   7 months ago    654MB
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$
```

To push our docker image to dockerhub, we first login into our dockerhub using *sudo docker login*

```
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ sudo docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: sivani4
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$
```

Then we push our docker image to dockerhub using *sudo docker push <docker-username/reponame:latest>*

```
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ sudo docker push sivani4/calculator:latest
The push refers to repository [docker.io/sivani4/calculator]
8bf315119cd9: Pushed
7b7f3078e1db: Mounted from library/openjdk
826c3ddbb29c: Mounted from library/openjdk
b626401ef603: Mounted from library/openjdk
9b55156abf26: Mounted from library/openjdk
293d5db30c9f: Mounted from library/openjdk
03127cdb479b: Mounted from library/openjdk
9c742cd6c7a5: Mounted from library/openjdk
latest: digest: sha256:852ec9a6b849241971a34ea2816c10d860f1b0023f6d2fe9f0cafc88aeb69824 size: 2003
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$
```



We run the container using command *docker run -it <docker-username/repo-name:tagname>*

Ansible :

Continuous deployment is a technique for automatically distributing software to lots of client computers. Ansible is a configuration management tool. However anything may be sent to a lot of "controlled hosts" using this method. On the control node, Ansible is installed and maintained. All copies of your Ansible project files and configuration data are kept on a control node. Managed hosts are computers that deliver the application or infrastructure as code. Ansible serves as a central distributor to all managed hosts from the control node. It is possible to view a list of managed hosts in the inventory file. Here, I'm using my localhost as the host rather than another VM or machine.

We create p2.yml. It is the playbook that contains the configuration and infrastructure as code.

Inventory file has the list of target servers we want to manage. I am using my localhost only.

p2.yml

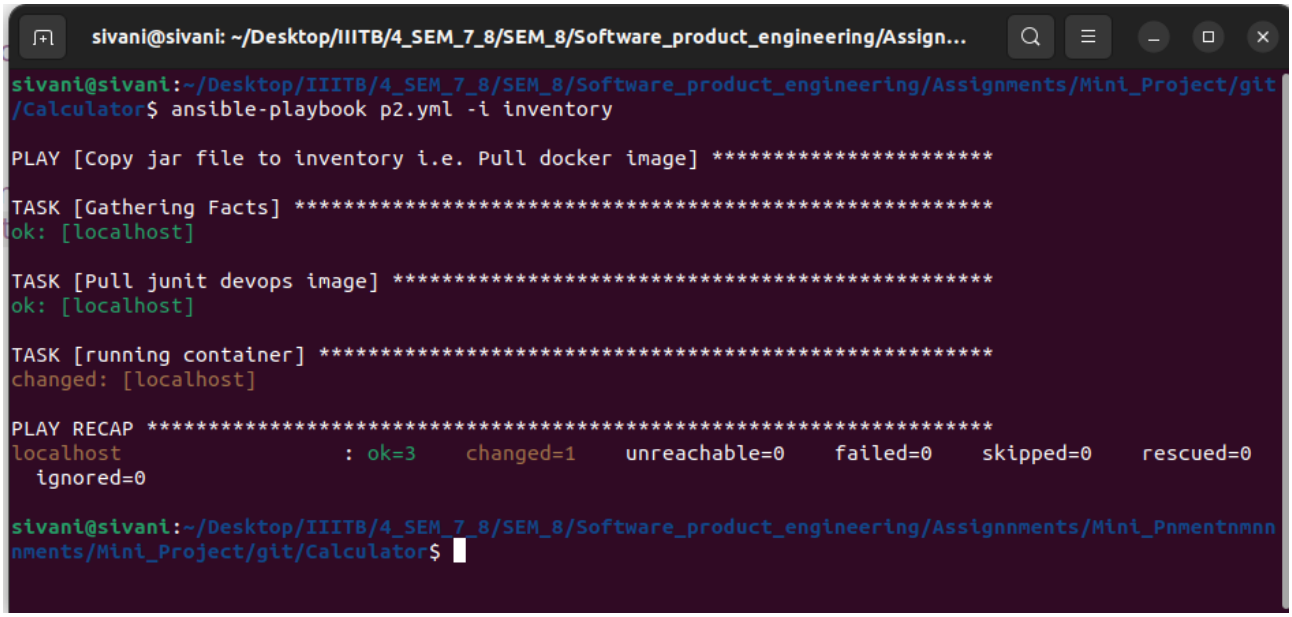
```
---
- name: Copy jar file to inventory i.e. Pull docker image
  hosts: all
  tasks:
          - name: Pull junit devops image
            docker_image:
                  name: sivani4/calculator
                  source: pull


          - name: running container
            shell: docker run -it -d sivani4/calculator /bin/bash
```

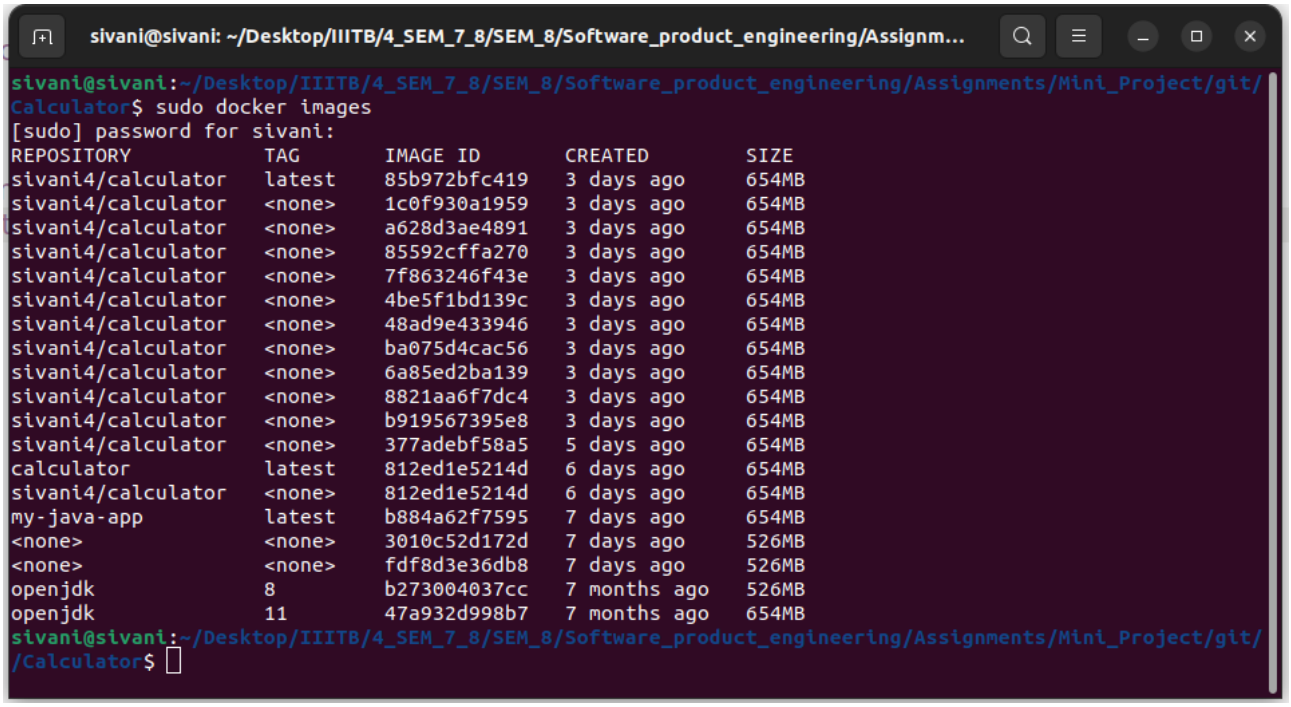
inventory

```
localhost ansible_user=sivani ansible_connection=local
```

The command "*ansible-playbook p2.yml -i inventory*" will run an Ansible playbook called "p2.yml" using an inventory file called "inventory".

- “ansible-playbook" is the command that tells Ansible to execute a playbook.
- "p2.yml" is the name of the playbook file that will be executed.
- "-i inventory" specifies the inventory file that Ansible should use to determine which hosts to run the playbook on. The inventory file contains a list of hosts and their associated configuration information, such as IP addresses and connection details.





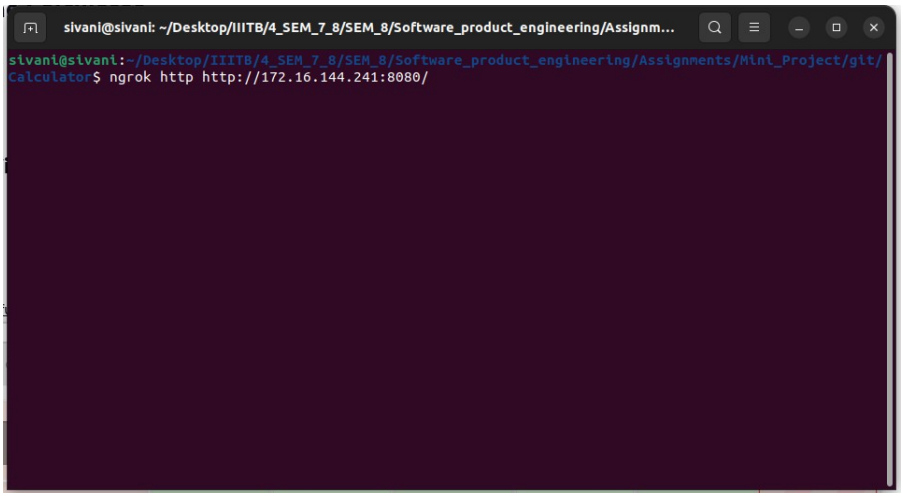
Ngrok and Github WebHooks :

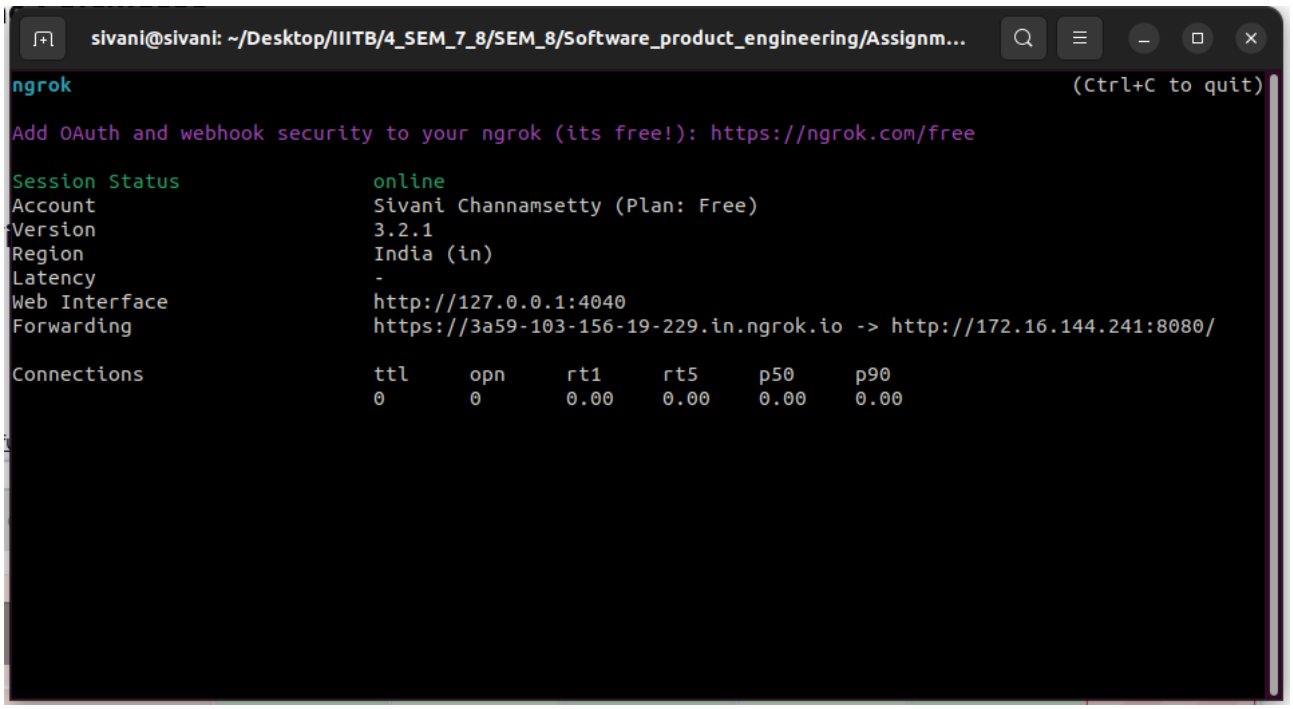
Whenever the developer builds the code and commit it and push that to Github, webhooks automatically build the code in Jenkinks. To perform a webhook, the private IP address of the local machine needs to be converted to a public IP address.

Automated messages known as "webhooks" are triggered when changes occur. Specifically, in our situation, updates made to the GitHub repository will activate the Jenkins pipeline through the webhook.

Ngrok is a tool that establishes secure tunnels to connect local servers located behind Network Address Translation (NAT) and firewalls to the internet. It features a web interface that provides real-time monitoring of any HTTP traffic that passes through the tunnels. This tool enables users to access the internet via a web server running on their local system, with the port on which the web server is listening to being specified to ngrok.

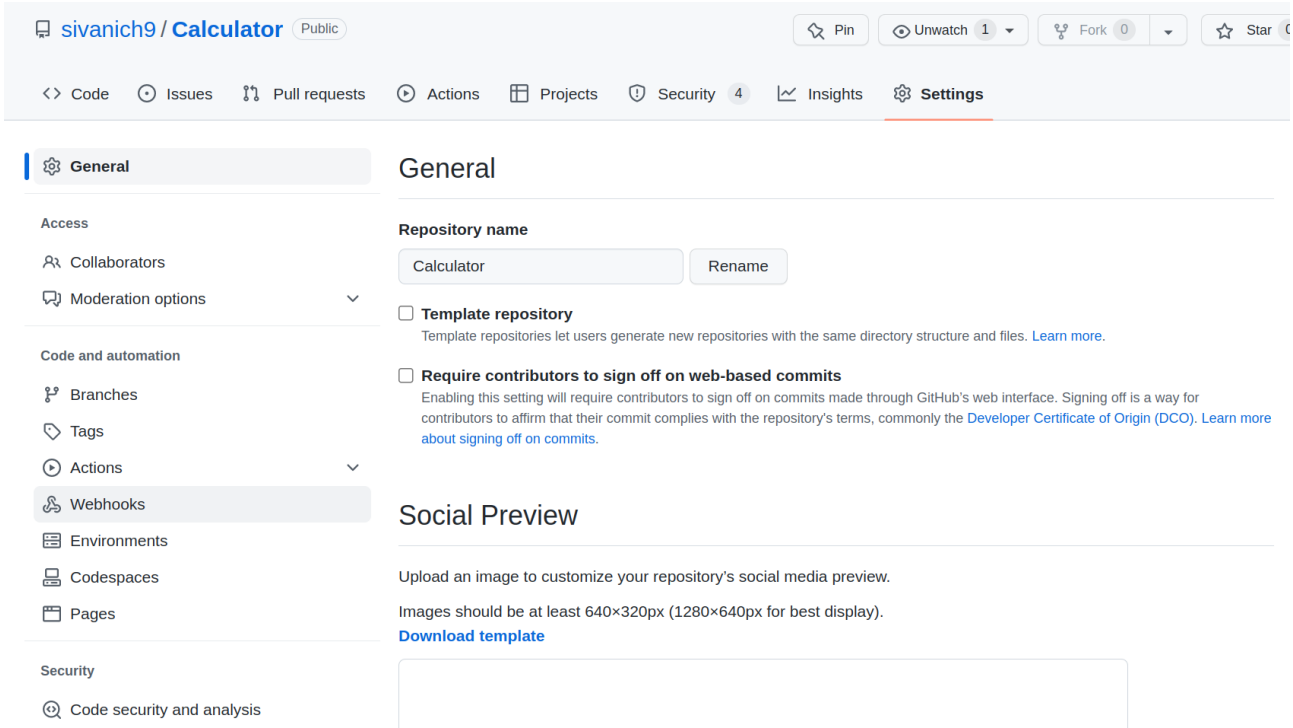
To start ngrok, use command : *ngrok <http of jenkins server> 8080*

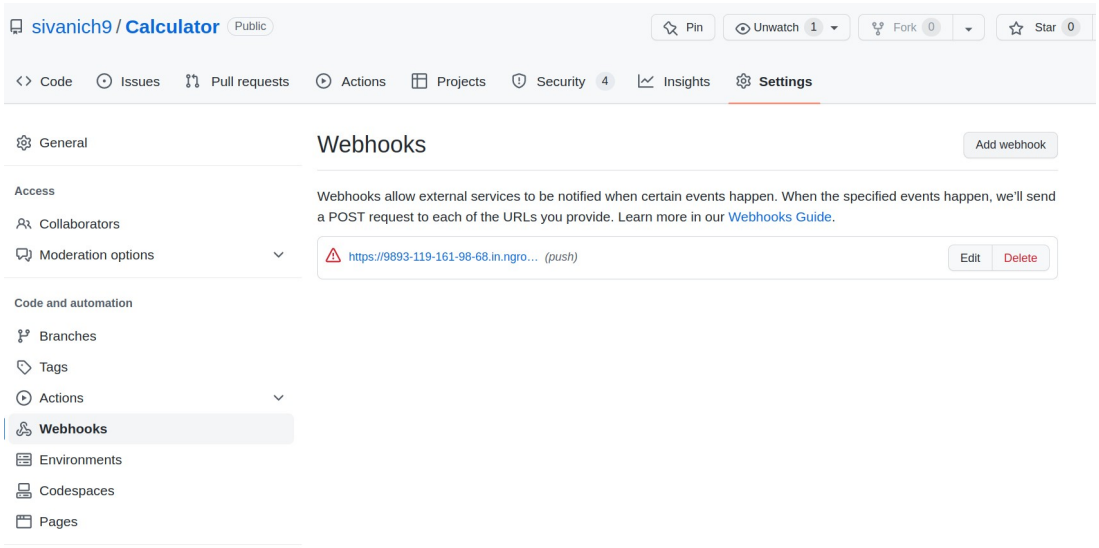




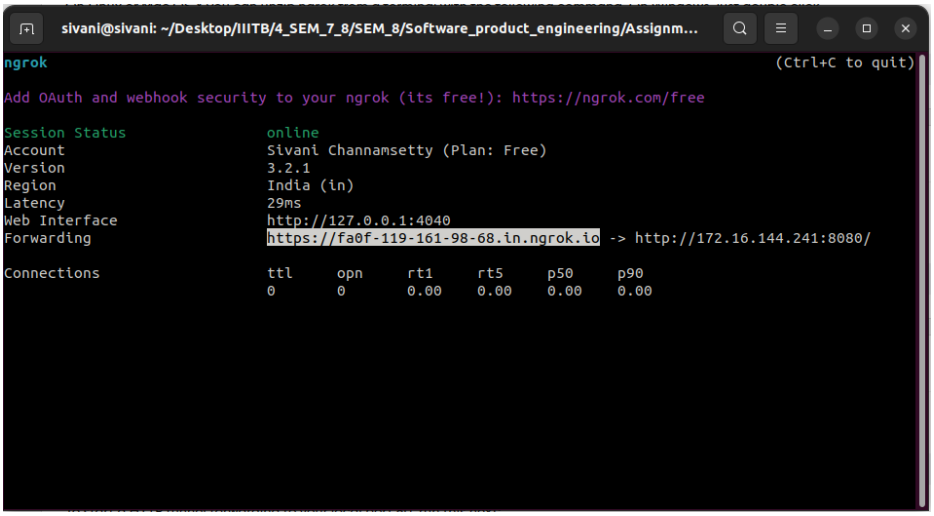
Setting up WebHooks :

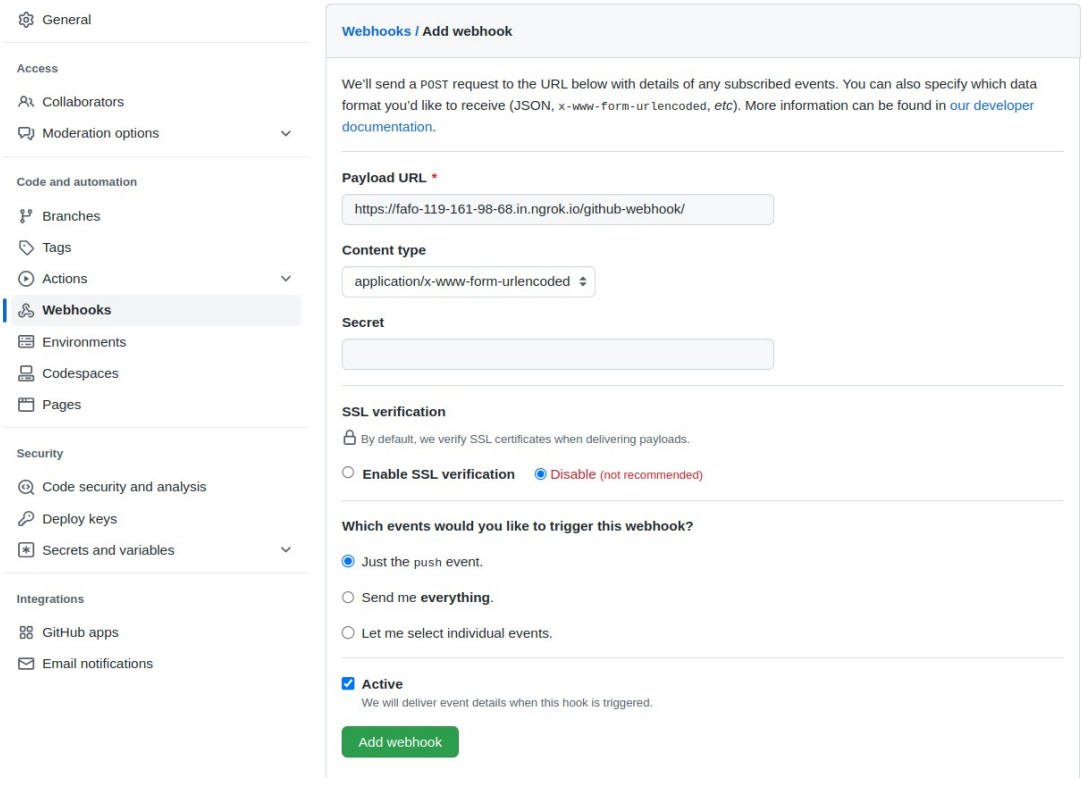
Navigate to the settings section of the Github repository, then proceed to the Webhooks tab. Add the ngrok address in payload URL and the personal access token in secret





Run ngrok in terminal and copy the ngrok public ip address

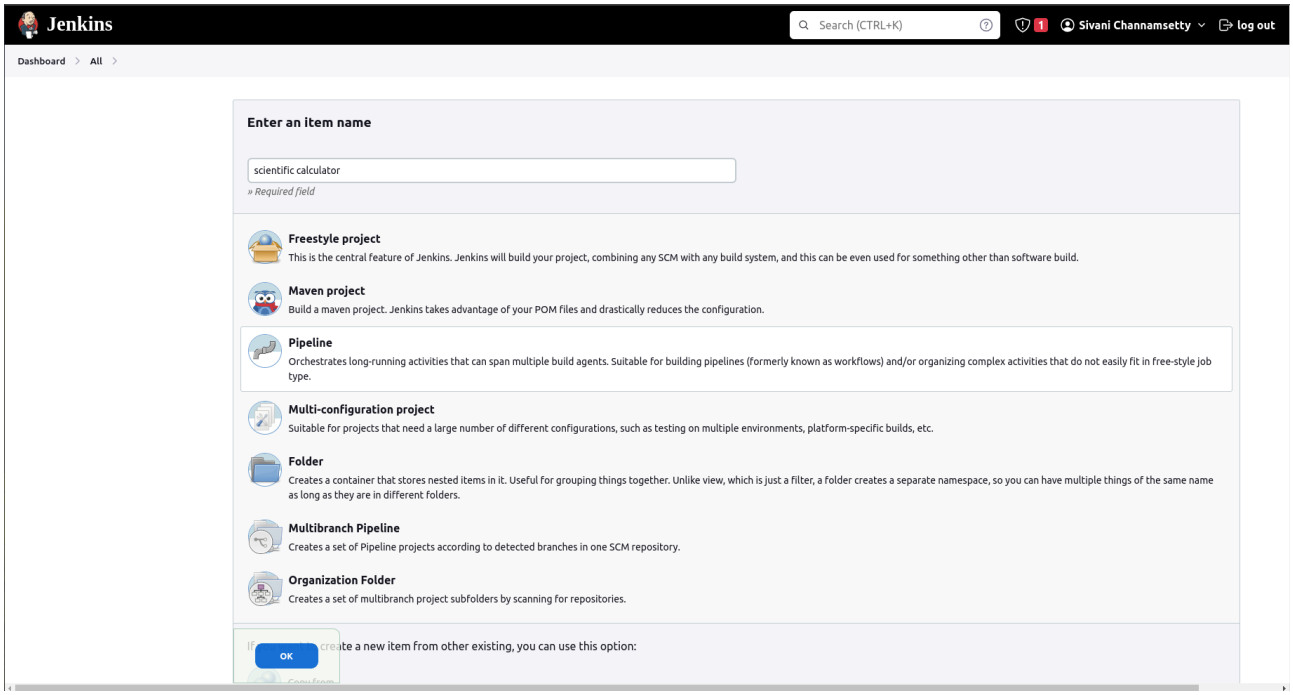




Jenkins :

An open-source automation server called Jenkins helps in automating a few steps in the software development process. It's a popular tool that development teams use to automate their build, test, and deployment processes as part of continuous integration and continuous delivery (CI/CD). Developers can use Jenkins to automatically build, test, and deploy their code to several environments, including development, staging, and production. To build a full CI/CD pipeline, Jenkins may also integrate with a variety of different tools and technologies. Jenkins can integrate with a variety of technologies, including version control systems, testing frameworks, and deployment tools, and employs plugins to expand its capability. It is compatible with a variety of operating systems, including Windows, Linux, and macOS. We need docker, github, maven, ansible and build pipeline plugins. So we download all plugins in jenkins.

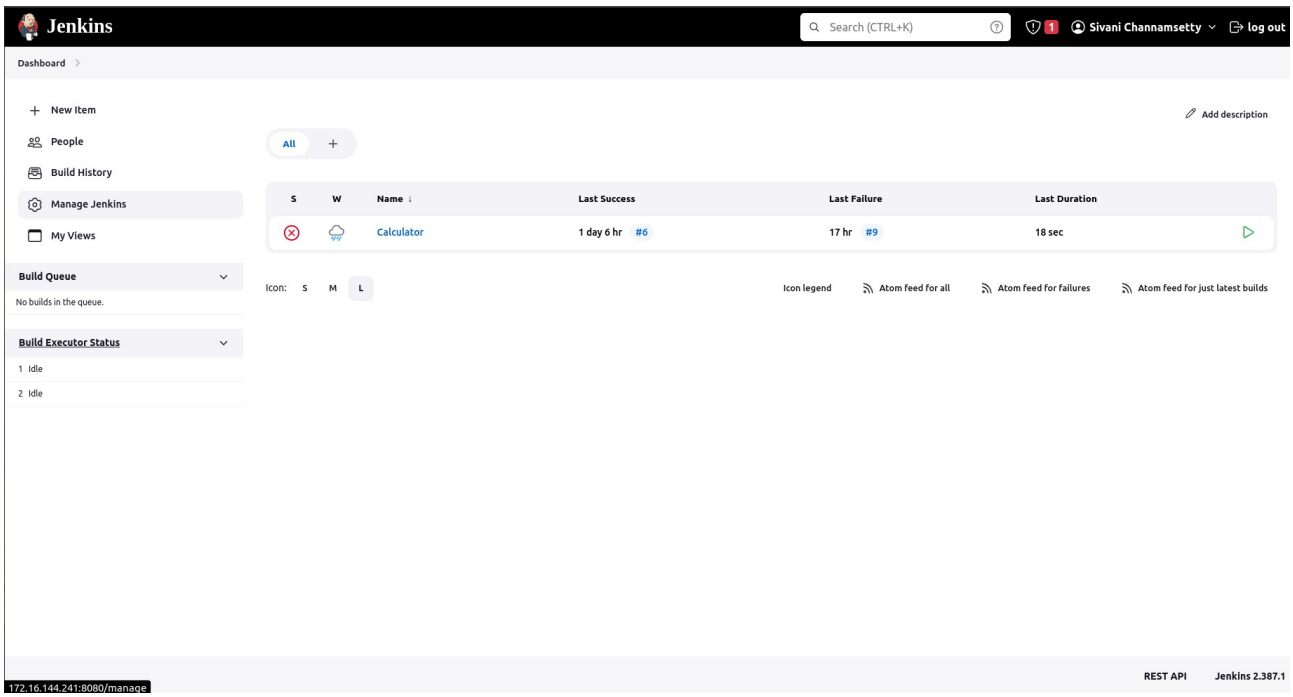
First create jenkins project and select pipeline

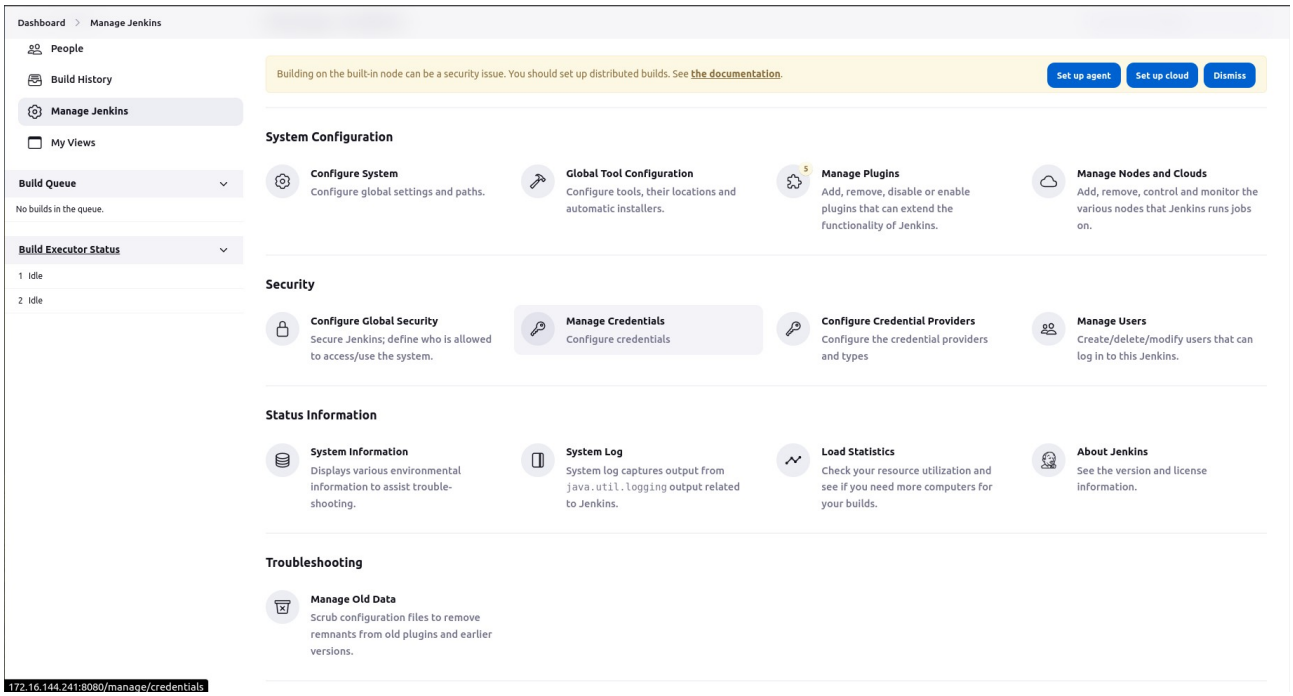


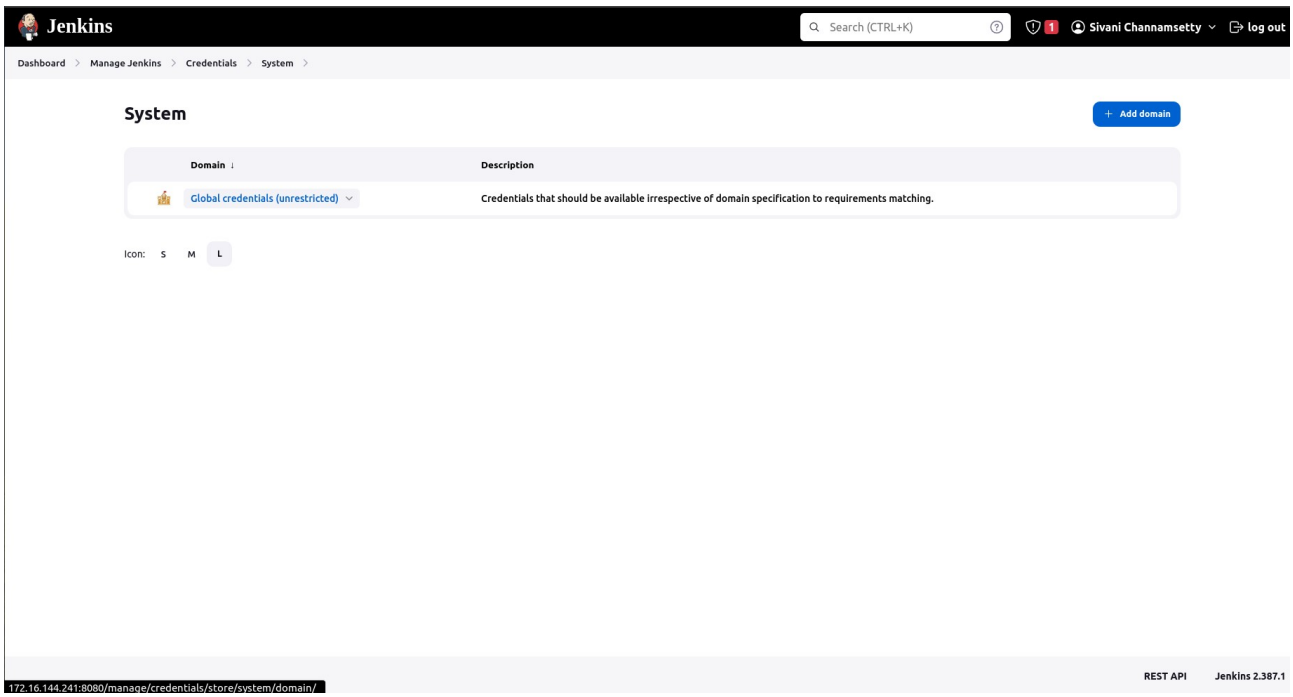
Add dockerhub credentials in jenkins

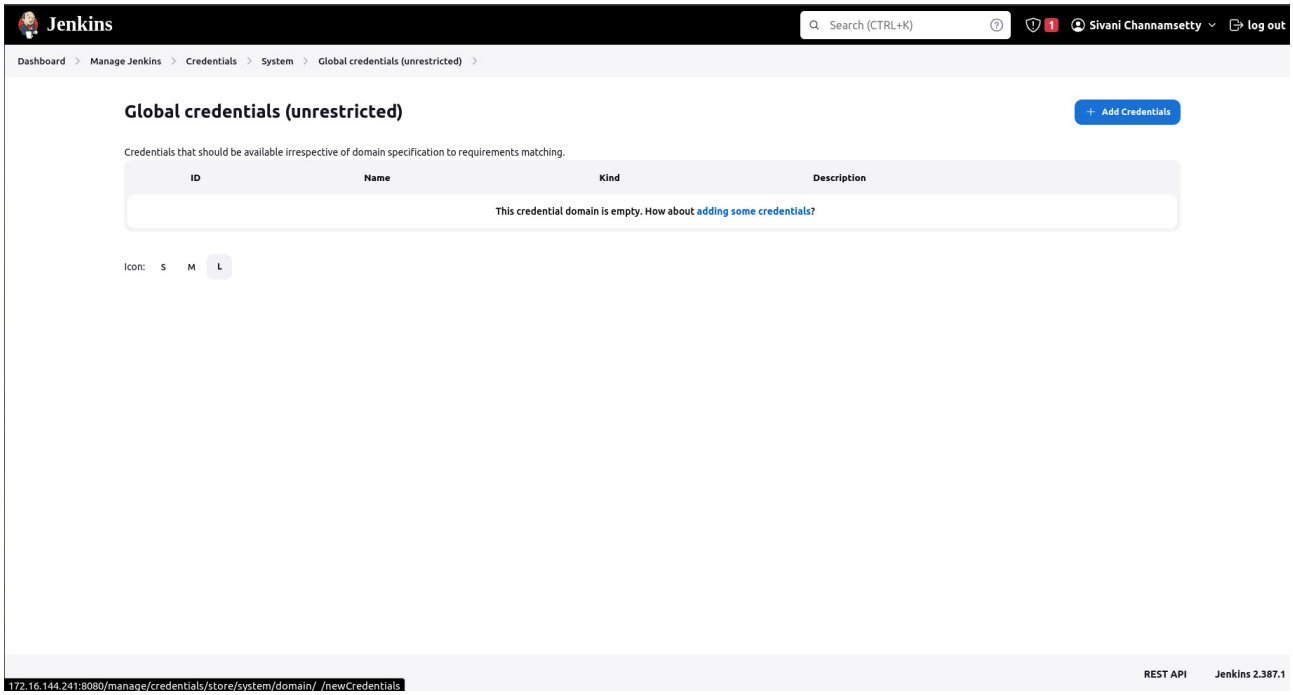
Go to Manage Jenkins > Manage Credentials > System > Global credentials (unrestricted) > Add Credentials

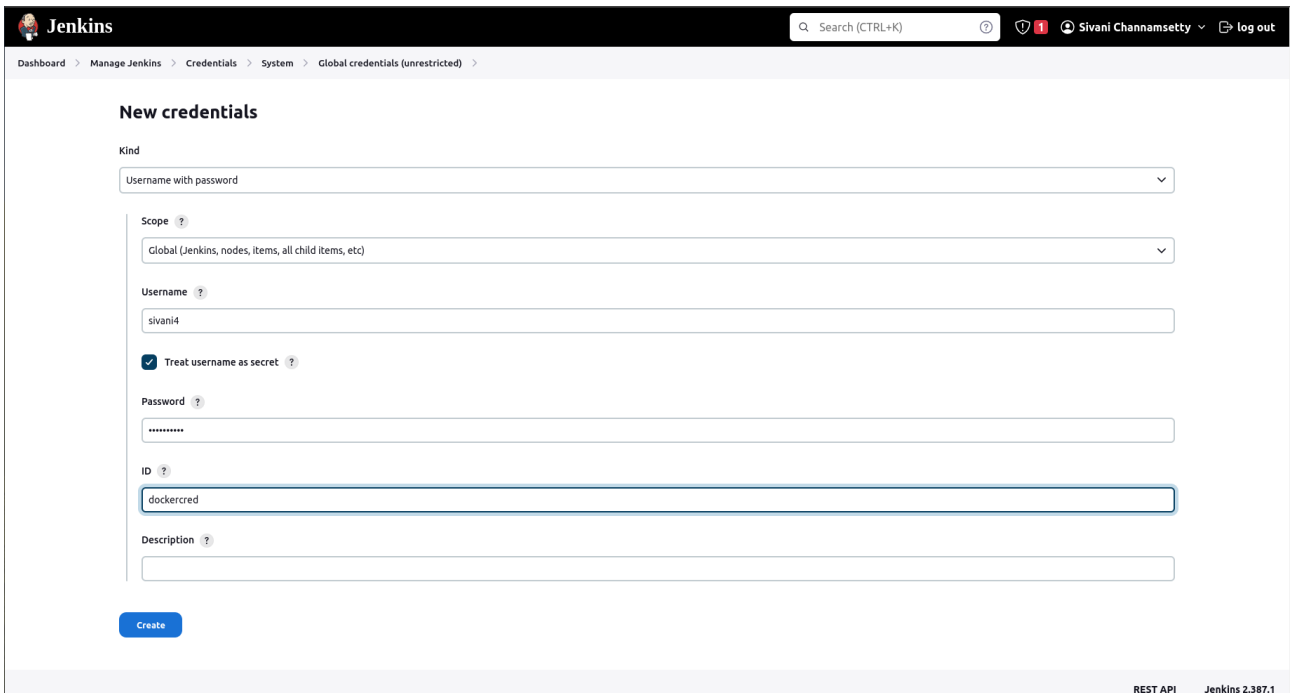
Choose Kind as Username with password and Scope as Global; provide your docker username and password with a unique ID of your own. This ID is used for referring the username with a password later in your pipelines.


Jenkins
Search (CTRL+K)
Sivani Channamsetty
log out
Dashboard
New Item
People
Build History
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Add description
All
S
W
Name
Last Success
Last Failure
Last Duration
Calculator
1 day 6 hr #6
17 hr #9
18 sec
Icon: S M L
Icon legend
Atom feed for all
Atom feed for failures
Atom feed for just latest builds
REST API
Jenkins 2.387.1
172.16.144.241:8080/manage


Dashboard
Manage Jenkins
People
Build History
Manage Jenkins
My Views
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Building on the built-in node can be a security issue. You should set up distributed builds. See the documentation.
Set up agent
Set up cloud
Dismiss
System Configuration
Configure System
Configure global settings and paths.
Global Tool Configuration
Configure tools, their locations and automatic installers.
Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.
Security
Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.
Manage Credentials
Configure credentials
Configure Credential Providers
Configure the credential providers and types
Manage Users
Create/delete/modify users that can log in to this Jenkins.
Status Information
System Information
Displays various environmental information to assist trouble-shooting.
System Log
System log captures output from java.util.logging output related to Jenkins.
Load Statistics
Check your resource utilization and see if you need more computers for your builds.
About Jenkins
See the version and license information.
Troubleshooting
Manage Old Data
Scrub configuration files to remove remnants from old plugins and earlier versions.
172.16.144.241:8080/manage/credentials

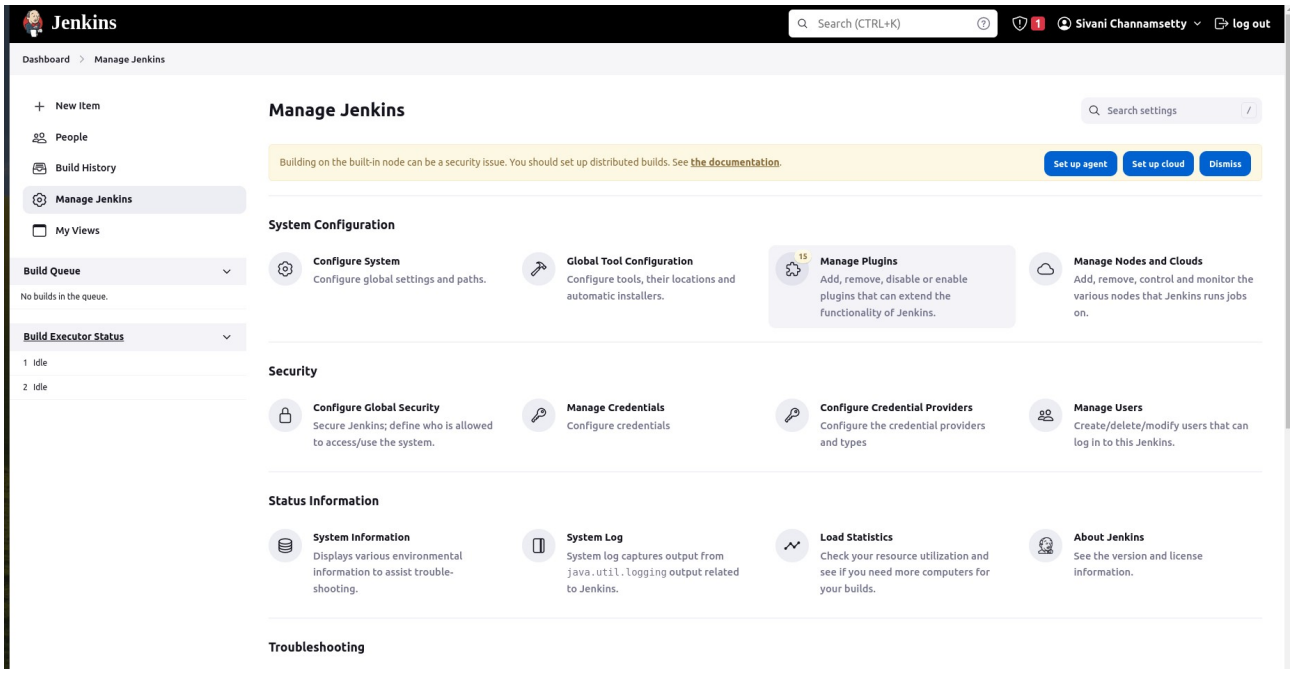

Jenkins
Search (CTRL+K)
1
Sivani Channamsetty
log out
Dashboard
Manage Jenkins
Credentials
System
System
Add domain
Domain
Description
Global credentials (unrestricted)
Credentials that should be available irrespective of domain specification to requirements matching.
Icon: S M L
172.16.144.241:8080/manage/credentials/store/system/domain/
REST API
Jenkins 2.387.1


Jenkins
Search (CTRL+K)
1
Sivani Channamsetty
log out
Dashboard
Manage Jenkins
Credentials
System
Global credentials (unrestricted)
Global credentials (unrestricted)
Add Credentials
Credentials that should be available irrespective of domain specification to requirements matching.
ID
Name
Kind
Description
This credential domain is empty. How about adding some credentials?
Icon: S M L
172.16.144.241:8080/manage/credentials/store/system/domain/_/newCredentials
REST API
Jenkins 2.387.1

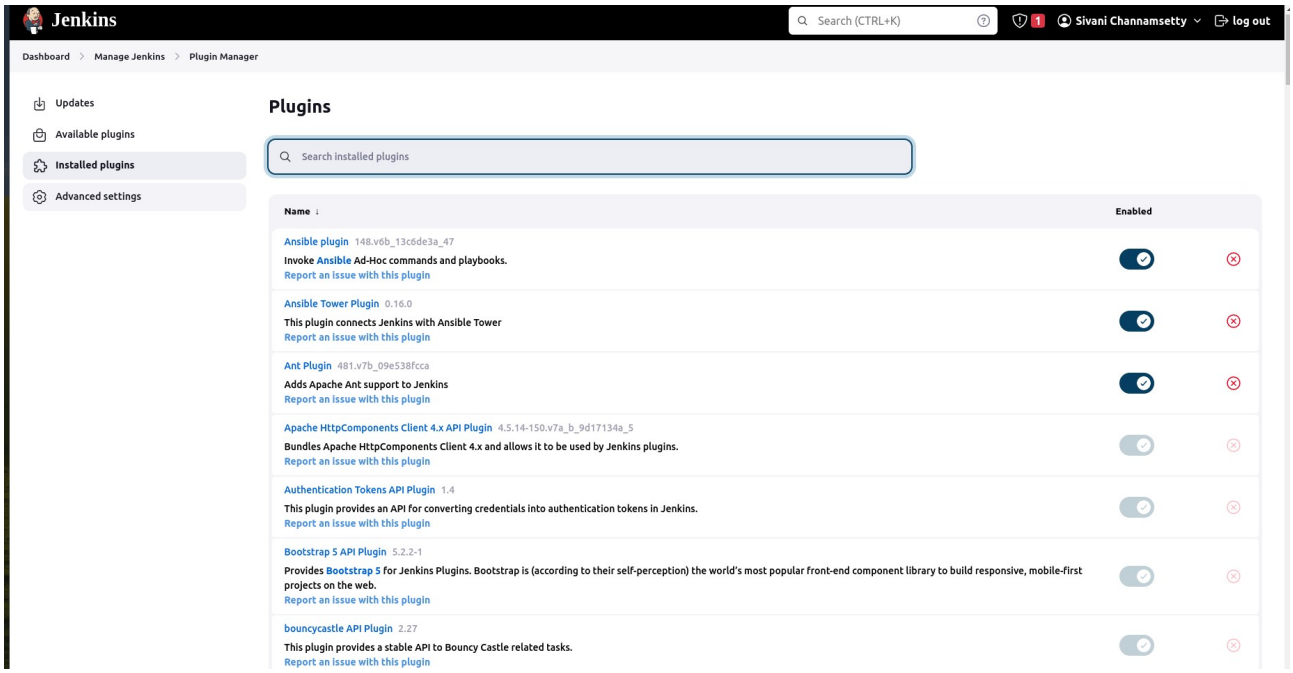


Install the following plugins : Maven, Git, Ansible and Docker.

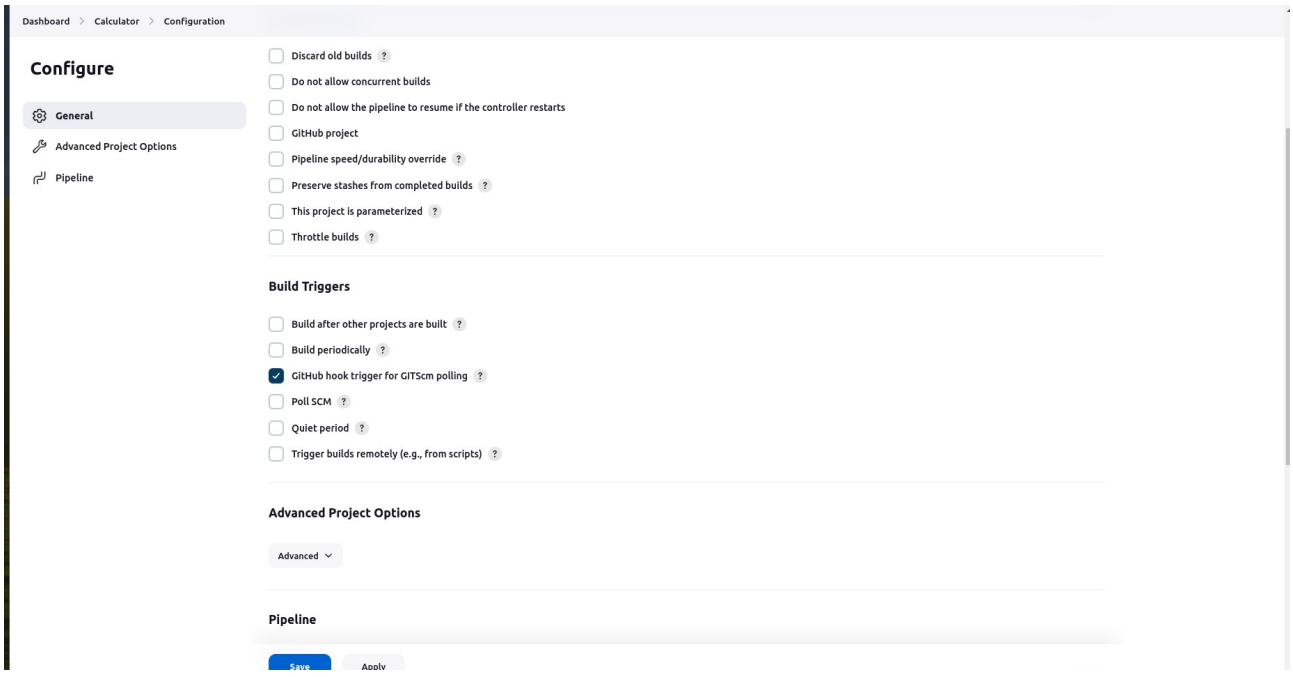
Go to *Dashboard* and the *Manage Jenkins* and then *Manage Plugins*



Install the plugins required



Select *Github hook trigger for GITScm polling*



Add pipeline script for stages *Git pull, Maven test, Maven build, Docker image creation , Deploying/Pushing docker image* and *Ansible deploy.*

Jenkins pipeline :

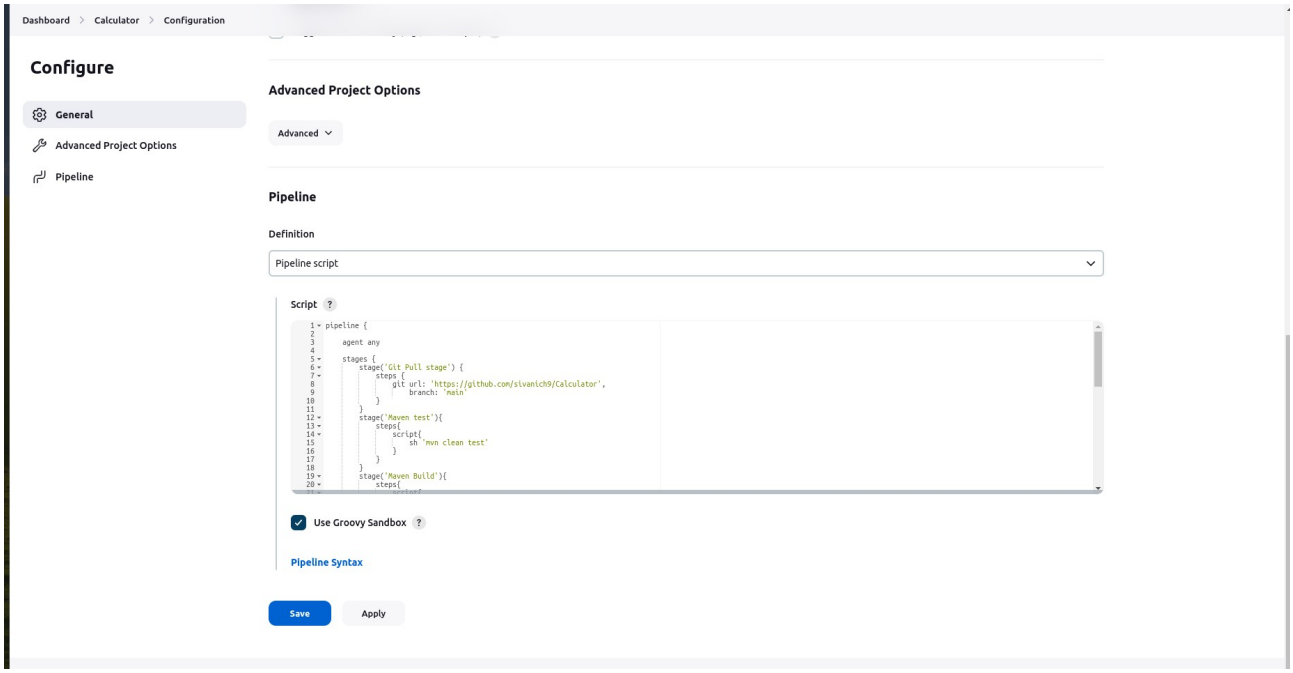
```
pipeline {

    agent any

    stages {
        stage('Git Pull stage') {
            steps {
                git url: 'https://github.com/sivanich9/Calculator',
                    branch: 'main'
            }
        }
        stage('Maven test'){
            steps{
                script{
                    sh 'mvn clean test'
                }
            }
        }
        stage('Maven Build'){
            steps{
                script{
                    sh 'mvn clean install'
                }
            }
        }
        stage('Docker Build Image'){
            steps{
                script{
                     sh 'docker build -t sivani4/calculator:latest .'
                }
            }
        }
        stage('Push Docker Image'){
            steps{
                script{
                    withDockerRegistry([ credentialsId: "dockercred", url: "" ])
                    {
                        sh 'docker push sivani4/calculator:latest'
```

```
        stage('Push Docker Image'){
            steps{
                script{
                    withDockerRegistry([ credentialsId: "dockercred", url: "" ])
                    {
                        sh 'docker push sivani4/calculator:latest'
                    }
                }
            }
        }
        stage("Ansible deploy") {
            steps {
                script {
                    // sh "ssh-keygen -t rsa"
                    // sh "ssh-copy-id  sivani@localhost"
                    // ansiblePlaybook disableHostKeyChecking: true, installation: 'ansible_1', inventory: 'inventory', playbook: 'p2.yml'
                    sh "/usr/bin/pip3 install docker"
                    sh "ansible-playbook p2.yml -i inventory"
                }
            }
        }
    }
}

```

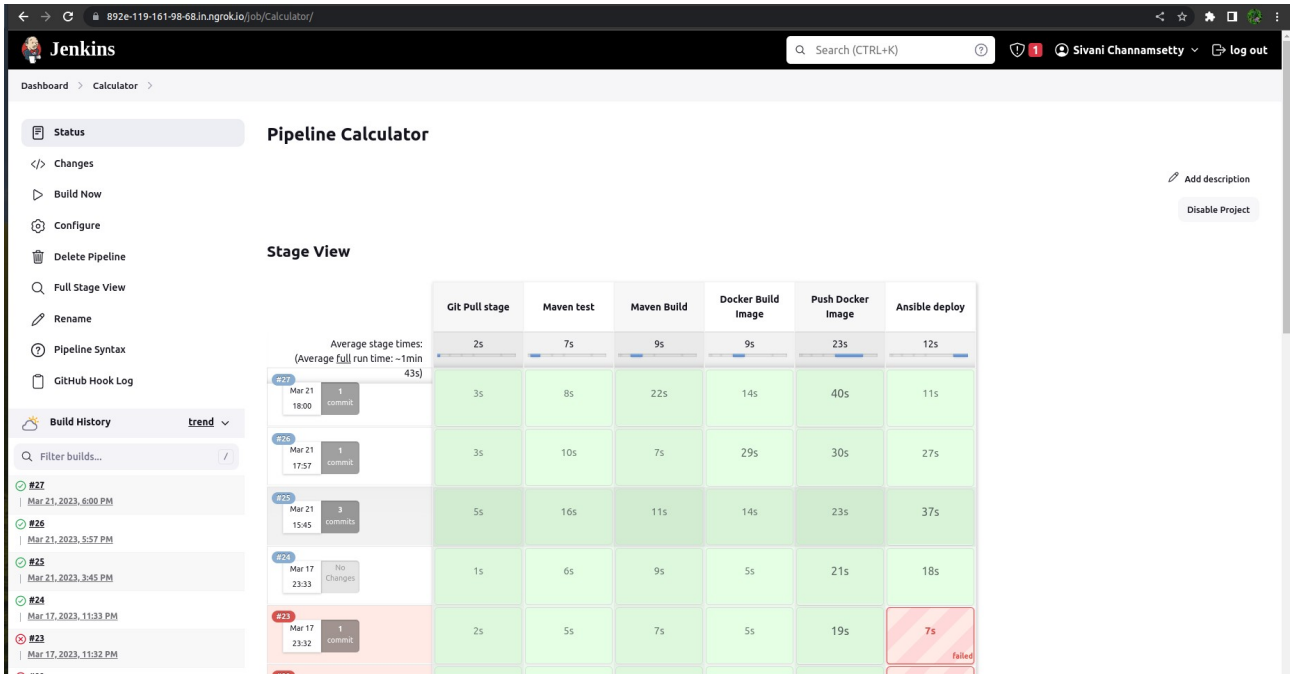
Code :

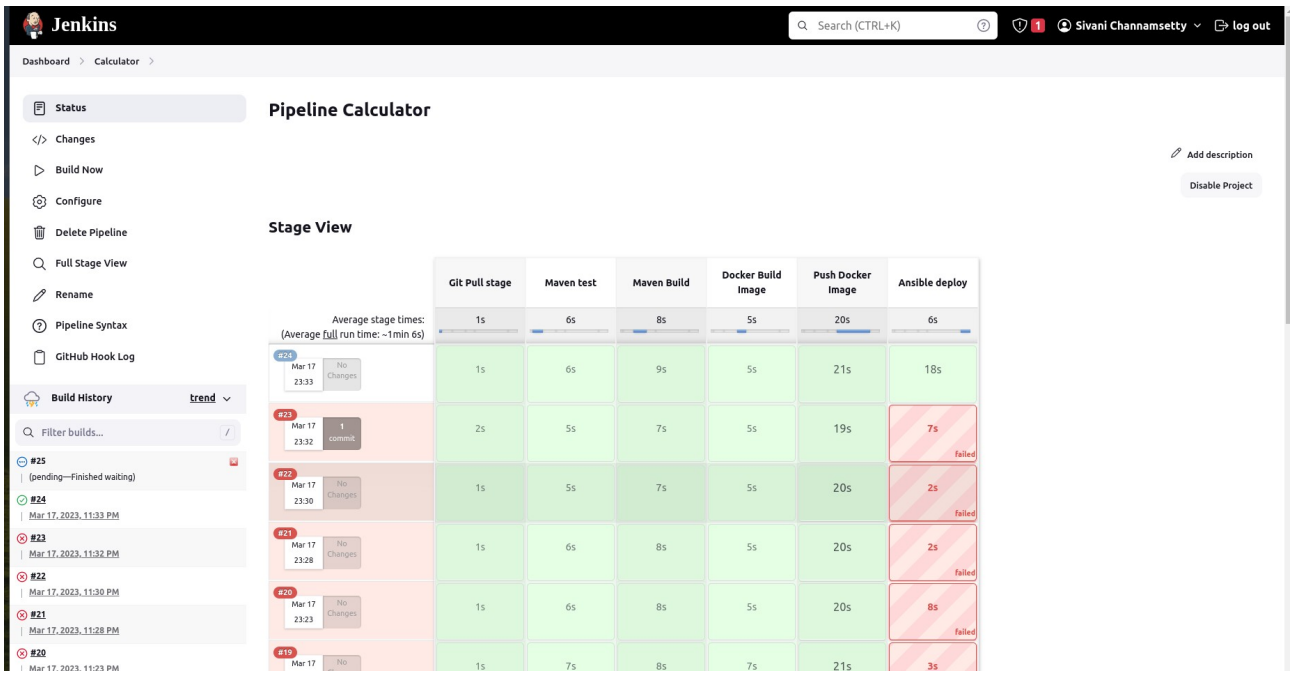
```
pipeline {

    agent any

    stages {
        stage('Git Pull stage') {
            steps {
                git url: 'https://github.com/sivanich9/Calculator',
                    branch: 'main'
            }
        }
        stage('Maven test'){
            steps{
                script{
                    sh 'mvn clean test'
                }
            }
        }
        stage('Maven Build'){
            steps{
                script{
                    sh 'mvn clean install'
                }
            }
        }
        stage('Docker Build Image'){
            steps{
                script{
                     sh 'docker build -t sivani4/calculator:latest .'
                }
            }
        }
        stage('Push Docker Image'){
            steps{
                script{
                    withDockerRegistry([ credentialsId: "dockercred", url: "" ])
                    {
                        sh 'docker push sivani4/calculator:latest'
                    }
                }
            }
        }
        stage("Ansible deploy") {
            steps {
                script {
                    // sh "ssh-keygen -t rsa"
                    // sh "ssh-copy-id  sivani@localhost"
                    // ansiblePlaybook disableHostKeyChecking: true, installation: 'ansible_1', inventory:
'inventory', playbook: 'p2.yml'
                    sh "/usr/bin/pip3 install docker"
```

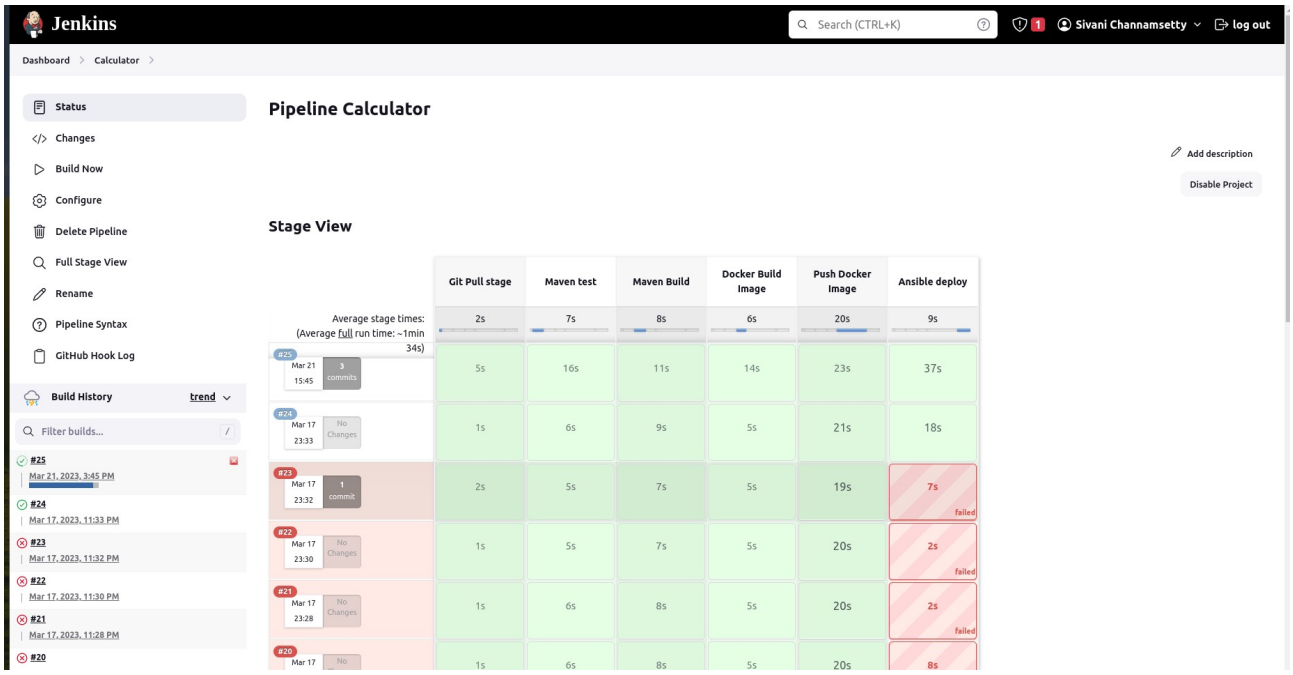
```
                sh "ansible-playbook p2.yml -i inventory"
            }
        }
    }
  }
}
```



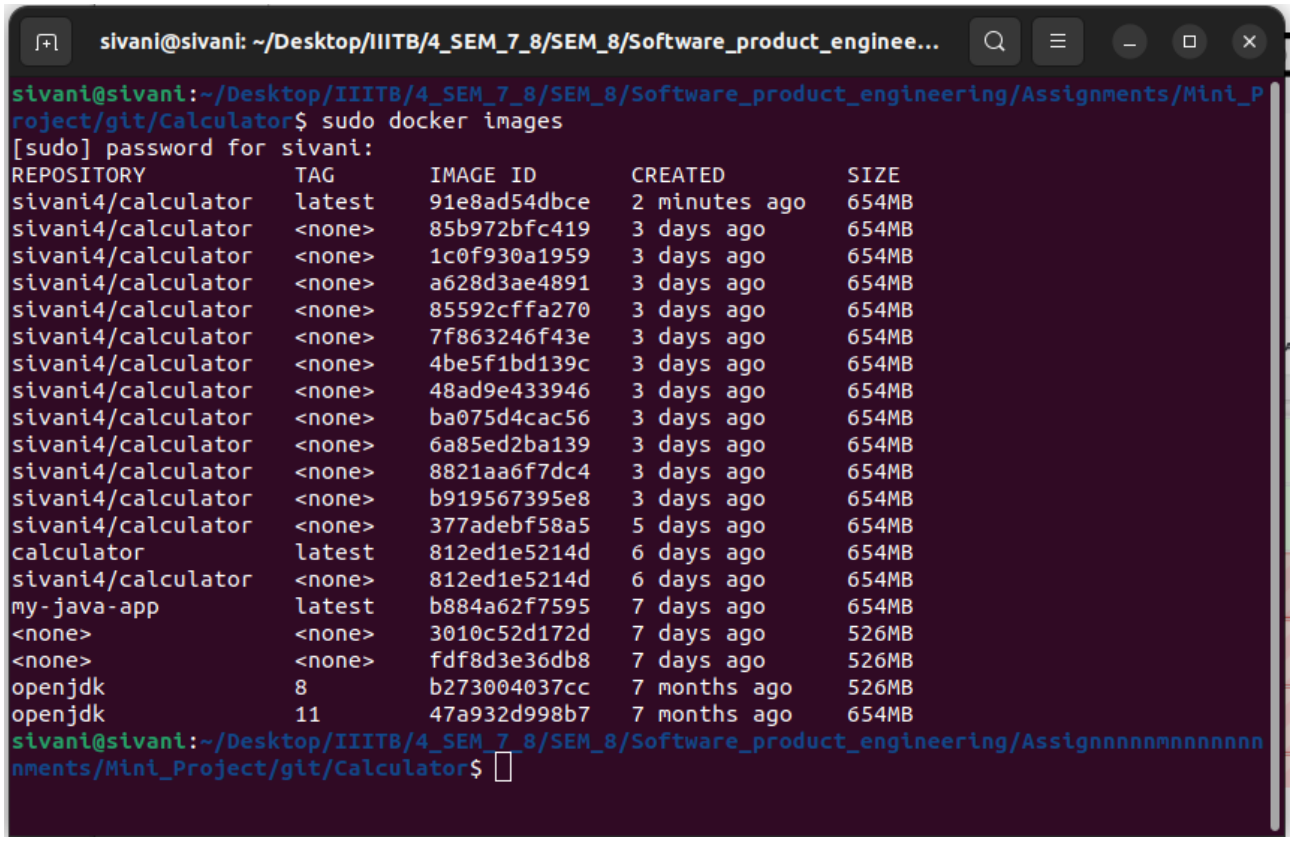
I added code to Github. As we set Github WebHooks, jenkins automatically build the code. Note that we have to run ngrok in our terminal. As you come out of ngrok, it doesn't build. You have to again run ngrok in our terminal then the address and accordingly we have to update the webhook in Github.

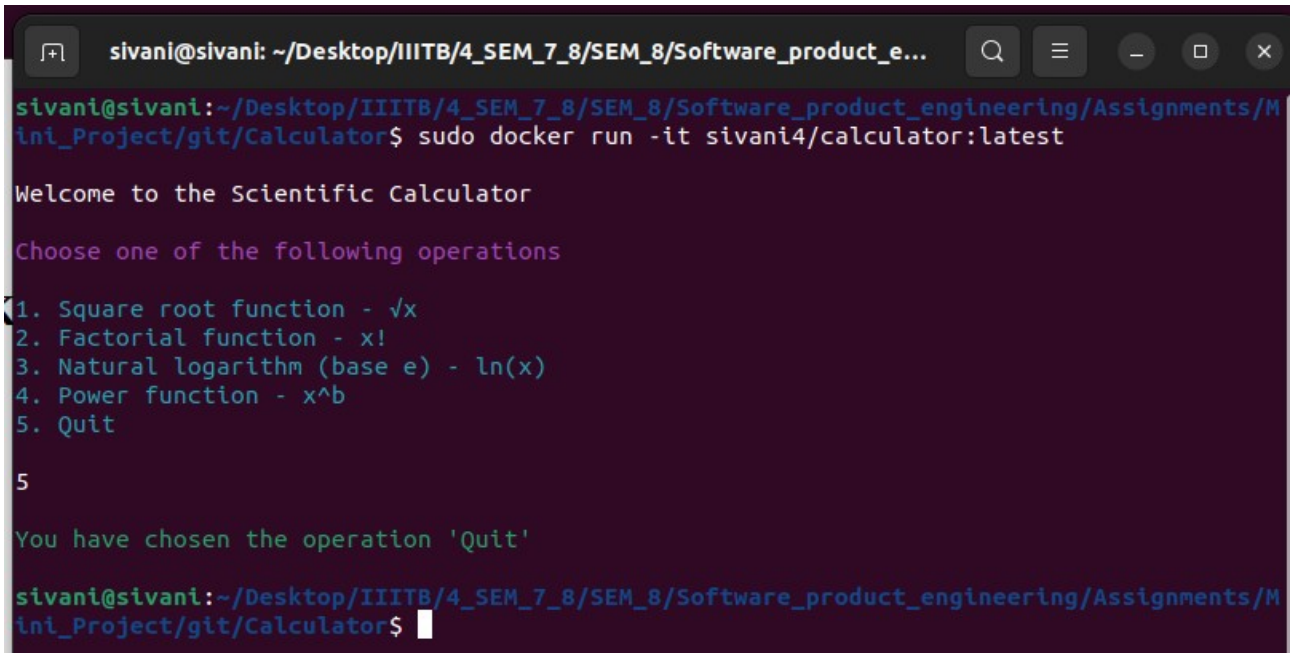






Now go to terminal and find docker image and run it. We will be able to run our project.

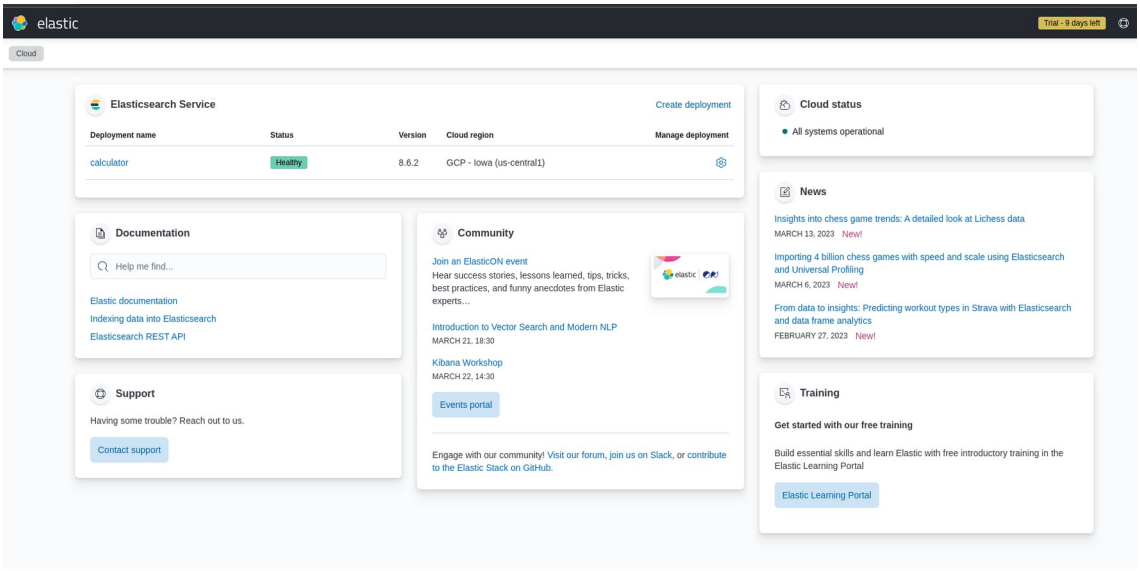




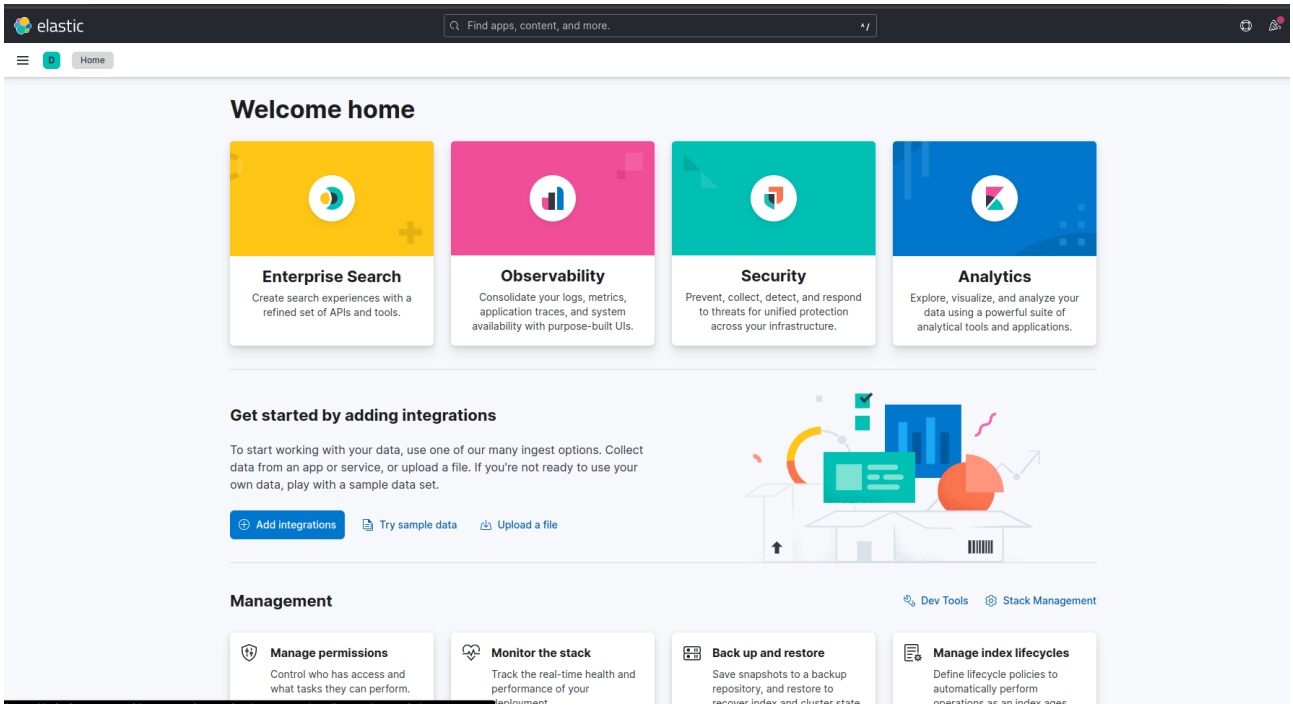
ELK-Stack :

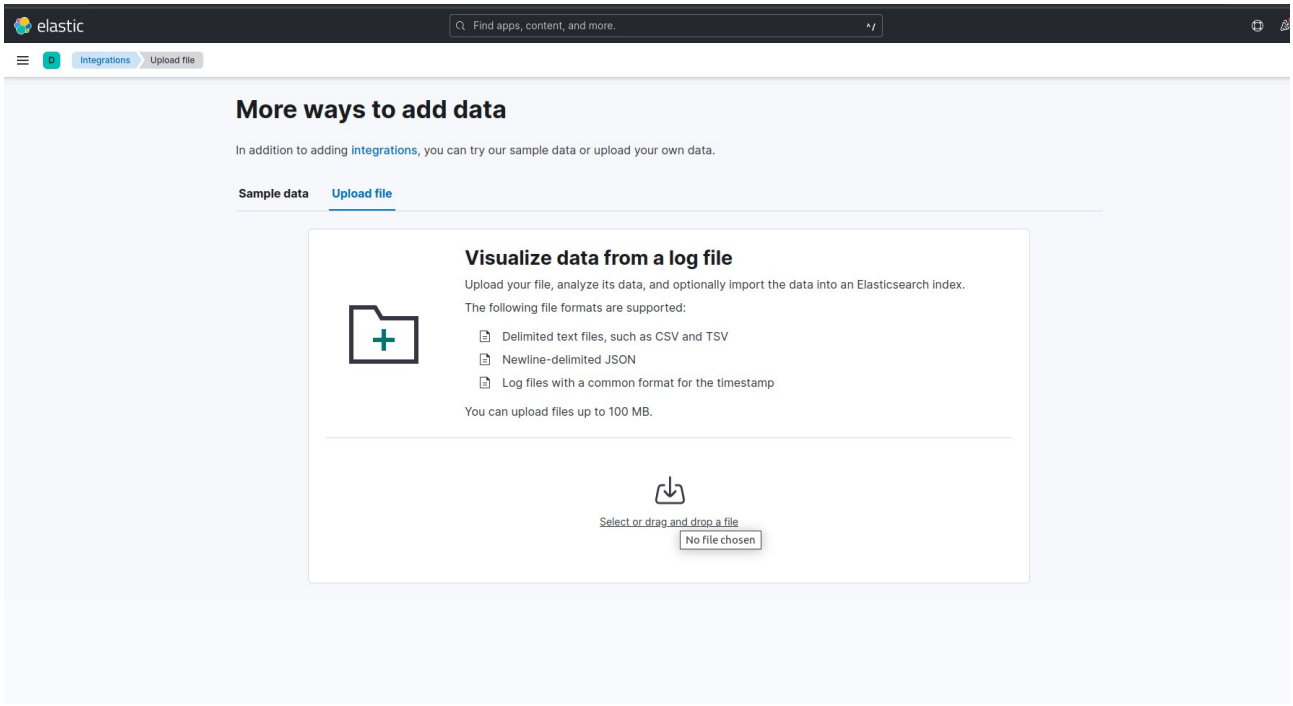
Once you finish deploying your software, the next crucial step is to continuosly monitor its performance. This involves verifying whether the software is functioning according to its intended design. The ELK stack offers a monitoring tool for any deployed software, which involves analyzing its logs. This analysis can then be accessed and visualized on the Kibana dashboard.

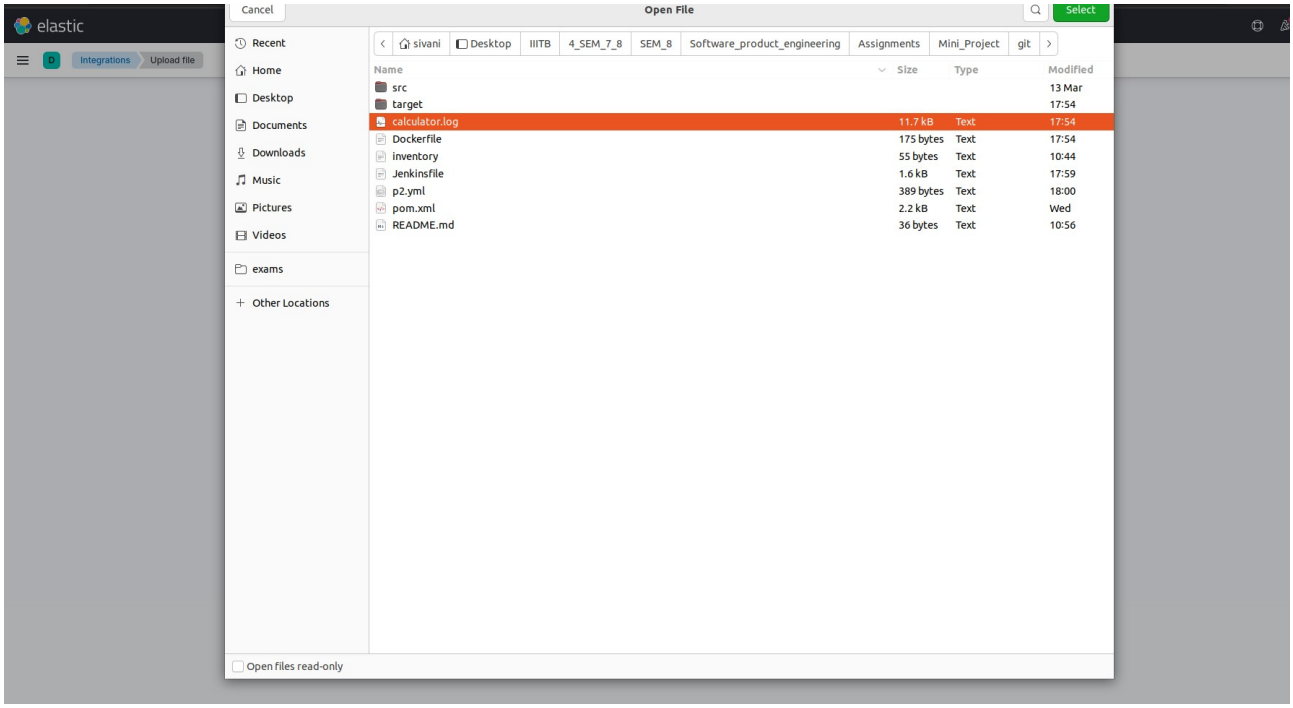
Go to https://cloud.elastic.co/login and create a deployment



And go to that deployment and then upload a file



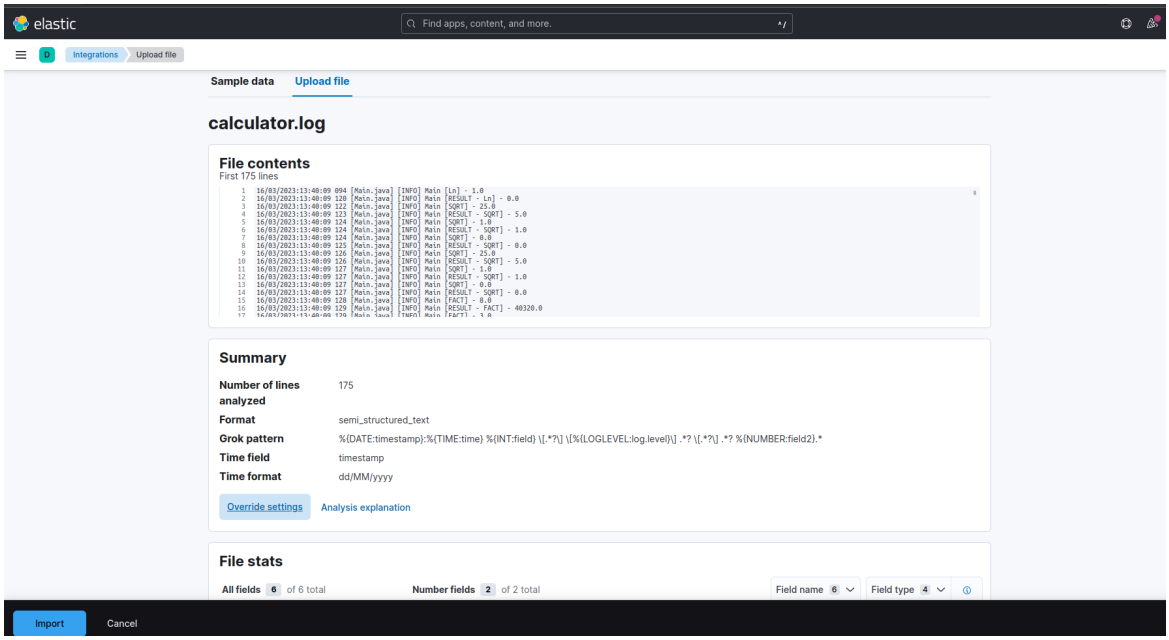


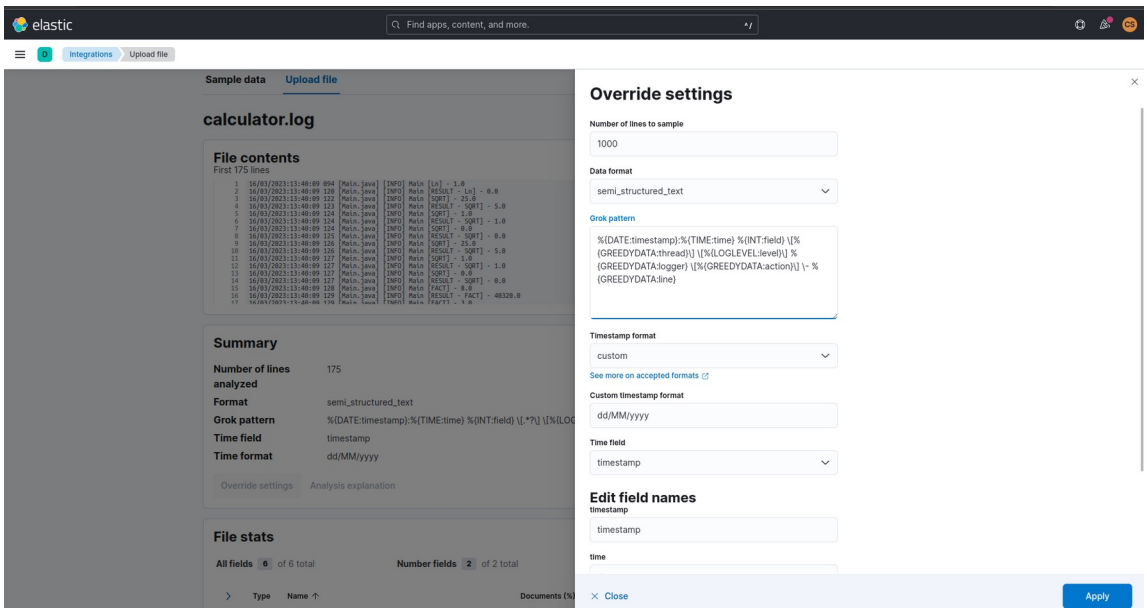


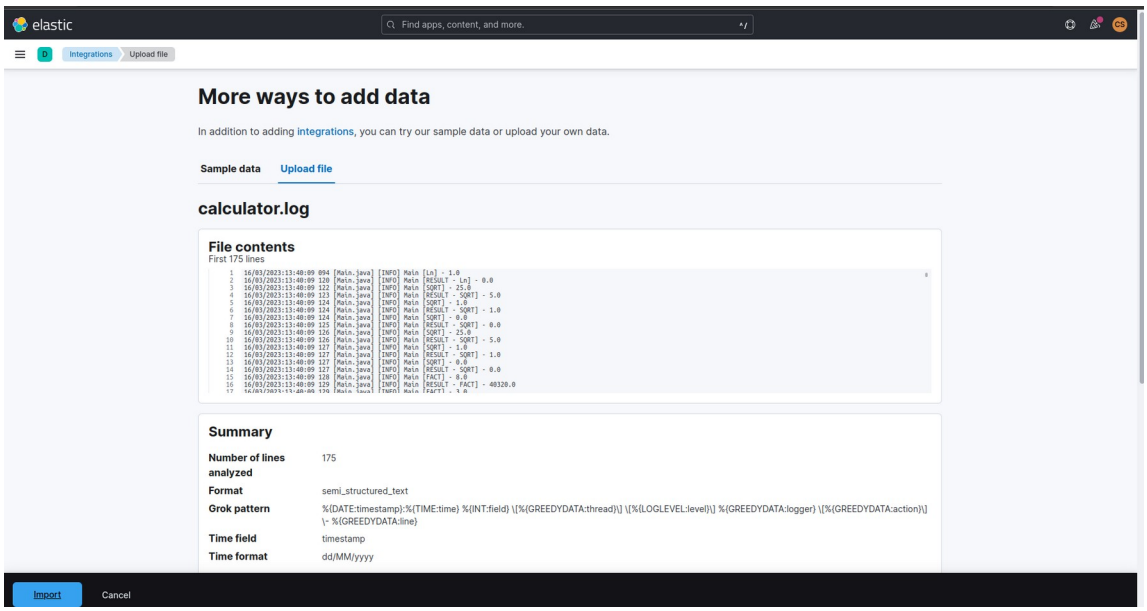
Calcultor.log file contains logs

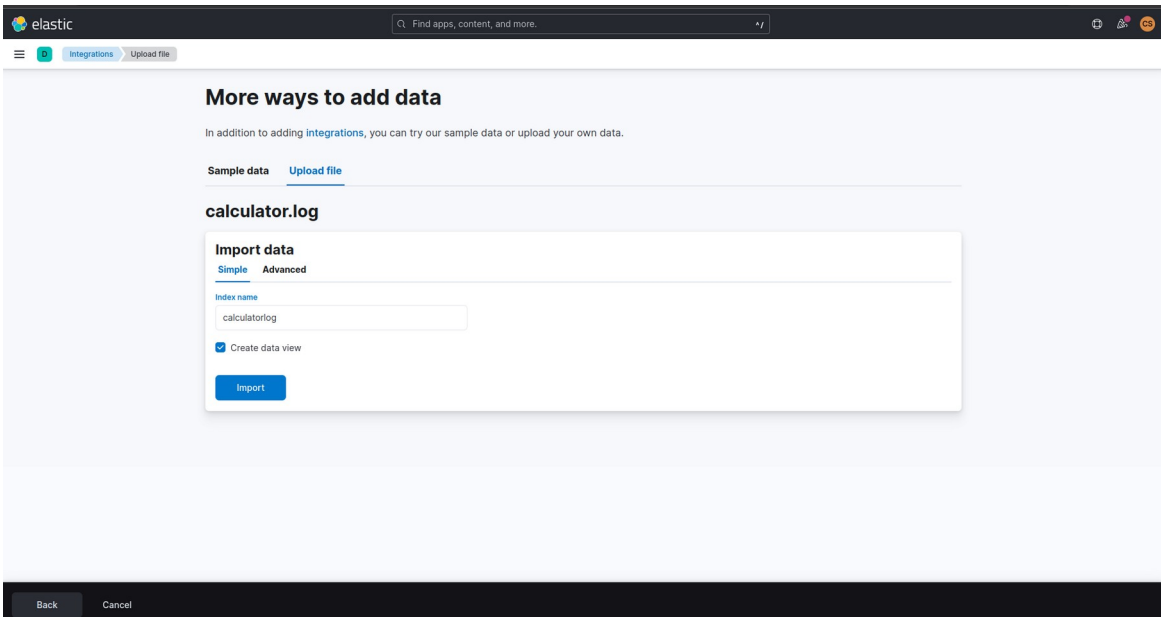
```
sivani@sivani: ~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator
sivani@sivani:~/Desktop/IIITB/4_SEM_7_8/SEM_8/Software_product_engineering/Assignments/Mini_Project/git/Calculator$ cat calculator.log
16/03/2023:13:40:09 094 [Main.java] [INFO] Main [Ln] - 1.0
16/03/2023:13:40:09 120 [Main.java] [INFO] Main [RESULT - Ln] - 0.0
16/03/2023:13:40:09 122 [Main.java] [INFO] Main [SQRT] - 25.0
16/03/2023:13:40:09 123 [Main.java] [INFO] Main [RESULT - SQRT] - 5.0
16/03/2023:13:40:09 124 [Main.java] [INFO] Main [SQRT] - 1.0
16/03/2023:13:40:09 124 [Main.java] [INFO] Main [RESULT - SQRT] - 1.0
16/03/2023:13:40:09 124 [Main.java] [INFO] Main [SQRT] - 0.0
16/03/2023:13:40:09 125 [Main.java] [INFO] Main [RESULT - SQRT] - 0.0
16/03/2023:13:40:09 126 [Main.java] [INFO] Main [SQRT] - 25.0
16/03/2023:13:40:09 126 [Main.java] [INFO] Main [RESULT - SQRT] - 5.0
16/03/2023:13:40:09 127 [Main.java] [INFO] Main [SQRT] - 1.0
16/03/2023:13:40:09 127 [Main.java] [INFO] Main [RESULT - SQRT] - 1.0
16/03/2023:13:40:09 127 [Main.java] [INFO] Main [SQRT] - 0.0
16/03/2023:13:40:09 127 [Main.java] [INFO] Main [RESULT - SQRT] - 0.0
16/03/2023:13:40:09 128 [Main.java] [INFO] Main [FACT] - 8.0
16/03/2023:13:40:09 129 [Main.java] [INFO] Main [RESULT - FACT] - 40320.0
16/03/2023:13:40:09 129 [Main.java] [INFO] Main [FACT] - 3.0
16/03/2023:13:40:09 129 [Main.java] [INFO] Main [RESULT - FACT] - 6.0
16/03/2023:13:40:09 129 [Main.java] [INFO] Main [FACT] - 0.0
16/03/2023:13:40:09 130 [Main.java] [INFO] Main [RESULT - FACT] - 1.0
16/03/2023:13:40:09 137 [Main.java] [INFO] Main [POW] - 2.0 5.0
16/03/2023:13:40:09 138 [Main.java] [INFO] Main [RESULT - POW] - 32.0
16/03/2023:13:40:09 138 [Main.java] [INFO] Main [POW] - 9.0 0.0
16/03/2023:13:40:09 139 [Main.java] [INFO] Main [RESULT - POW] - 1.0
16/03/2023:13:40:09 139 [Main.java] [INFO] Main [POW] - 3.0 2.0
16/03/2023:13:40:09 139 [Main.java] [INFO] Main [RESULT - POW] - 9.0
16/03/2023:13:40:09 140 [Main.java] [INFO] Main [Ln] - 15.0
16/03/2023:13:40:09 140 [Main.java] [INFO] Main [RESULT - Ln] - 2.70805020110221
16/03/2023:13:40:09 141 [Main.java] [INFO] Main [Ln] - 29.0
16/03/2023:13:40:09 141 [Main.java] [INFO] Main [RESULT - Ln] - 3.367295829986474
16/03/2023:13:40:09 141 [Main.java] [INFO] Main [Ln] - 1.0
16/03/2023:13:40:09 142 [Main.java] [INFO] Main [RESULT - Ln] - 0.0
```

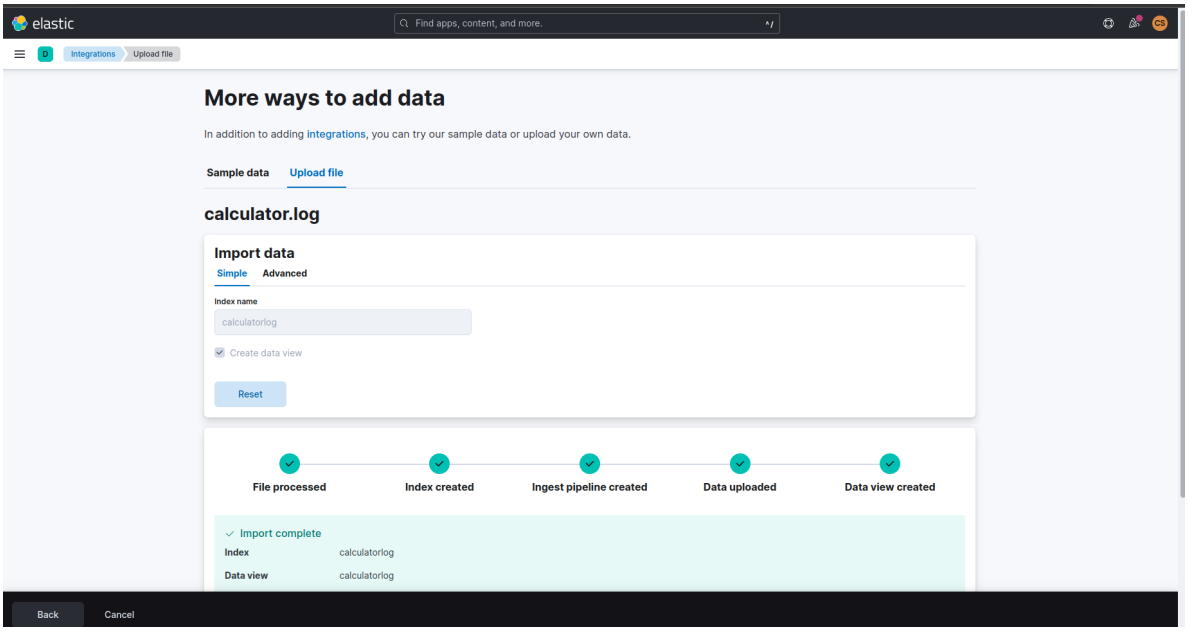
After that override settings to change grok pattern according to our log file



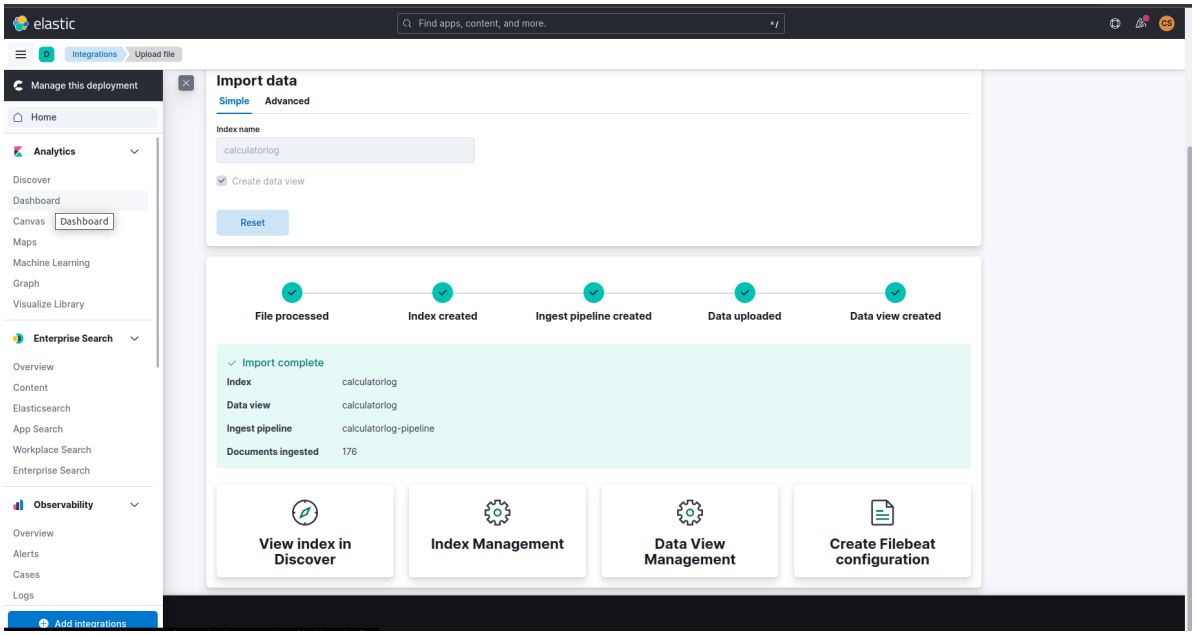


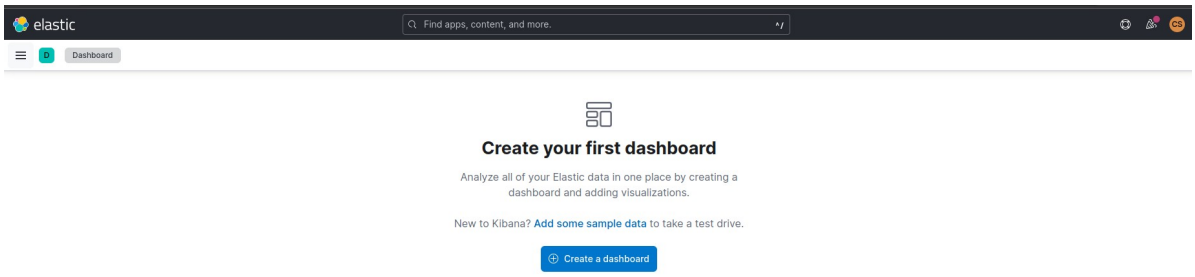

elastic
Find apps, content, and more.
Integrations
Upload file
More ways to add data
In addition to adding integrations, you can try our sample data or upload your own data.
Sample data
Upload file
calculator.log
File contents
First 175 lines
Summary
Number of lines analyzed
175
Format
semi_structured_text
Grok pattern
%{DATE:timestamp}:%{TIME:time} %{INT:field} \[%{GREEDYDATA:thread}\] \[%{LOGLEVEL:level}\] %{GREEDYDATA:logger} \[%{GREEDYDATA:action}\] \- %{GREEDYDATA:line}
Time field
timestamp
Time format
dd/MM/yyyy
Import
Cancel

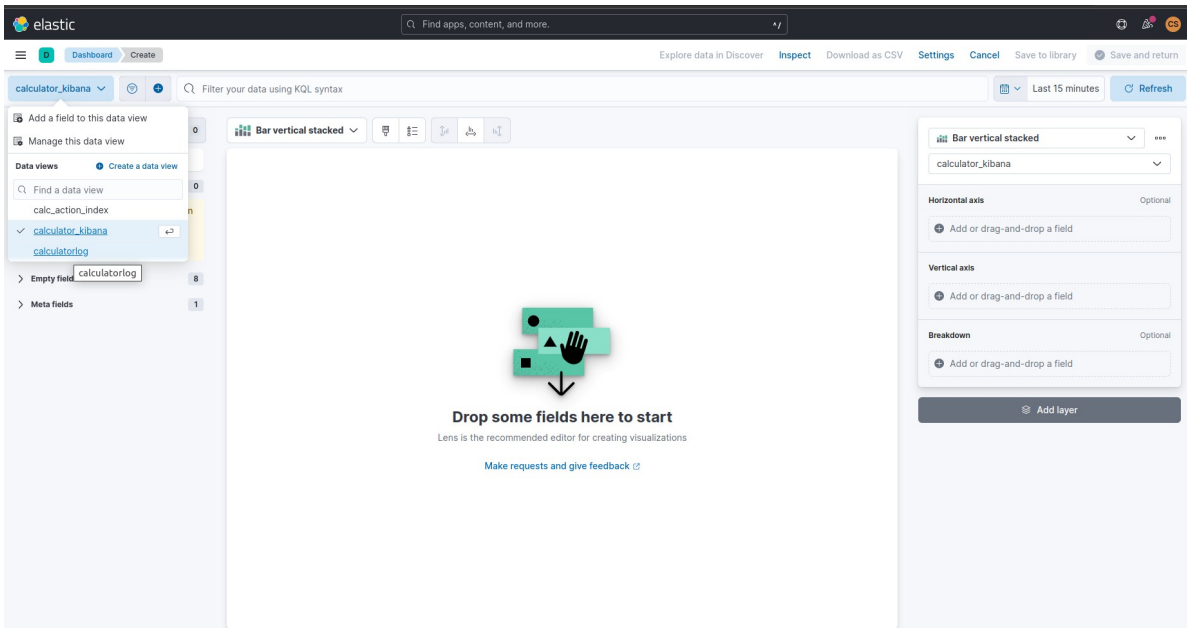

elastic
Find apps, content, and more.
Integrations
Upload file
More ways to add data
In addition to adding integrations, you can try our sample data or upload your own data.
Sample data
Upload file
calculator.log
Import data
Simple
Advanced
Index name
calculatorlog
Create data view
Import
Back
Cancel


elastic
Find apps, content, and more.
Integrations
Upload file
More ways to add data
In addition to adding integrations, you can try our sample data or upload your own data.
Sample data
Upload file
calculator.log
Import data
Simple
Advanced
Index name
calculatorlog
Create data view
Reset
File processed
Index created
Ingest pipeline created
Data uploaded
Data view created
Import complete
Index
calculatorlog
Data view
calculatorlog
Back
Cancel

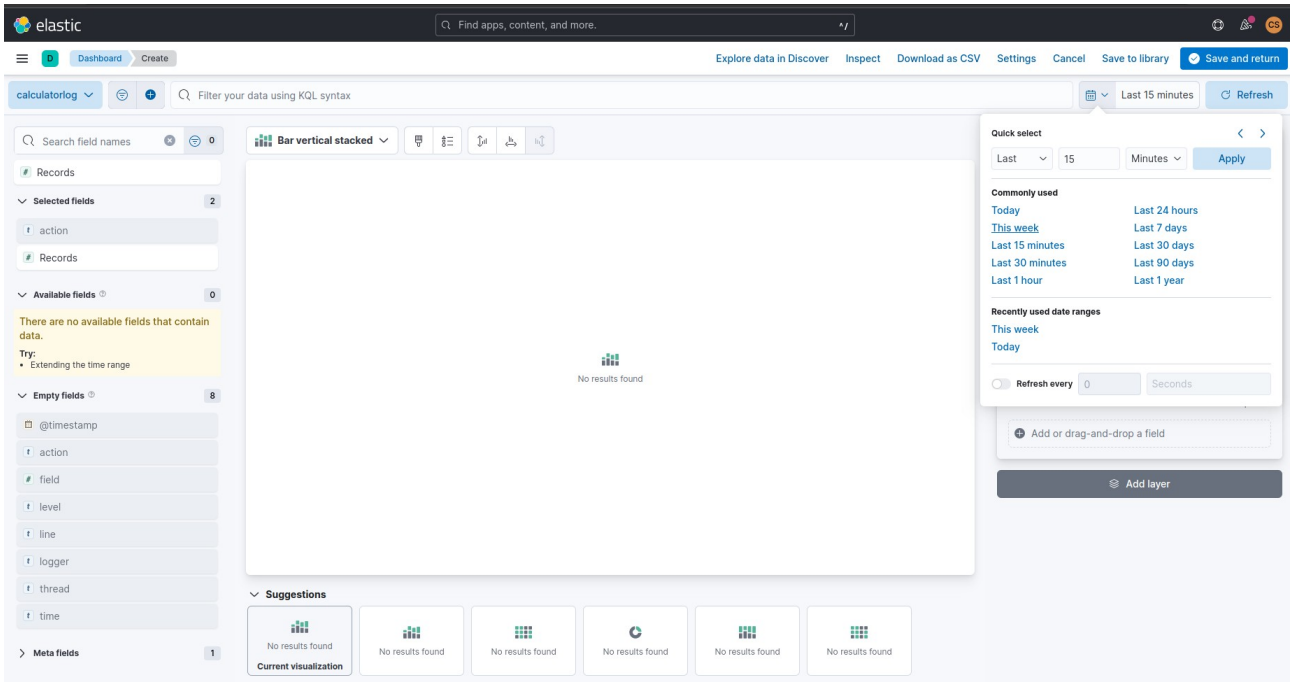
Then go to the dashboard to create a dashboard and then select the kibana

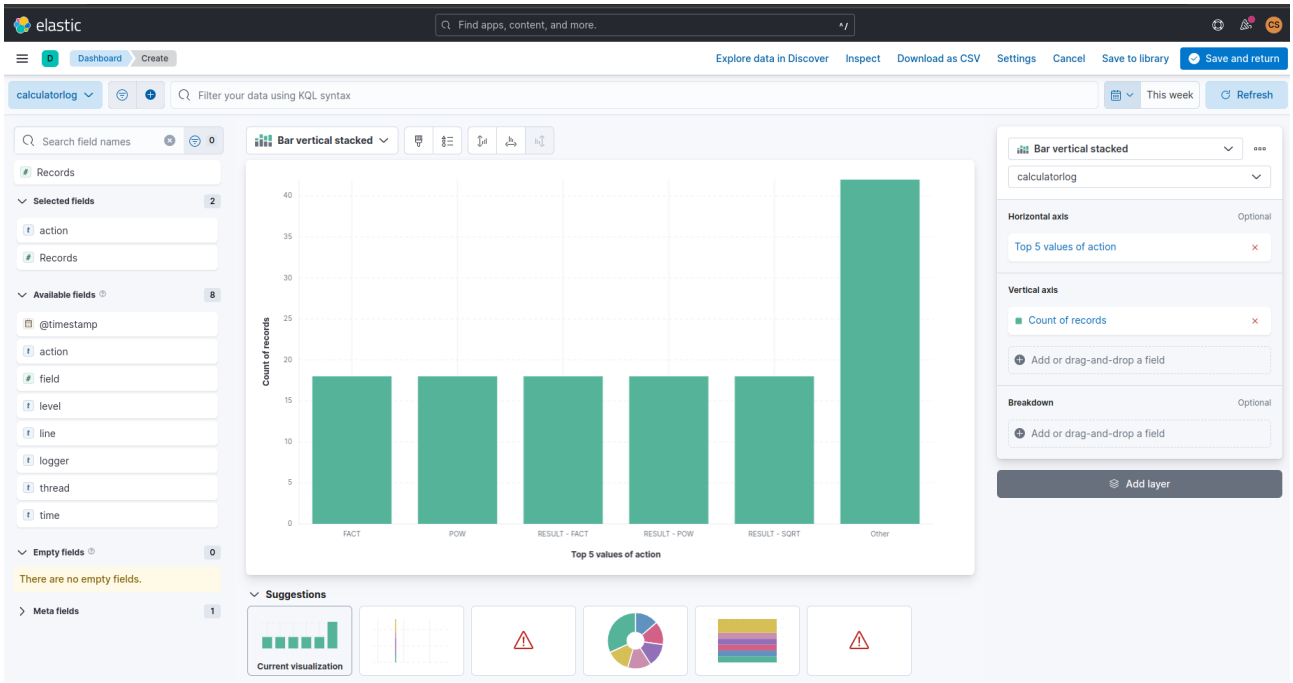




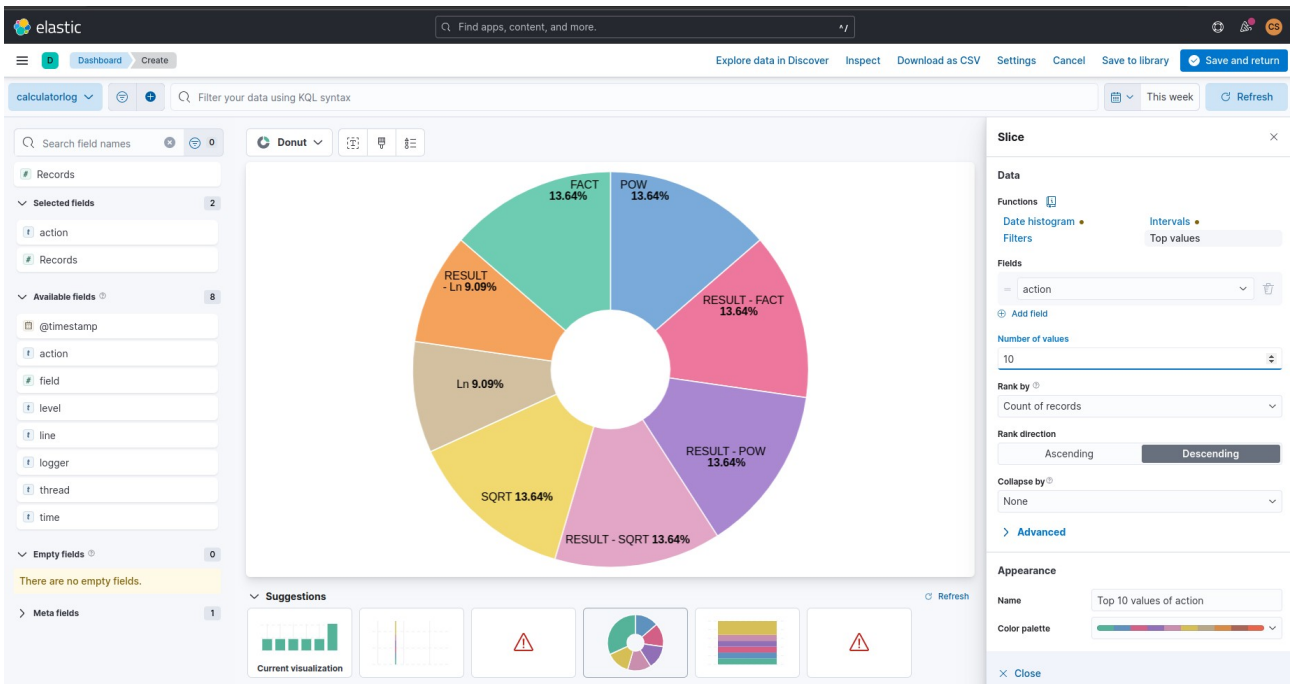


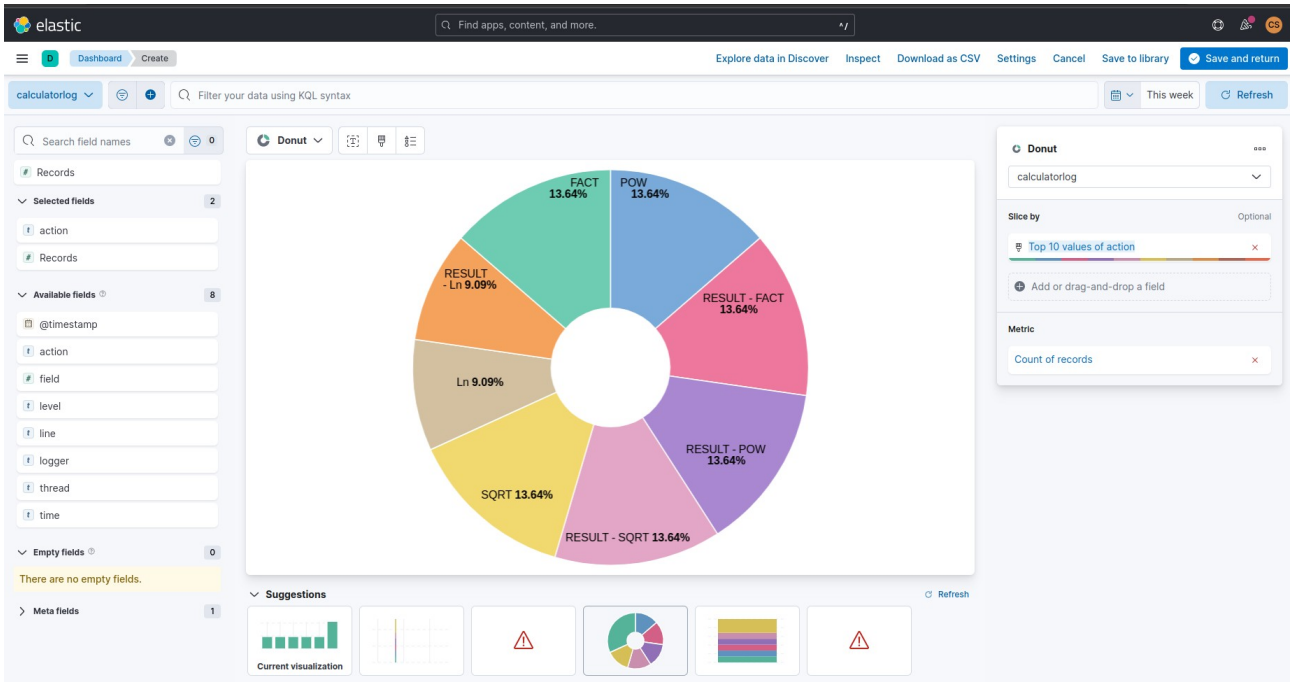
Then drag the *action* to start. In this visualization, we can see how many logs are there for every function.





We can also change the value of records we wanted to see.





Challenges :

1. I got error when I am building docker in Jenkins

Solution : We have to login docker locally using *sudo docker login* and use *sudo usermod -a -G docker jenkins*

This command adds the user "jenkins" to the "docker" group on a Linux-based system.

Here is a breakdown of the command:

- "sudo" is a command used to run a command with administrative or "superuser" privileges.

- "usermod" is a command used to modify user account settings.
- "-a" specifies that the modification is an addition to the existing settings.
- "-G" specifies the group to which the user will be added.
- "docker" is the name of the group to which the user "jenkins" will be added.
- "jenkins" is the name of the user that will be added to the "docker" group.

2. I got errors with JDK while building maven project

Solution : I used one java version and set one version in maven. I changed that.

3. I got error in deployment in ansible step in jenkins

Solution : Use command *sh "/usr/bin/pip3 install docker"*. It installs python docker in jenkins.

Links :

Github link : https://github.com/sivanich9/Calculator
DockerHub link : https://hub.docker.com/repository/docker/sivani4/calculator/general