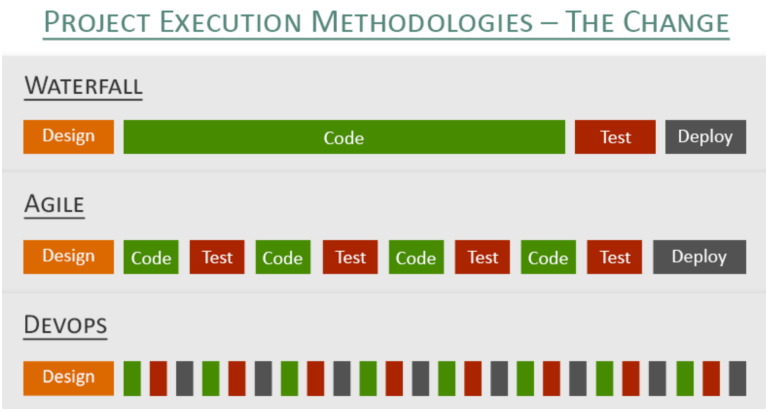
DevOps Calculator (SPE Mini-Project)

This is a Dev-Ops project. The aim of this project was to create a pipeline for continuous integration, continuous delivery and continuous deployment. The idea to demonstrate is that if there is a change that is made,in the code, when it is pushed to a remote version control system like GitHub, it gets tested, built, delivered and deployed using automation, without any human intervention.

What is DevOps? And Why DevOps?

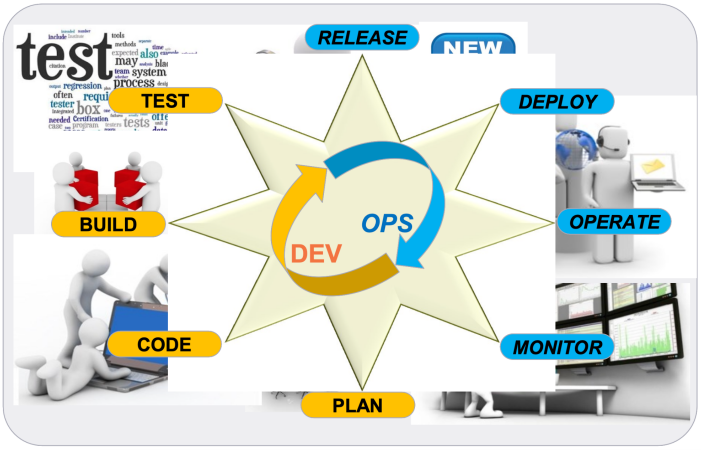
Using the DevOps methodology, software development teams and IT operations teams are encouraged to work together and communicate. DevOps aims to improve both the software's quality and the speed and efficiency with which it is developed and deployed.

Infrastructure automation, continuous integration, and continuous delivery are just a few of the techniques and tools used in DevOps. Continuous integration allows for early detection of integration errors by having developers frequently merge their code changes into a central repository. Automating the software deployment process through continuous delivery enables prompt and dependable release of new features and updates. Utilizing tools to automatically configure and manage IT infrastructure, such as servers and networks, is known as infrastructure automation.



DevOps promotes a culture of cooperation, communication, and shared accountability amongst developers, operations teams, and other stakeholders in addition to these methods and technologies. This entails regular meetings, feedback loops, and a concentration on ongoing learning and development.

Ultimately, the goal of DevOps is to eliminate barriers between development and operations teams and to establish a setting where software is built and deployed more quickly, dependably, and effectively.



Type of Project:

Command Line Application

Language Used:

Java

Tools used

1. Source Code Management Tools - GitHub, Git
2. Testing - Tested the code using either JUnit
3. Build - Built the code using Maven
4. Continuous Integration - Continuous integrated the code using Jenkins
5. Containerize - used Docker
6. Push the created Docker image to Docker hub.
7. Deployment - Did configuration management and deployment using either Ansible. Also used it to pull the docker image and ran it on the localhosts.
8. Monitoring - for monitoring used the ELK stack (Using a log file to do the monitoring). Generated the log file in the mini project and passed it in the ELK stack.

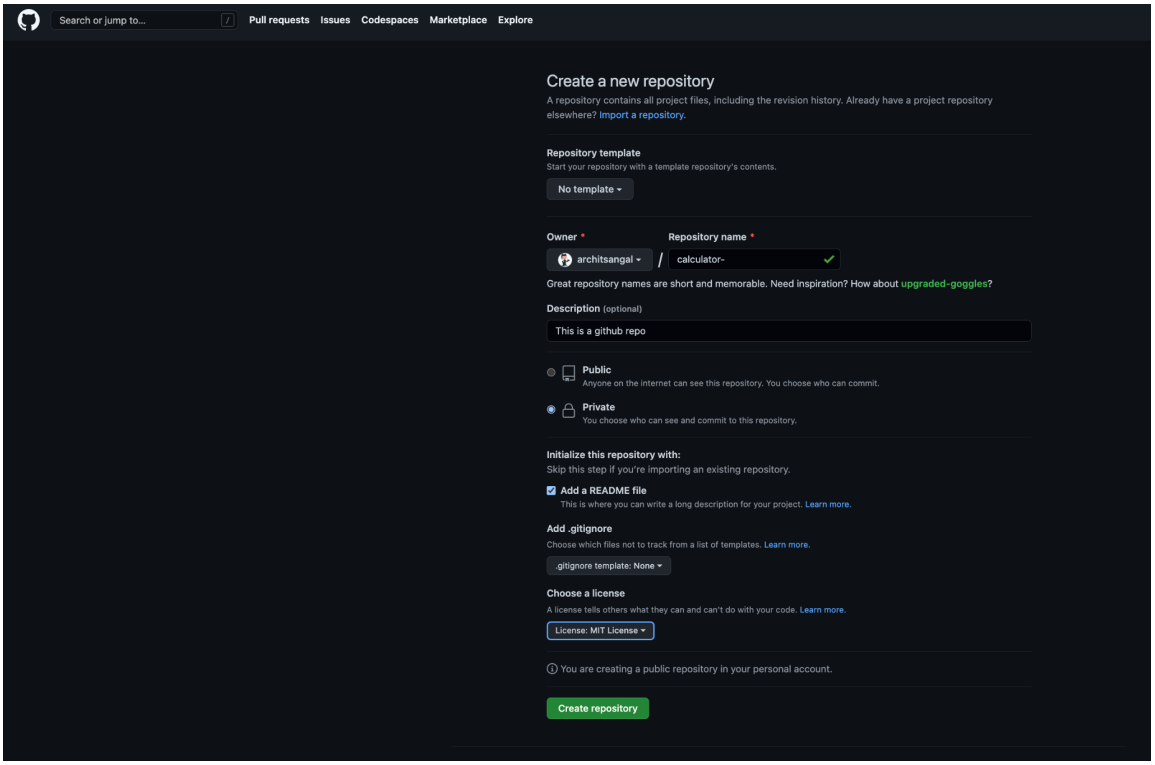
How to setup the components?

This code is tested in the MacOS environment (MacOS 13.2.1). It may not work in other environments. Also, if you are stuck in any part of the process, you may also want to refer to the Resources folder. They are the resources that I used for solving my errors and getting on board.

1. Have a GitHub Repository with Read and Write Access

- Link to the GitHub Repository - https://github.com/architsangal/calculator

Either make a fork of this repository or make a new repository on GitHub. If you choose to make a new repository,



Upload the code from this repository, to the newly made repository.

2. Assumptions

I am assuming the following installations are already in place in your local system -

- 'git' version control system

```
$ git --version
git version 2.39.0
```

- Java 11

```
$ java --version
java 11.0.16.1 2022-08-18 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.16.1+1-LTS-1)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.16.1+1-LTS-1, mixed mode)
```

- 'Jenkins' basic setup, where you can login and create a pipeline job.

```
$ whereis jenkins-lts
jenkins-lts: /opt/homebrew/bin/jenkins-lts
```

- 'Docker' is installed in the local system.

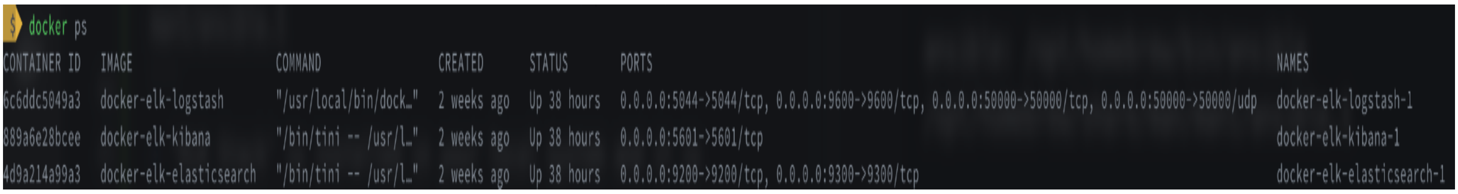
```
$ whereis docker
docker: /usr/local/bin/docker
```

- 'Ansible' is installed in your local system.

```
$ whereis ansible
ansible: /opt/homebrew/bin/ansible /opt/homebrew/share/man/man1/ansible.1
```

- 'ELK Stack' is installed in your local system and it is running.

```
$ docker ps
```



$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6c6ddc5049a3	docker-elk-logstash	"/usr/local/bin/dock…"	2 weeks ago	Up 38 hours	0.0.0.0:5044->5044/tcp, 0.0.0.0:9600->9600/tcp, 0.0.0.0:50000->50000/tcp, 0.0.0.0:50000->50000/udp	docker-elk-logstash-1
889a6e28bcee	docker-elk-kibana	"/bin/tini -- /usr/l…"	2 weeks ago	Up 38 hours	0.0.0.0:5601->5601/tcp	docker-elk-kibana-1
4d9a214a99a3	docker-elk-elasticsearch	"/bin/tini -- /usr/l…"	2 weeks ago	Up 38 hours	0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp	docker-elk-elasticsearch-1

- 'IntelliJ IDEA CE' is installed in the system.

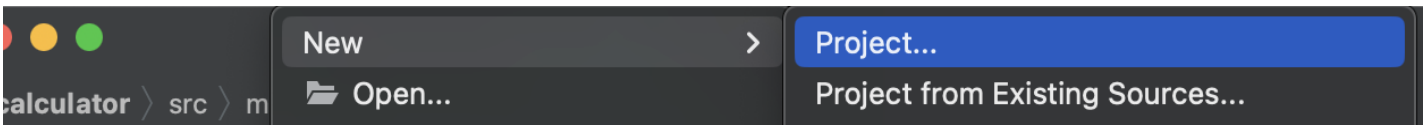
- 'ngrok' is installed on the local system.

The versions of the above softwares may vary upto some extent and depending on how you installed them, using **`brew`** or **`.app`** files, etc.

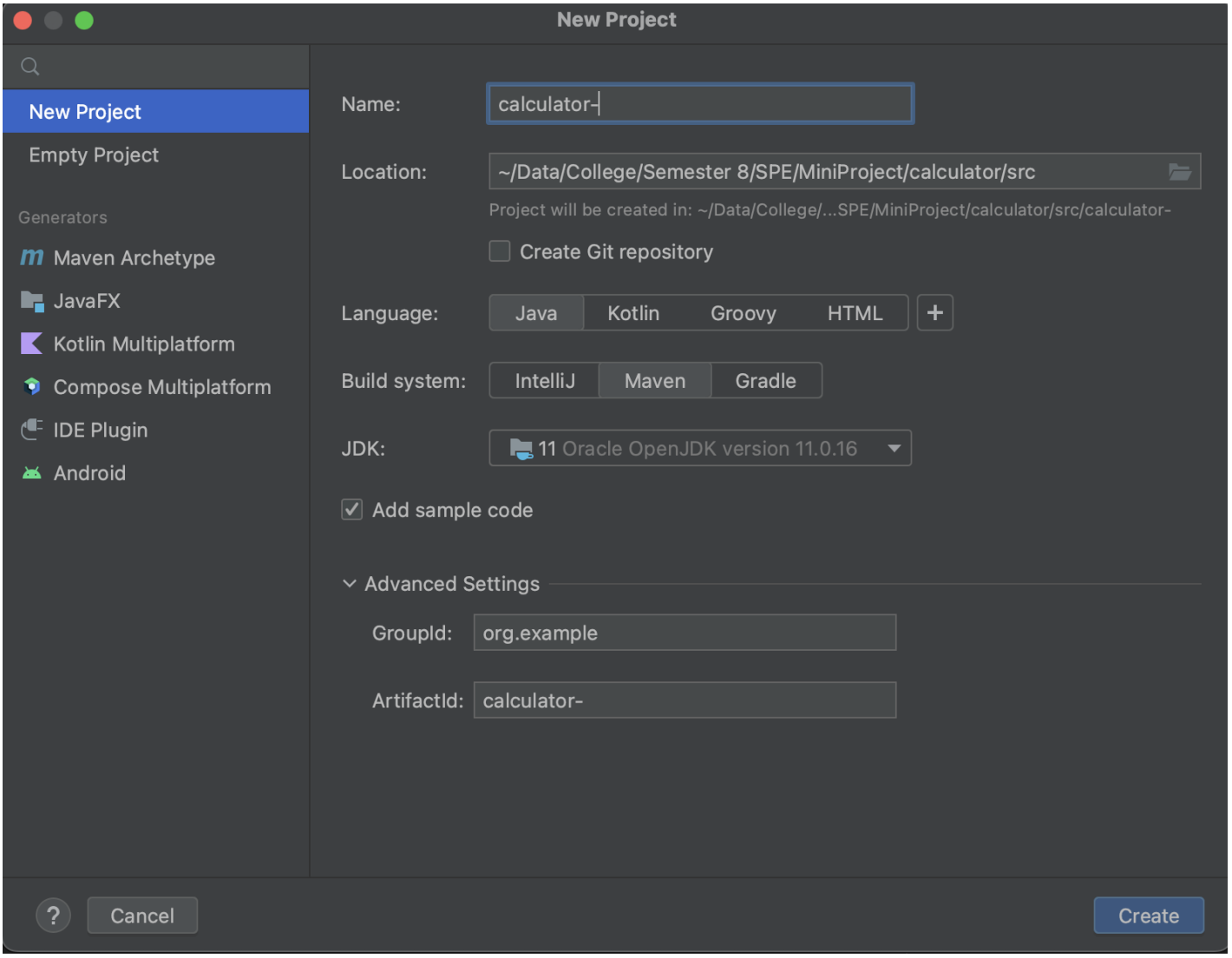
3. New Maven Project

If you have your own code or you want to start from scratch.

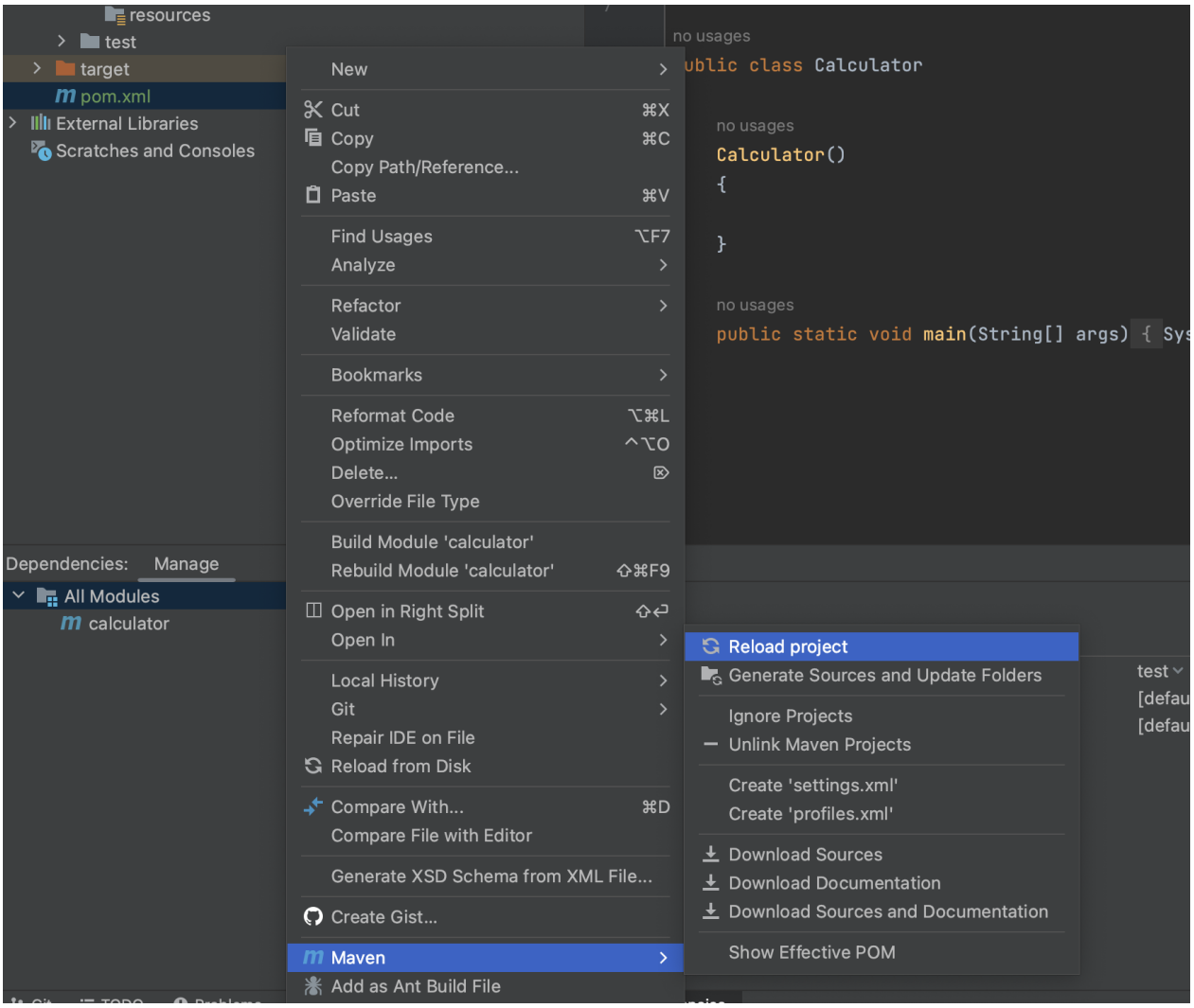
- Make a new maven project.



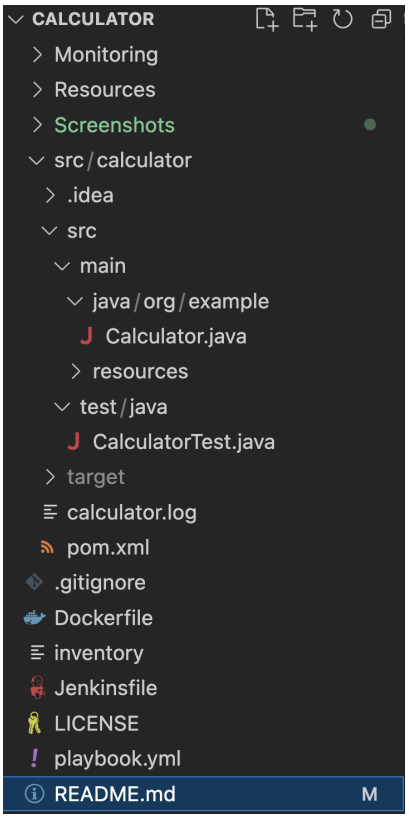
- Select the configurations given below-



- Add your code in the respective files.
- Code is a standard maven project with a test module with pom.xml managing the dependencies. See the project directory.
- Update the projects with the dependencies which you have included in your pom.xml file using the procedure given below -

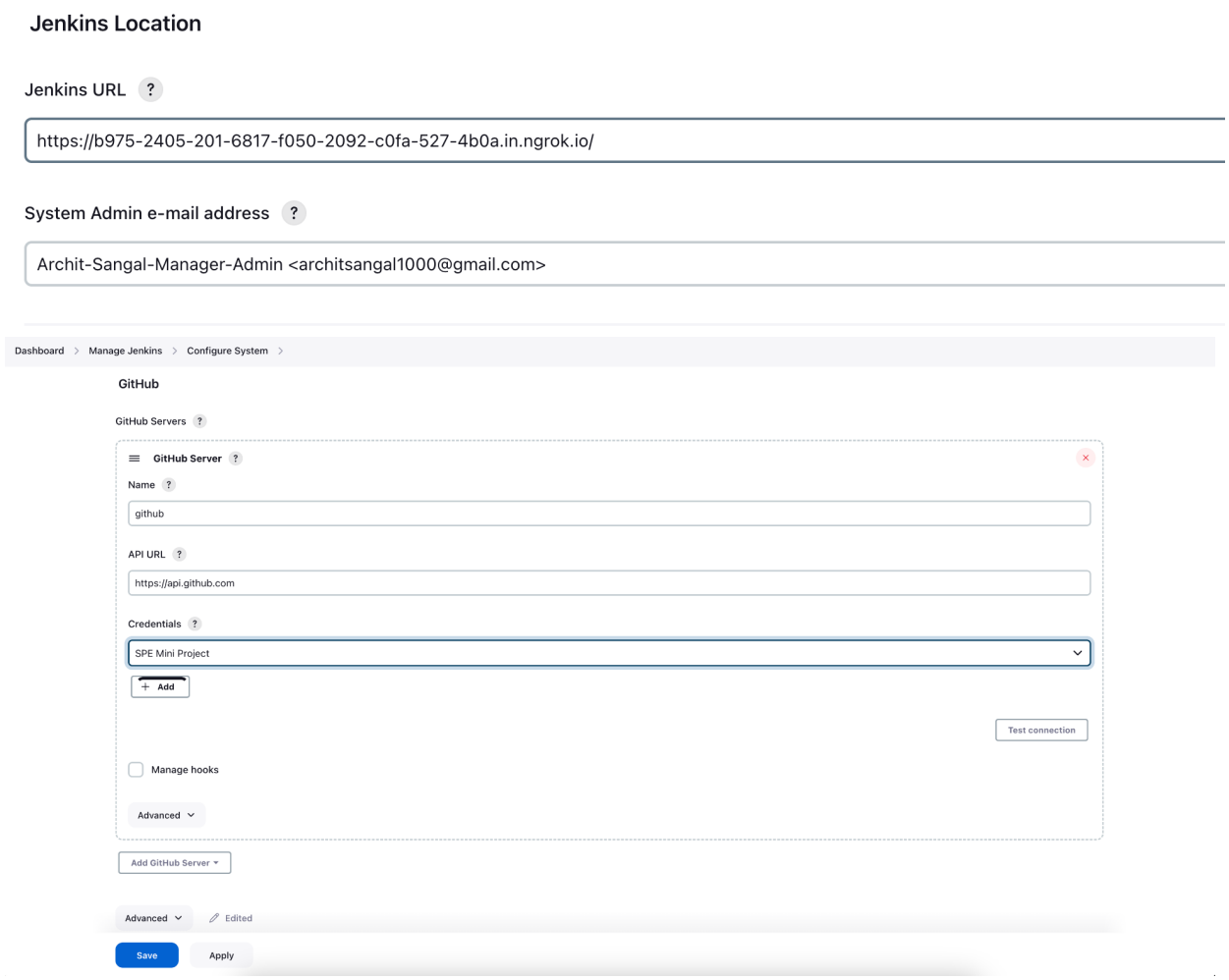


Our Current Directory Structure Looks like this-

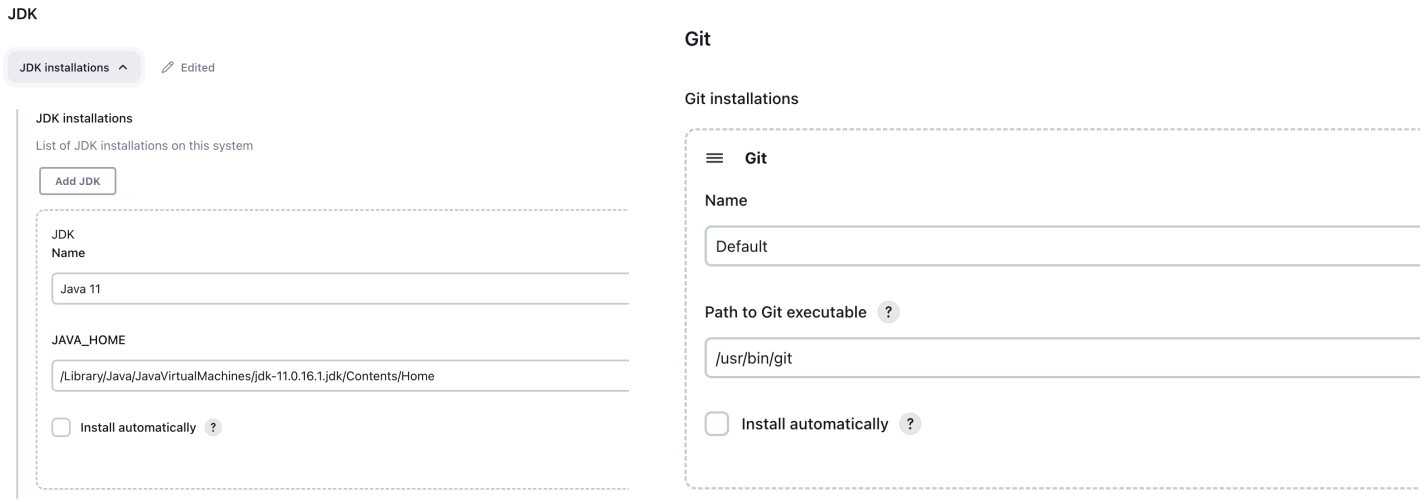


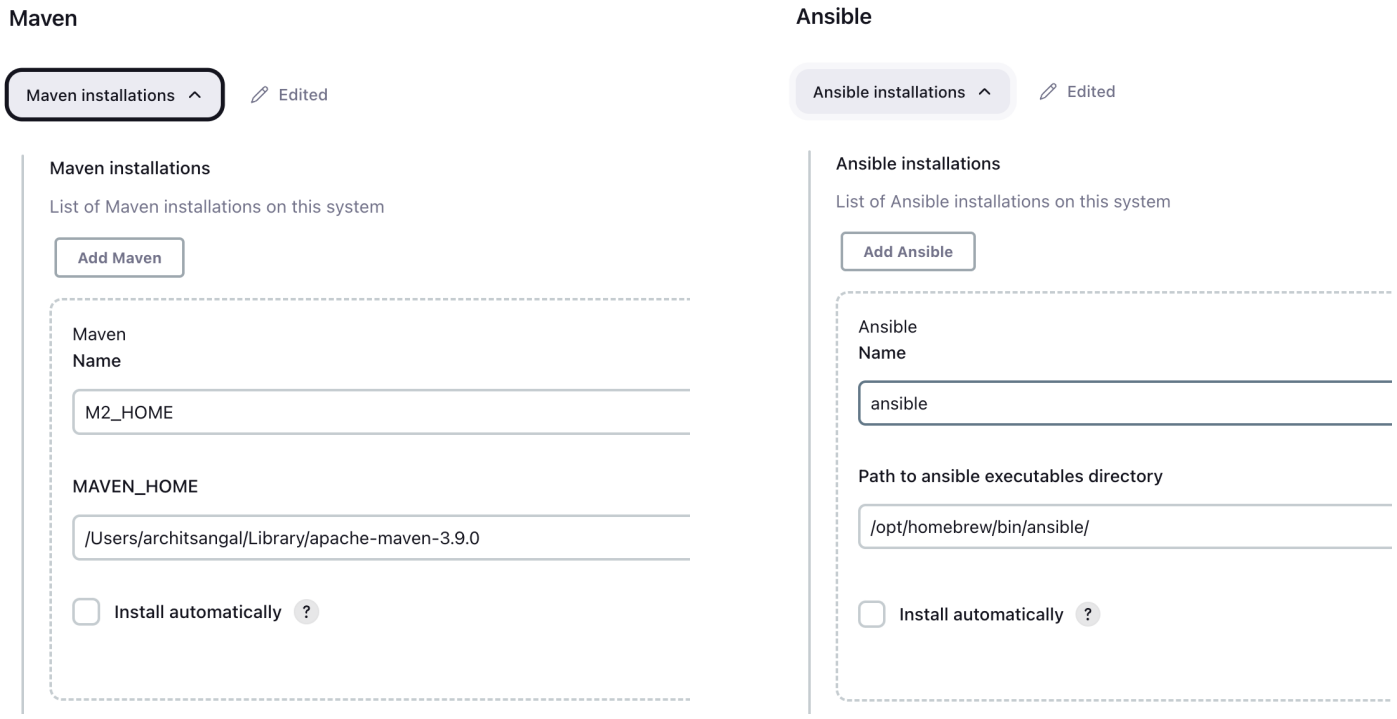
4. Jenkins Setup

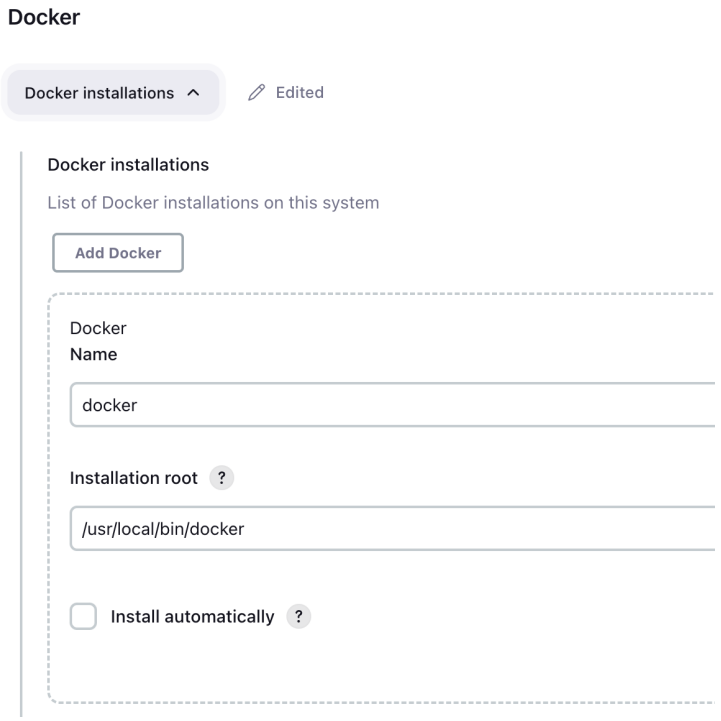
- Login into the Jenkins.
- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Configure System, and replicate the below -



- SPE Mini Project is a secret text, which is for GitHub WebHook.
- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Configure Global Tools, and replicate the below -







- Go to celler repo of jenkins and add path to it's **`homebrew.mxcl.jenkins-lts.plist`** file.

```
$ cd /opt/homebrew/Cellar/jenkins-lts/
$ cd 2.375.3 #some version
$ ls
```

There you will find a file `homebrew.mxcl.jenkins-lts.plist`. Add path to it -

```
<key>EnvironmentVariables</key>
<dict>
    <key>PATH</key>
<string>/opt/homebrew/bin/:/usr/local/bin/:/usr/local/bin/docker:/usr/local/bin:/usr/bi
n:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/:/Users/admin/Li
brary/Group\ Containers/group.com.docker</string>
</dict>
```

After the above step, `homebrew.mxcl.jenkins-lts.plist` shpuld look something like this-

```
<?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/local/bin/docker:/usr/local/bin:/usr/bi
n:/bin:/usr/sbin:/sbin:/Applications/Docker.app/Contents/Resources/bin/:/Users/admin/Li
brary/Group\ Containers/group.com.docker</string>
    </dict>
</dict>
</plist>
```

- Follow the path: Jenkins Dashboard -> Manage Jenkins -> Manage Credentials, add the keys and user name, mine looks like the following:

Credentials

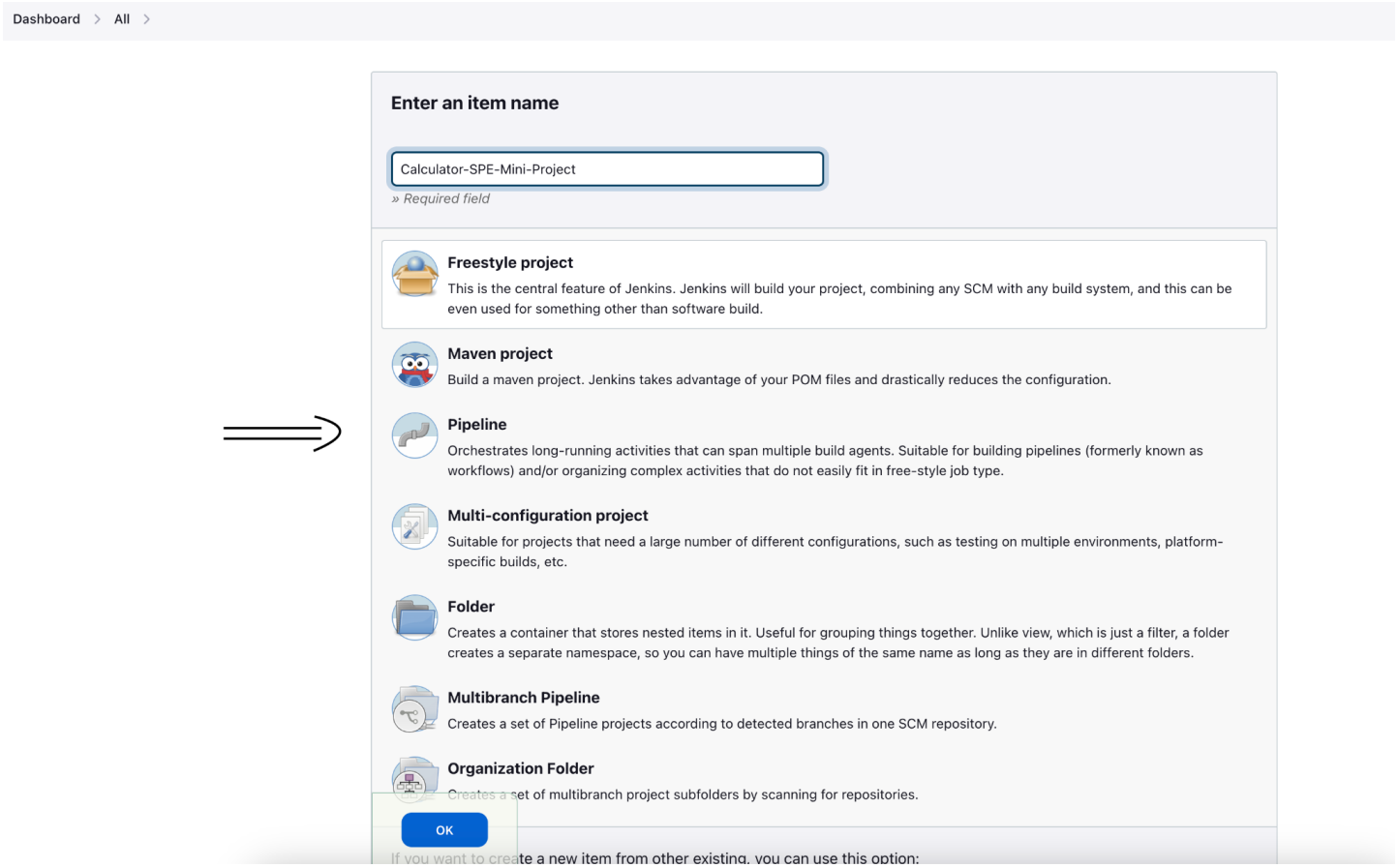
T	P	Store ↓	Domain	ID	Name
		System	(global)	49654976-14f2-4c60-aeb6-4804fc359cec	Secret text
		System	(global)	8ad63d4d-7c23-43a4-8638-47f78b28ace5	Secret text
		System	(global)	Master_Jenkins_Private_key	jenkins (Jenkins Master Private Key to Add Multiple Agents)
		System	(global)	dockerhub	architsangal/****** (DockerHub Credentials)
		System	(global)	2cc913cf-8c05-4c83-988d-42c1451aedd1	SPE Mini Project

Stores scoped to Jenkins

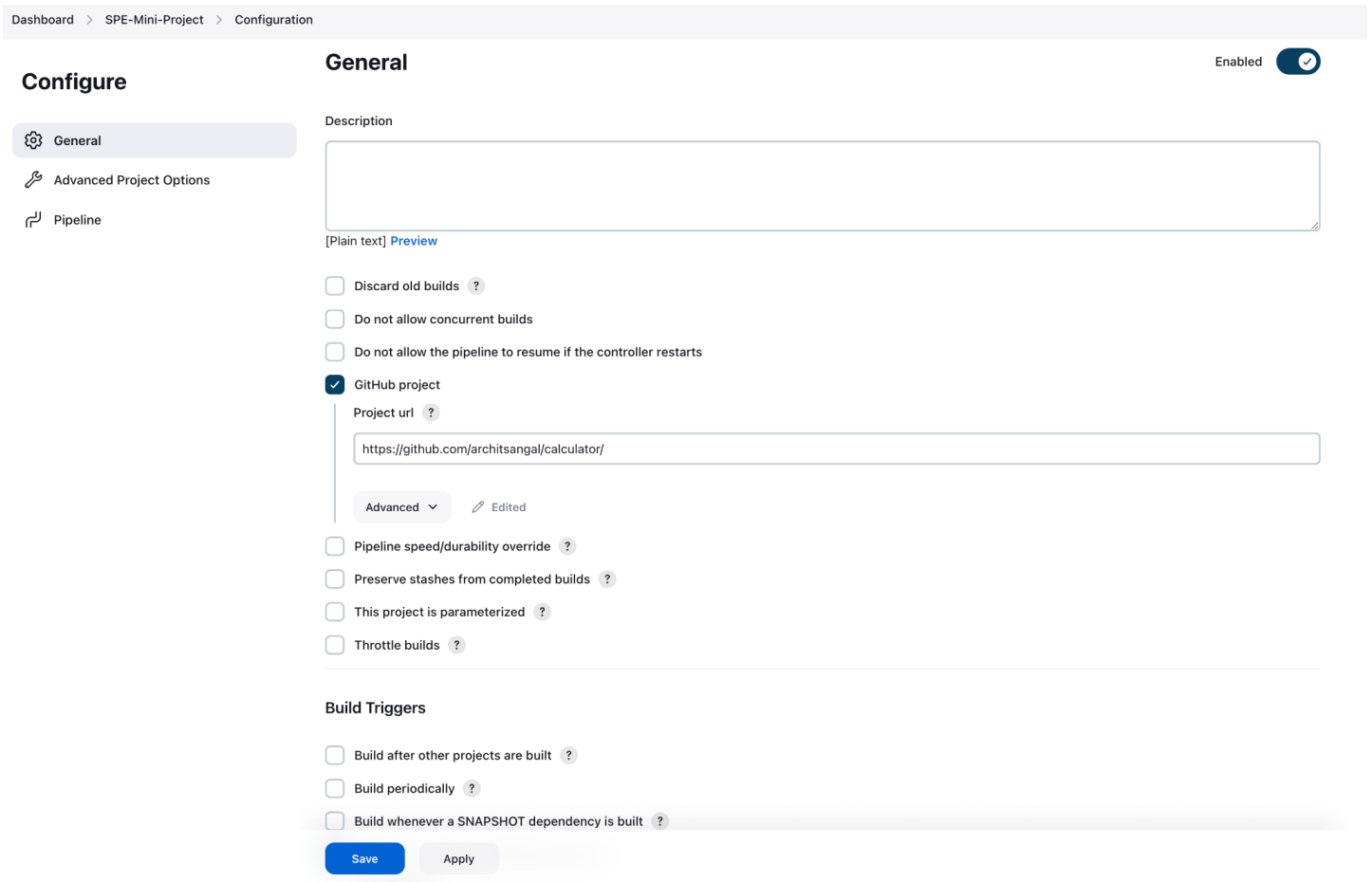
P	Store ↓	Domains
	System	(global)

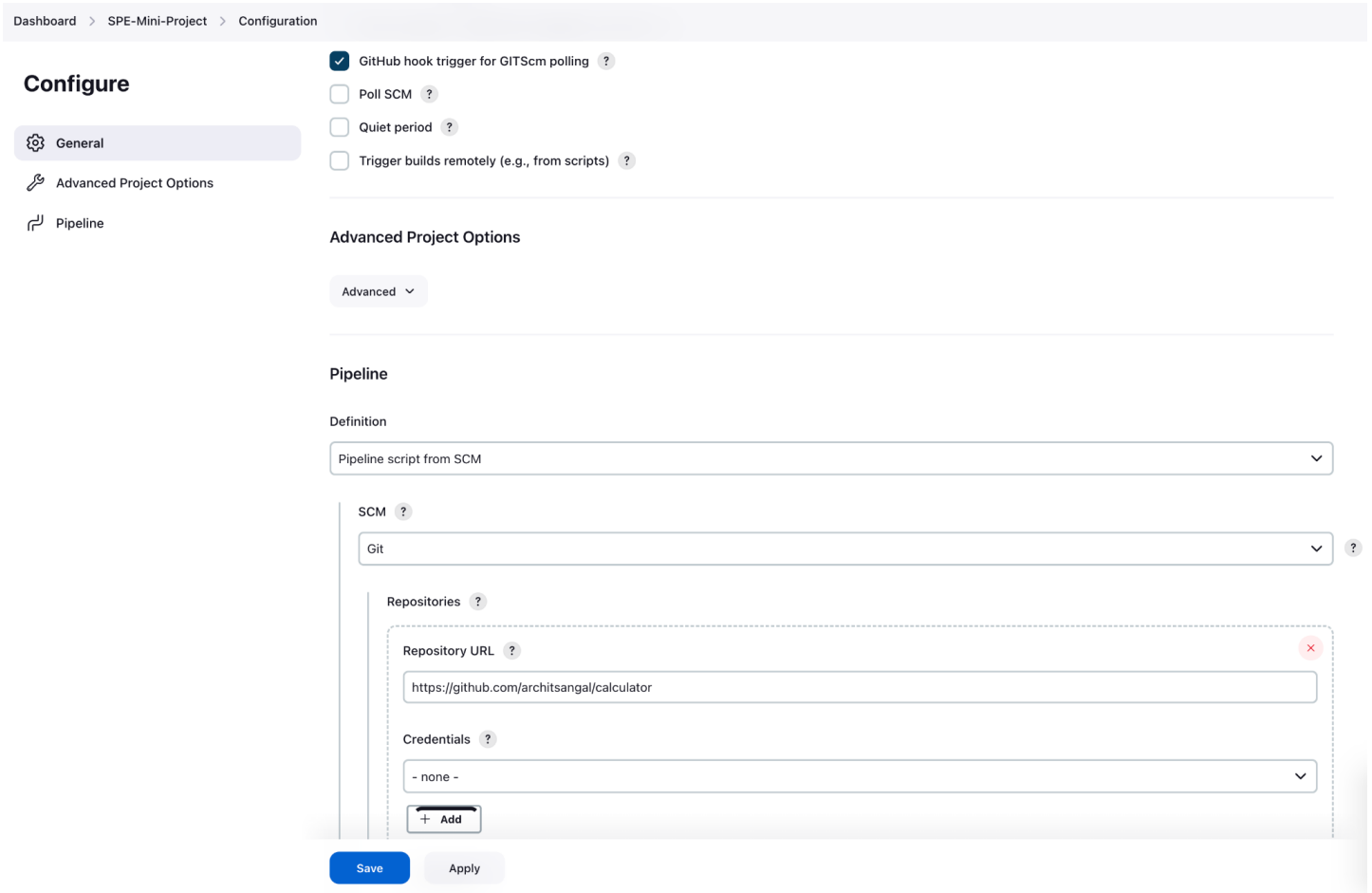
5. Jenkins Item Setup

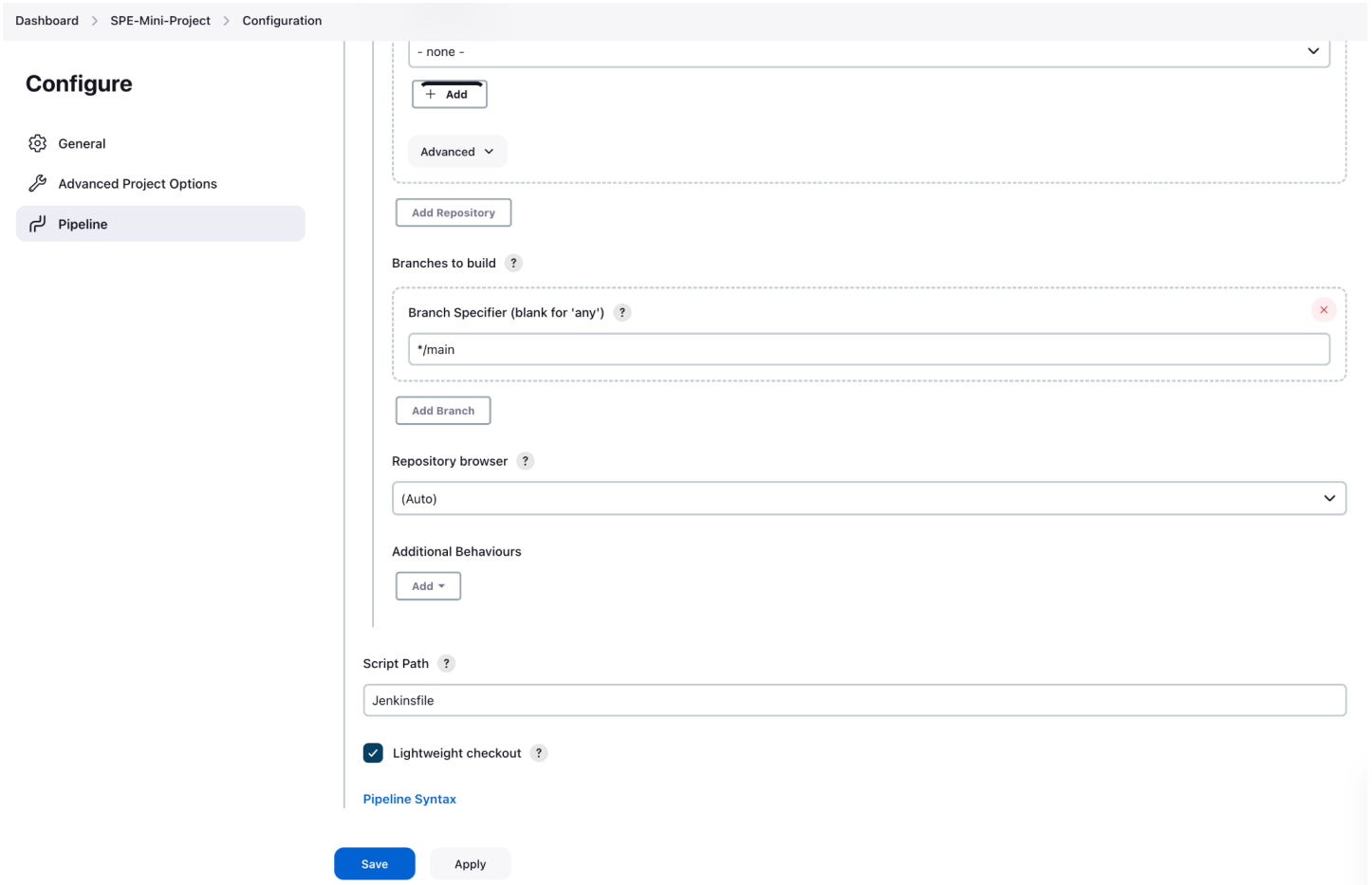
- Add a new Item of type `Pipeline`



- Go to Configure Section, and replicate the following:

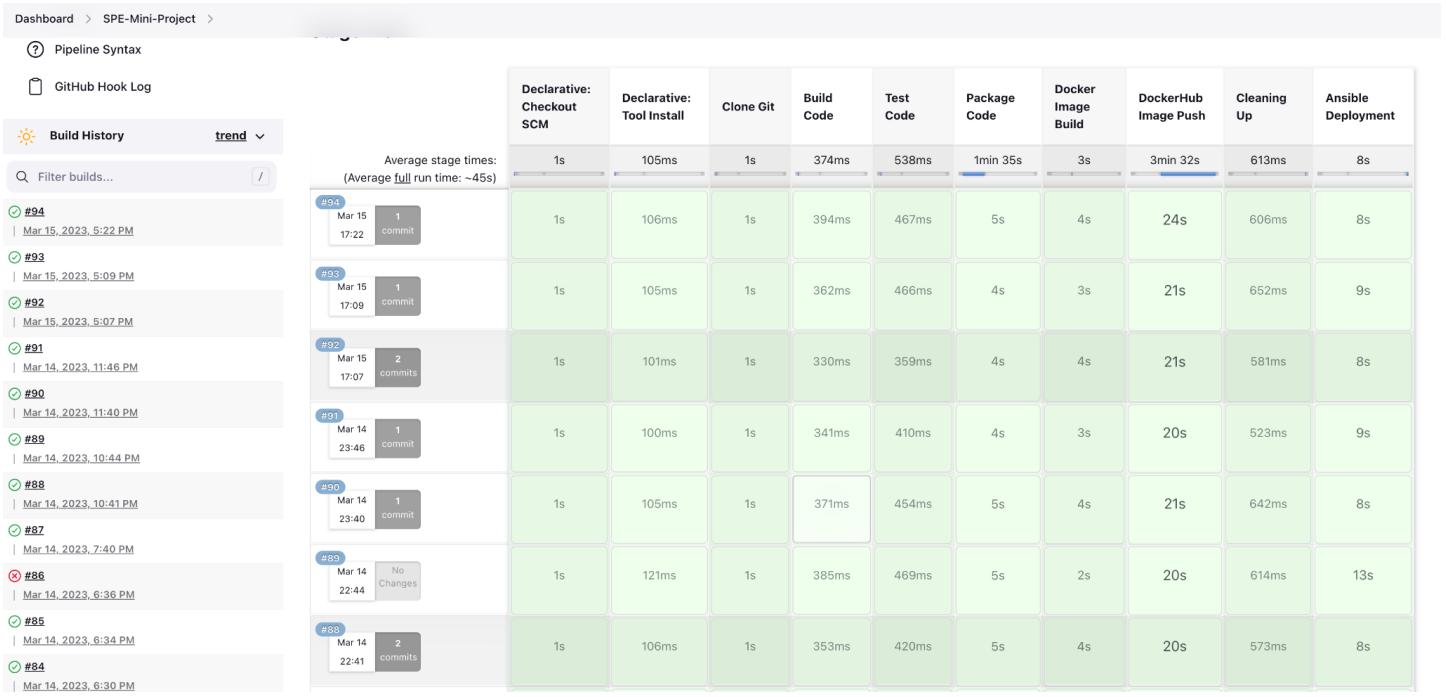

Dashboard > SPE-Mini-Project > Configuration
Configure
General
Advanced Project Options
Pipeline
General
Enabled
Description
[Plain text] Preview
Discard old builds
Do not allow concurrent builds
Do not allow the pipeline to resume if the controller restarts
GitHub project
Project url
https://github.com/architsangal/calculator/
Advanced
Edited
Pipeline speed/durability override
Preserve stashes from completed builds
This project is parameterized
Throttle builds
Build Triggers
Build after other projects are built
Build periodically
Build whenever a SNAPSHOT dependency is built
Save
Apply


Dashboard > SPE-Mini-Project > Configuration
Configure
General
Advanced Project Options
Pipeline
GitHub hook trigger for GITScm polling
Poll SCM
Quiet period
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Advanced
Pipeline
Definition
Pipeline script from SCM
SCM
Git
Repositories
Repository URL
https://github.com/architsangal/calculator
Credentials
- none -
Add
Save
Apply



6. Jenkins Pipeline

- start the 'jenkins' service.
- Click On build Now, you have your pipeline.



The pipeline code below includes the tools and sets up the environment for the jenkins pipeline.

```
pipeline
{
    tools {
        maven "M2_HOME"
    }

    agent any
    environment
    {
        registry = "architsangal/speminiproject"
        registryCredential = "dockerhub" // links the credentials
        dockerImage = ""
    }
    stages
    {
```

The code below is a Jenkins stage in pipeline. It clone the main branch of the calculator repo.

```
stage('Clone Git')
        {
            steps
            {
                git branch: 'main',
                url: 'https://github.com/architsangal/calculator.git'
            }
        }
```

The code below is a Jenkins stage in pipeline. It builds the clone code using maven. Many feature are not used in this pipeline, hence they are switched off so that don't produce any logs that are misleading.

```
stage('Build Code')
        {
            steps
            {
                dir("src/calculator/")
                {
                    withMaven(options: [
                        openTasksPublisher(disabled: true),
                        dependenciesFingerprintPublisher(disabled: true),
                        artifactsPublisher(disabled: true),
                        //junitPublisher(disabled: true),
                        jgivenPublisher(disabled: true),
                        invokerPublisher(disabled: true),
                        findbugsPublisher(disabled: true),
                        concordionPublisher(disabled: true),
```

```
                    pipelineGraphPublisher(disabled: true)
                    ]
                )
                {
                    // TODO uncomment this
                    sh "mvn clean"
                    //sh "mvn validate"
                    sh "mvn compile"
                }
            }
        }
    }
```

```
    stage('Test Code')
    {
        steps
        {
            dir("src/calculator/")
            {
                withMaven(options: [
                    openTasksPublisher(disabled: true),
                    dependenciesFingerprintPublisher(disabled: true),
                    artifactsPublisher(disabled: true),
                    //junitPublisher(disabled: true),
                    jgivenPublisher(disabled: true),
                    invokerPublisher(disabled: true),
                    findbugsPublisher(disabled: true),
                    concordionPublisher(disabled: true),
                    pipelineGraphPublisher(disabled: true)
                    ]
                )
                {
                    // TODO uncomment this
                    sh "mvn test"
                    //sh "mvn package"
                    //sh "mvn verify"
                }
            }
        }
    }
```

The code below is a Jenkins stage in pipeline. It builds the clone code using maven. Many feature are not used in this pipeline, hence they are switched off so that don't produce any logs that are misleading. below code package the files from the code using maven.

```
stage('Package Code')
        {
            steps
            {
                dir("src/calculator/")
                {
                    withMaven(options: [
                        openTasksPublisher(disabled: true),
                        dependenciesFingerprintPublisher(disabled: true),
                        artifactsPublisher(disabled: true),
                        //junitPublisher(disabled: true),
                        jgivenPublisher(disabled: true),
                        invokerPublisher(disabled: true),
                        findbugsPublisher(disabled: true),
                        concordionPublisher(disabled: true),
                        pipelineGraphPublisher(disabled: true)
                        ]
                    )
                    {
                        // TODO uncomment this
                        sh "mvn package"
                        //sh "mvn verify"
                    }
                }
            }
        }
```

Below code makes the image of the current code and package generated and uses the credentials stored in jenkins to push the image to docker hub.

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

```
                {
                    docker.withRegistry('', registryCredential)
                    {
                        dockerImage.push()
                    }
                }
            }
        }
```

This step can be refined. It untags the previous image and tries to delete it.

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

Runs the ansible inventory with name of the file as ‘inventory’ and playbook with name ‘playbook.yml’.

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

7. Ansible

```
[host_machine]
localhost ansible_connection=local
```

The code above is the inventory code. This file keeps a track of group, on which we may do the configuration management.

Code below the playbooks code, it the tasks on all the hosts (here all, which is the local machine) mentioned. The tasks are to pull the image from the docker hub. It also checks if the docker process is running. Then it frees the common resources like the containers with the port number or with the name calculator. Then it starts the named container calculator.

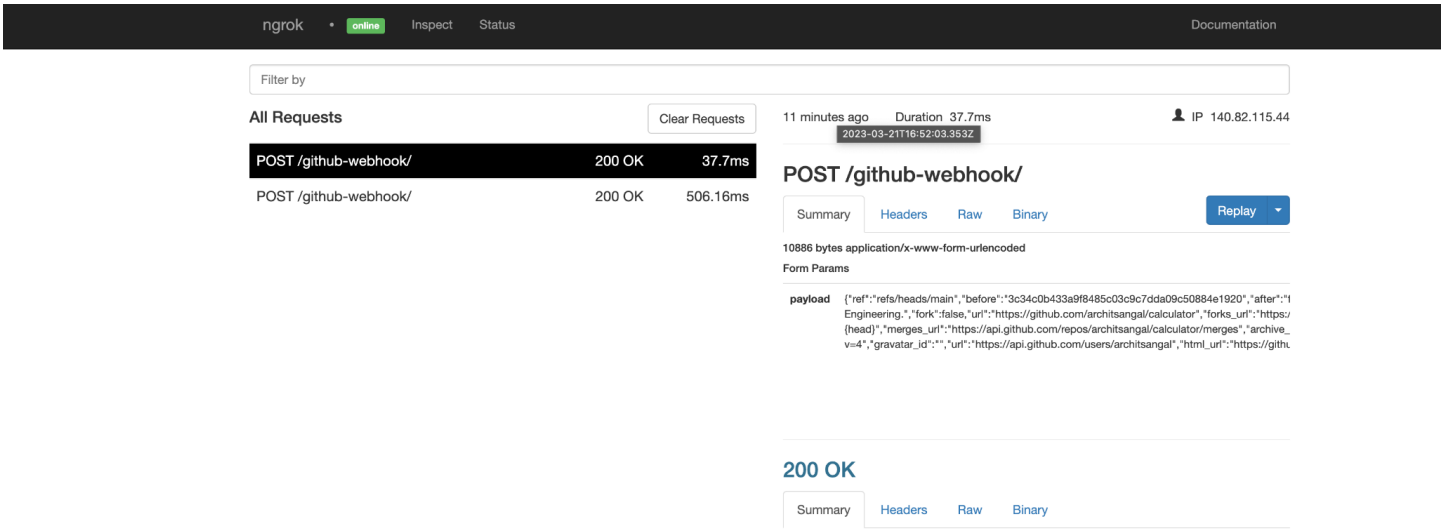
```
---
- name: Pull Docker Image
  hosts: all
  tasks:
    - name: pull an image
      docker_image:
        name: architsangal/speminiproject:latest
        source: pull
    - name: Freeing the common resources
      shell: docker ps -aq --filter publish="2023" | xargs docker stop | xargs
docker rm
      shell: docker ps -aq --filter name="calculator" | xargs docker stop |
xargs docker rm
    - name: Running Docker Image Using a Container
      shell: docker run -p 2023:22 --name calculator -it -d
architsangal/speminiproject
```

7. Start ngrok

```
ngrok http 8080
```

- Setup the GitHub WebHook. Now you don't need to do 'Build Now' step again and again, after every change.

We can also look at the web interface, for the same:

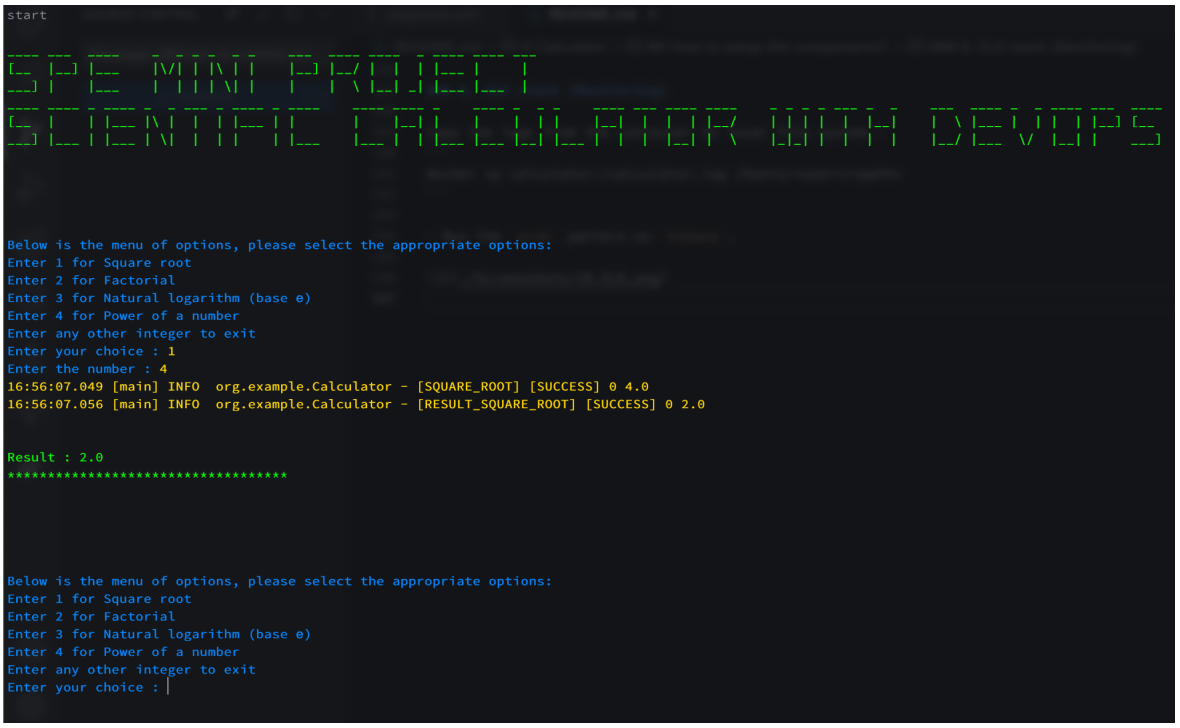


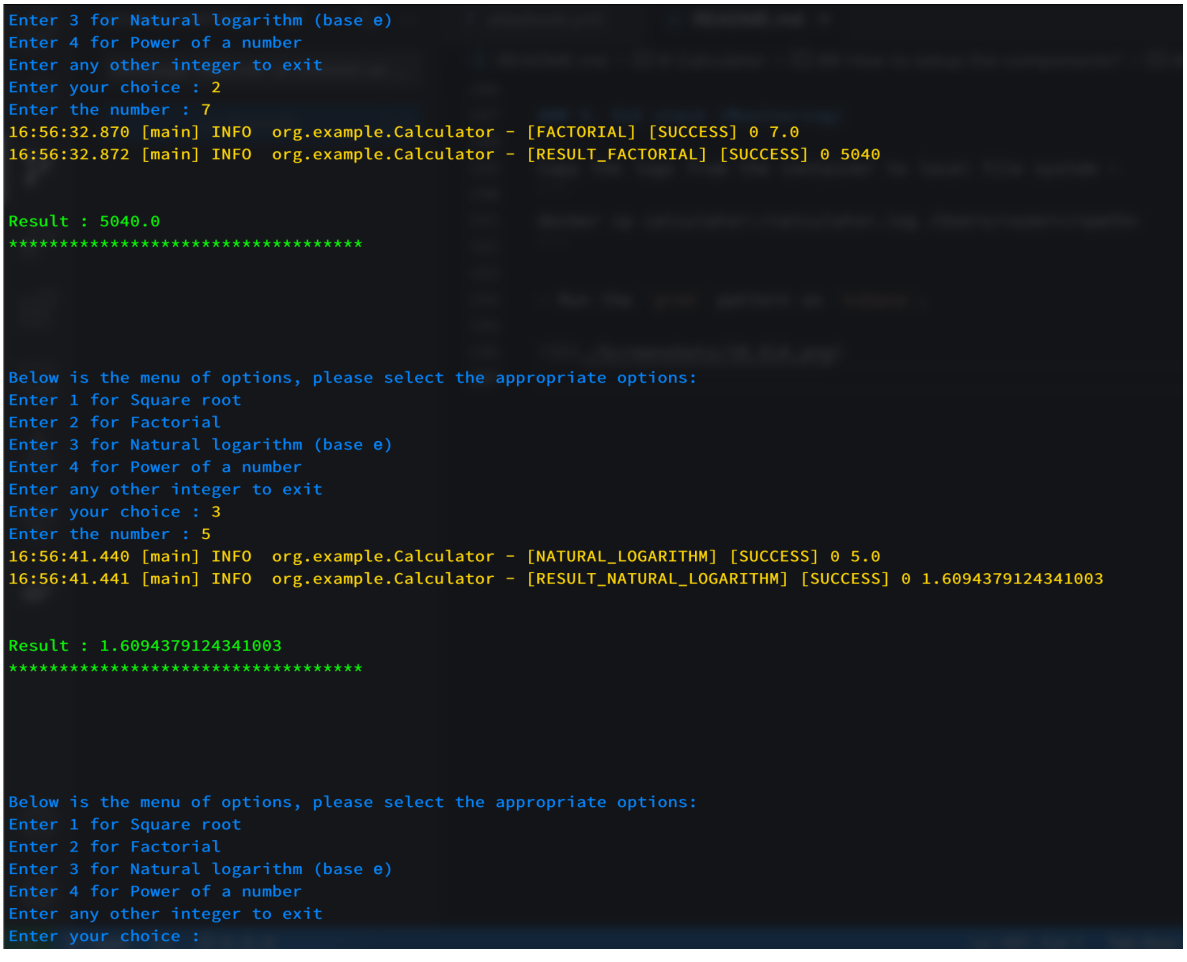
8. Connect to the server

- Ansible has deployed the file in a container, which represents a server. If it was a remove, server, we could have used **`ssh`** but as it is on the same system, we directly use the following command-

```
$ docker attach calculator # as the container is a named container.
```

After you are done using the calculator, stop it by pressing 5.





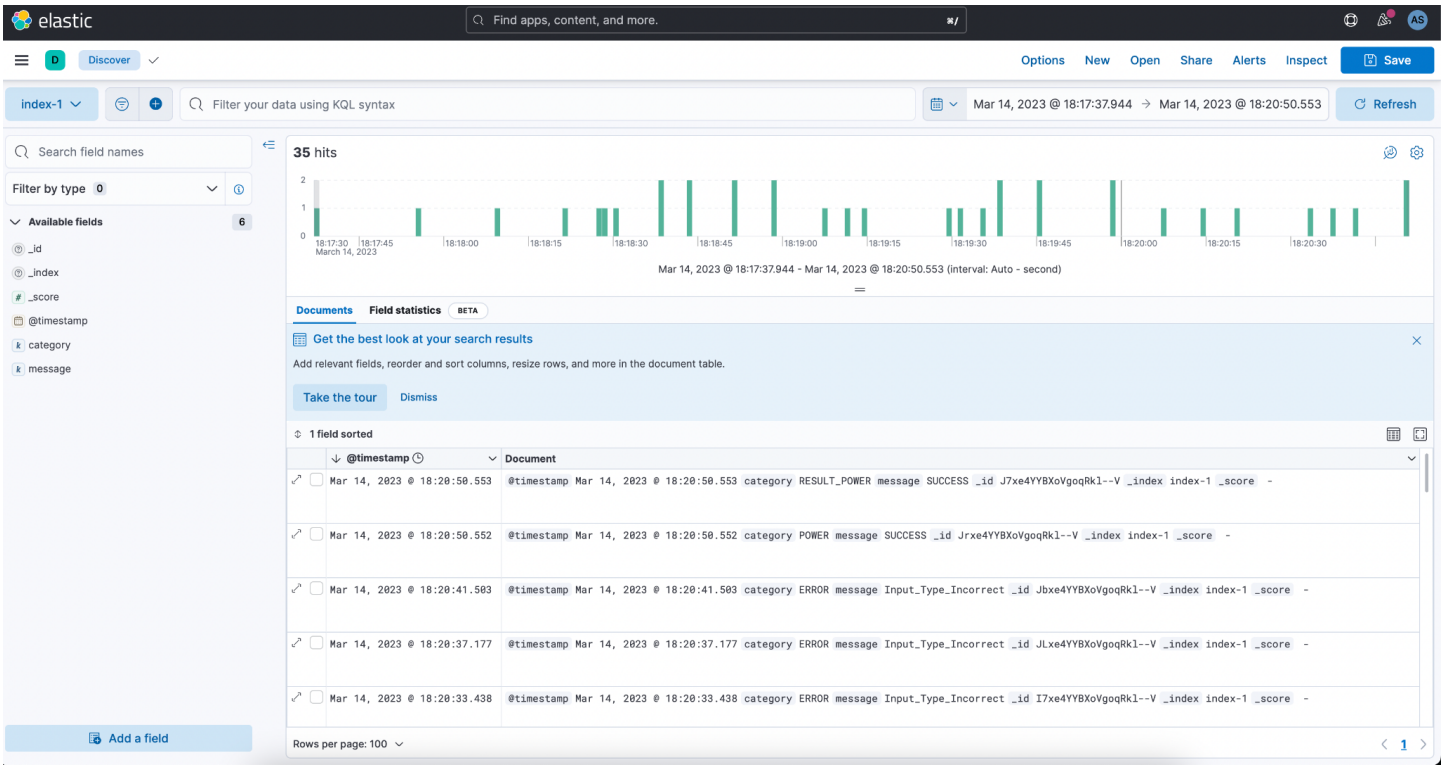
```
16:56:41.441 [main] INFO  org.example.Calculator - [RESULT_NATURAL_LOGARITHM] [SUCCESS] 0 1.6094379124341003

Result : 1.6094379124341003
**********************************



Below is the menu of options, please select the appropriate options:
Enter 1 for Square root
Enter 2 for Factorial
Enter 3 for Natural logarithm (base e)
Enter 4 for Power of a number
Enter any other integer to exit
Enter your choice : 4
Enter the base number : 5
Enter the base number : 4
16:57:05.720 [main] INFO  org.example.Calculator - [POWER] [SUCCESS] 5.0 4.0
16:57:05.721 [main] INFO  org.example.Calculator - [RESULT_POWER] [SUCCESS] 0 625.0


Result : 625.0
**********************************



Below is the menu of options, please select the appropriate options:
Enter 1 for Square root
Enter 2 for Factorial
Enter 3 for Natural logarithm (base e)
Enter 4 for Power of a number
Enter any other integer to exit
Enter your choice : 5


Exiting The Scientific Calculator with DevOps....
```

9. ELK stack (Monitoring)

Copy the logs from the container to local file system -

```
docker cp calculator:/calculator.log /Users/<user>/<path>
```

- Run the **grok** pattern on **kibana**.



Demo Video Link:

Video Link: https://youtu.be/T32r5e4TW-w

Made by: Archit Sangal (IMT2019012)