International Institute of Information Technology, Bangalore

Software Production Engineering

Scientific Calculator with DevOps

Submitted by,
Vishwajeet Deulkar, MT2020154
April 17 2022

Under the Guidance of

Prof. B. Thangaraju

Teaching Assistant:

Neha Kothari

Contents

1. Introduction

What is DevOps?

DevOps enables formerly siloed roles—development, IT operations, quality engineering, and security—to coordinate and collaborate to produce better, more reliable products. By adopting a DevOps culture along with DevOps practices and tools, teams gain the ability to better respond to customer needs, increase confidence in the applications they build, and achieve business goals faster.

Why should one use DevOps?

Teams that adopt DevOps culture, practices, and tools become high-performing, building better products faster for greater customer satisfaction. This improved collaboration and productivity is also integral to achieving business goals like these:

I. Accelerating time to market
II. Adapting to the market and competition
III. Maintaining system stability and reliability
IV. Improving the meantime to recovery

In this project, a Scientific Calculator web application has been created and DevOps tools are used to automate the process. Currently, the calculator has the following menu-driven operations.

1. Natural Logarithm (base e)
2. Square root function
3. Factorial function
4. Power function

The CI/CD pipeline is built using the following tools:

1. Development tools
    a. OS: Ubuntu 20.04.4 LTS and Ubuntu 18.04 server for deployment.
    b. IntelliJ IDEA
    c. Language: JAVA, HTML, CSS, JS
    d. Java Environment: OpenJDK version “11.0.14.1”
    e. Build: Apache Maven (3.6.3)
    f. Framework: Spring Boot
    g. SCM: Git, GitHub
2. Testing: JUnit
3. Integration: Apache Maven, Jenkins
4. Delivery: Docker, Docker Hub, Jenkins
5. Deployment: Docker, Ansible, Jenkins
6. Monitoring: ELK stack

The Scientific calculator source code can be found here:
GitHub Link: Scientific-Calculator-webapp-usingDevOps
DockerHubLink:
https://hub.docker.com/repository/docker/vishwajeet1321/scientific-calculator-webapp-devops

2. Source Code and Output

The source code is written in java and a menu driven program will be run for users. The code contains HTML, CSS, JS templates for frontend and spring boot REST API controllers for backend calls. The code has a Calculator.java class in which simple functions are written for different functionalities. For unit testing, a test class(CalculatorTest.java) is written. All dependencies are in pom.xml.

2.1 Project setup in IntelliJ

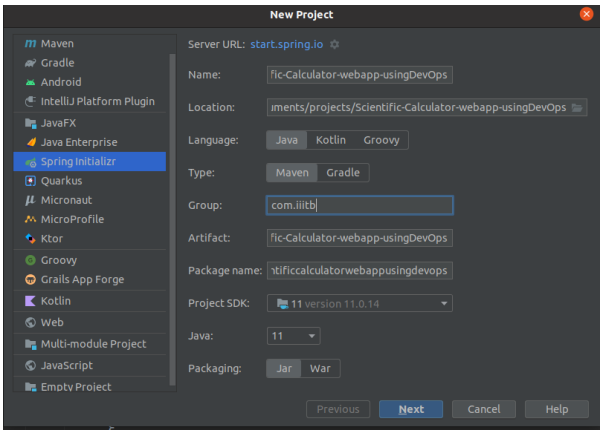


Figure 1 : IntelliJ project create tab

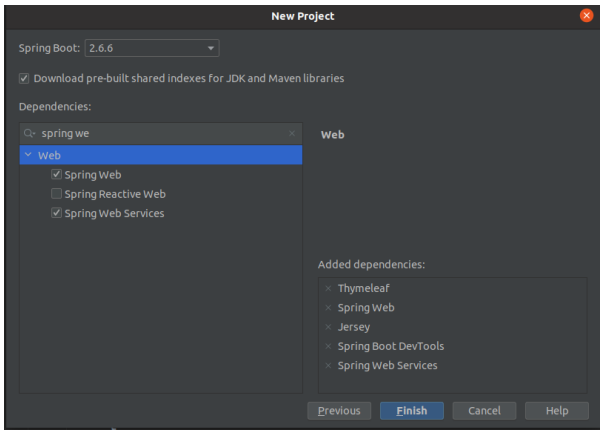


Figure 2 : Adding Dependencies

2.2 Calculator class spring boot API controller code

Sample function has been mentioned below, for all other functions APIs implemented similarly.

```
package com.iiitb.scientificcalculatorwebappusingdevops;

@RestController
public class Calculator {
   private static final Logger logger = LogManager.getLogger("Calculator");
   @RequestMapping(value = "/sqrt",method = RequestMethod.POST)
   public double sqrt(@RequestBody Map<String,Object> payload){
      double res;
      double input1 = Double.parseDouble((String) payload.get("input1"));
      res = Math.sqrt(input1);
      logger.info("SQUARE_ROOT - Input:" + input1 + " - Output:" + res);
      return res  ;
   } }
```

2.3 CalculatorTest class code

Sample function has been mentioned below, for all other unit test functions are implemented similarly.

```
package com.iiitb.scientificcalculatorwebappusingdevops;

public class CalculatorTest {
   private static final double DELTA = 1e-15;
   Calculator calculator = new Calculator();

   @Test
   public void squareRootTruePositive(){
      Map<String,Object> payload = new HashMap();
      payload.put("input1",(Object) new String("36"));

      Map<String,Object> payload2 = new HashMap();
      payload2.put("input1",(Object) new String( "12.25"));

      assertEquals("Squaring a number for True Positive", 6, calculator.sqrt(payload), DELTA);
      assertEquals("Squaring a number for True Positive", 3.5, calculator.sqrt(payload2), DELTA);

   }

   @Test
   public void squareRootFalsePositive(){
      Map<String,Object> payload = new HashMap();
      payload.put("input1",(Object) new String("36"));

      Map<String,Object> payload2 = new HashMap();
      payload2.put("input1",(Object) new String( "12.25"));
      assertNotEquals("Squaring a number for False Positive", 10, calculator.sqrt(payload),
DELTA);
      assertNotEquals("Squaring a number for False Positive", 2.5, calculator.sqrt(payload2),
DELTA);
   }
}
```

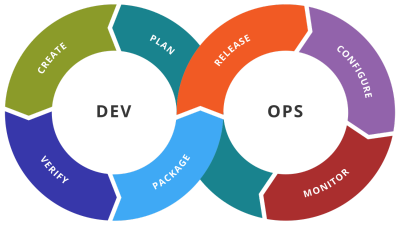
2.4 JavaScript file API calls to mapped functions

Sample function has been mentioned below, for all other API call functions are implemented similarly.

```
//var base = 'http://localhost:8085'
var base = "http://172.16.134.244:8085"
 async function sqrt() {
       const Input1 = document.getElementById("num1").value;
       fetch(base+'/sqrt', {
          method: 'POST',
          headers: {
             'Content-Type': 'application/json;charset=utf-8'
          },
          body: JSON.stringify({
             input1:Input1,

          })
       }).then(response => response.json())
          .then((data) => {
             document.getElementById("output").value = data;
          });
    }
```

3. SDLC and Source Code Management

3.1 SDLC workflow in the project

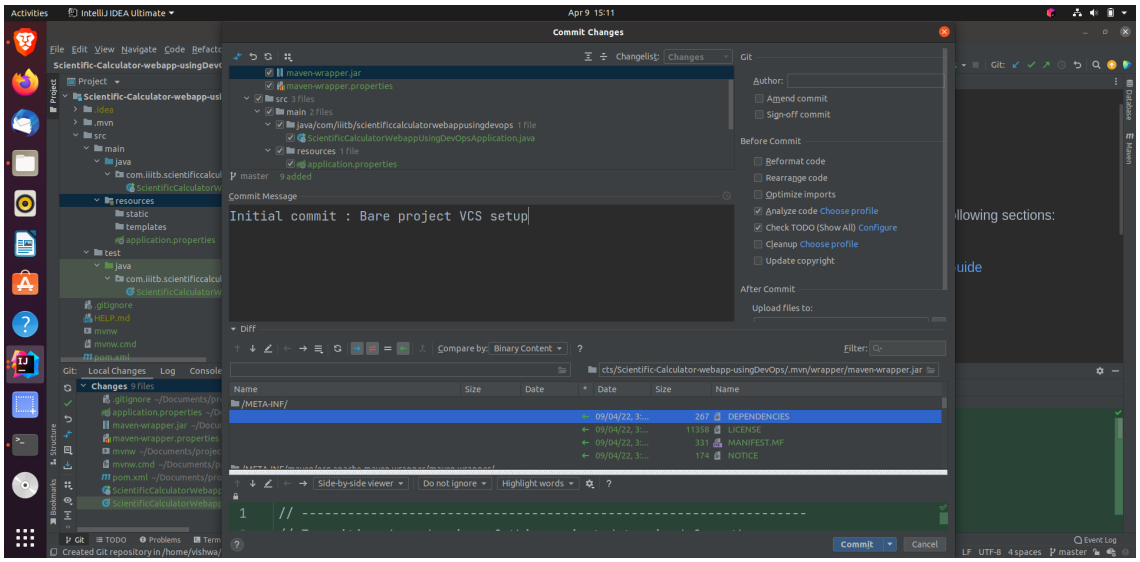


In the Software Development Life Cycle there are many stages. Using DevOps we can automate stages in a pipeline. For example in our project for code SCM we are using Git and GitHub, Maven & Junit have been used for build and test respectively. DockerHub is used at the delivery stage, Deployment is done using Ansible, in our project we are deploying Scientific Calculator inside a Ubuntu server within a docker container. The ELK stack is used for monitoring and visualizing logs.

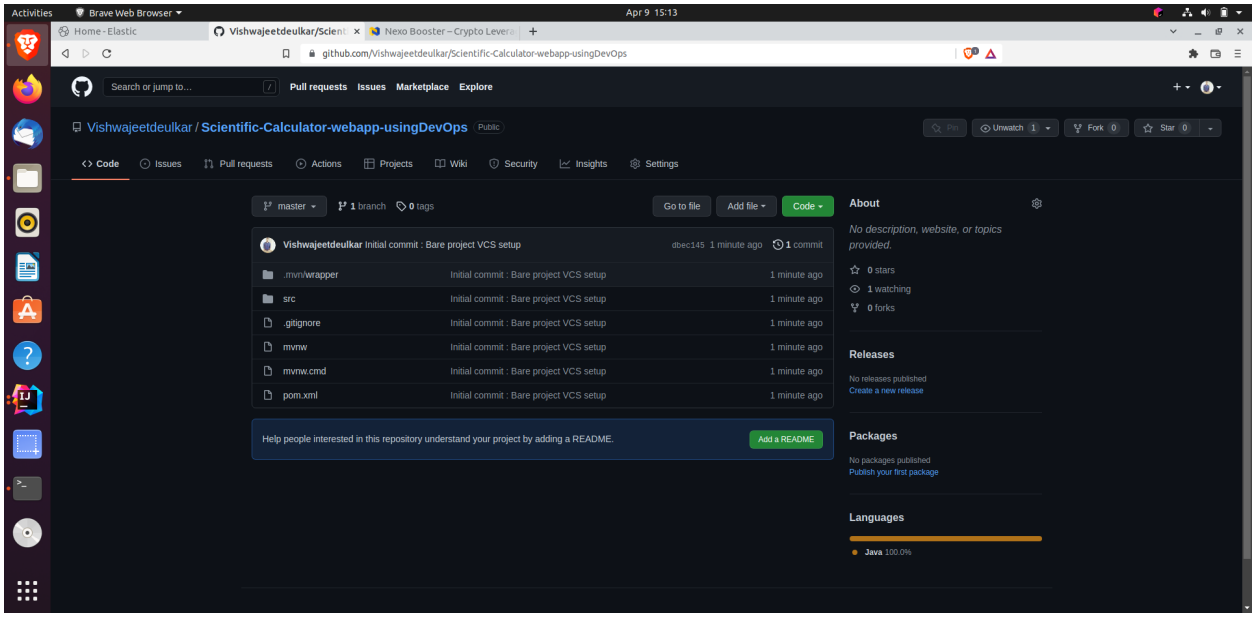
3.2 Source Code Management (SCM)

Source code management is the practice of tracking modifications to source code. Keeping a running history of the changes made to a codebase helps programmers, developers and testers ensure that they're always working with accurate and up-to-date code and helps resolve conflicts when merging code from multiple sources. Here, we used Git as a SCM and GitHub for online repository hosting.

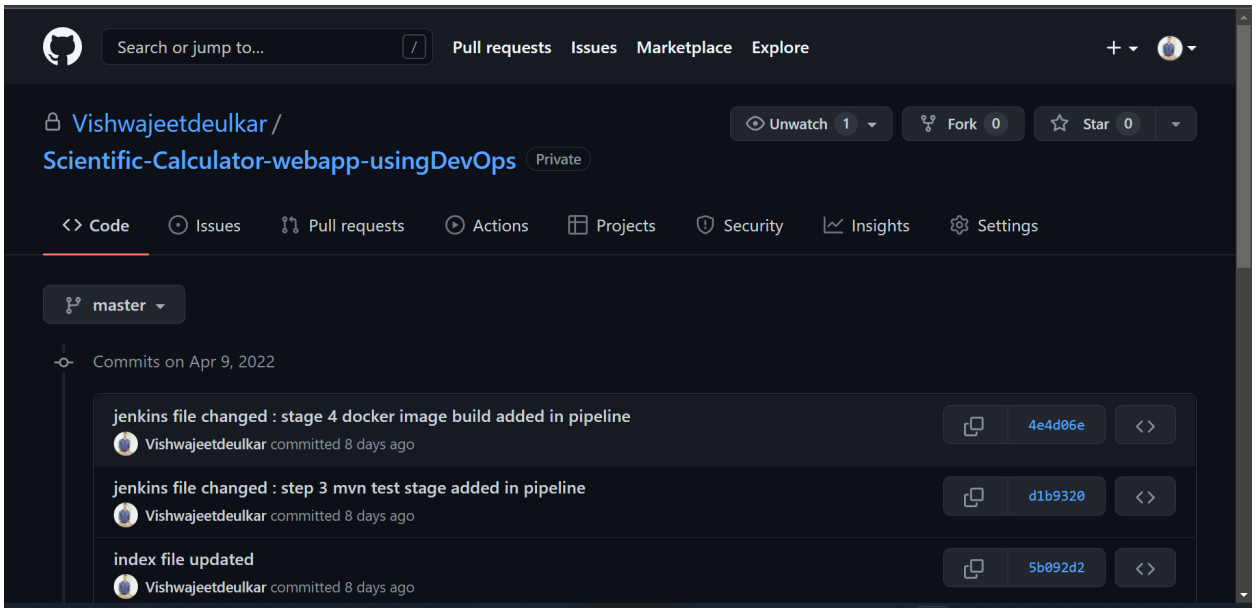
3.3 Sharing project on Git



After creating a project in the IDE we used the enable VCS option to integrate with Git. After the first commit and push, we share a repository on Github. **Git -> Github -> Share Project on Github.**



All source commits can be seen here:
[Scientific calculator GitHub project commits]



4. Build and Test

We are using Apache Maven for building the project and managing dependencies. Maven is used for adding jars and other dependencies to the project. Maven will finally create the SNAPSHOT jar file of the project which is compiled with java classes and other inbuilt classes on which functions depend. All dependencies are written in the pom.xml file. Project's pom.xml file is shown below:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>
```

As for monitoring we have to set patterns for logs and filter out some server logs, hence we need to use Apache log4j2. So to avoid log file creation errors and hide the basic logging functionality

provided by Spring Boot we have excluded spring-boot-starter-logging dependency. and added dependency for log4j2.

To build the project without running test cases:

```
$ mvn -B -DskipTests clean package
```

Here we will remove the target folder and compile & package the source code using a distributable format such as jar.

To test the project we have used Junit, a Java unit testing framework.

```
$ mvn test
```

Using this command we will test compiled source code. Testing need not be required to create a packaged jar.

5. Continuous Integration/Continuous Deployment Pipeline (CI/CD)

5.1 Continuous Integration (CI)

Continuous integration (CI) is the practice of automating the integration of code changes from multiple contributors into a single software project. It's a primary DevOps practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run. Automated tools are used to assert the new code's correctness before integration. Here, we are using Jenkins for CI, it will keep a check on the SCM system for any changes in code and detects the changes and as well as builds the code.

5.1.1 Jenkins Installation

To install Jenkins follow the step given below:

```
$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo
apt-key add -
$  sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt update
$ sudo apt install Jenkins
$ sudo systemctl enable --now jenkins
```

Jenkins will run on http://localhost:8080, it will use 8080 as a default port for accessing jenkins.

Basic Jenkins setup steps:

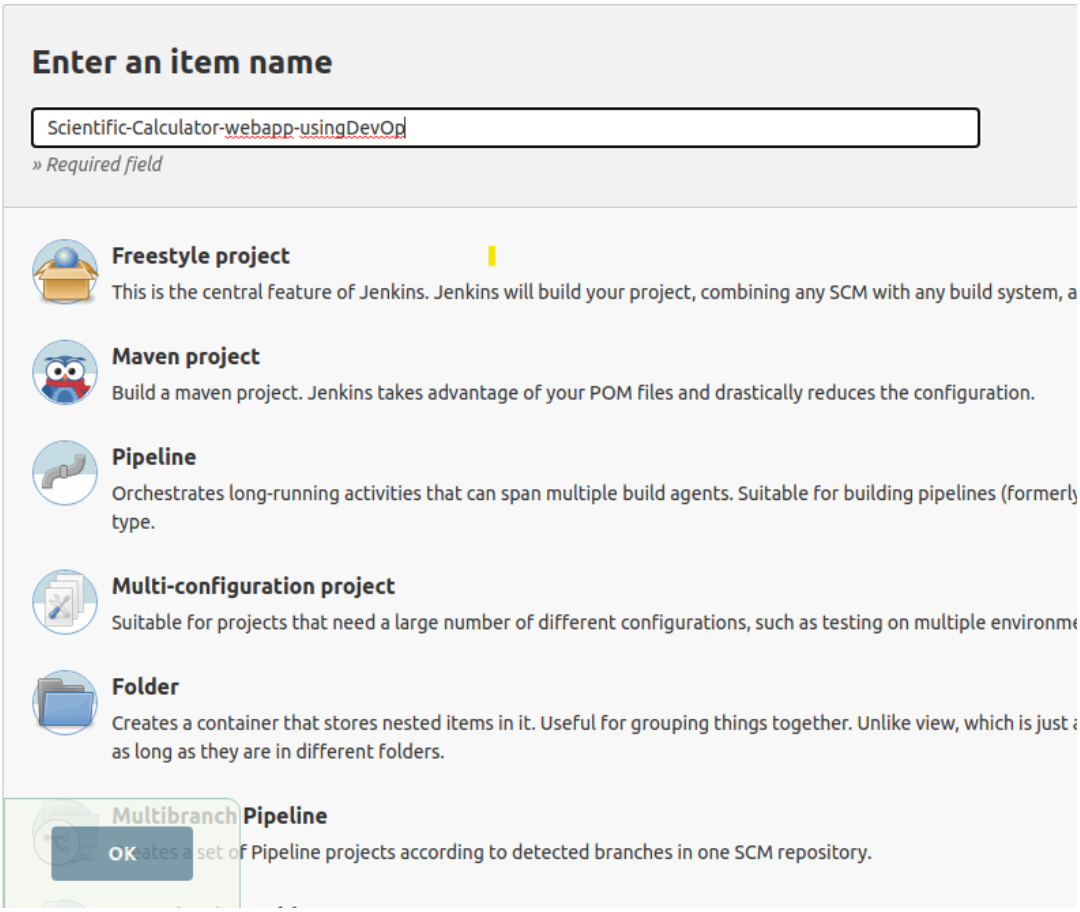
1. Create a admin user
2. Install plugins for Maven, Junit, Docker, Ansible.**Manage Jenkins -> Manage Plugins**
3. As we are accessing cloud repositories such as GitHub and DockerHub we need to add their credentials to access the files.**Manage Jenkins -> Manage Credentials -> Add Credentials.**

5.1.2 Jenkins Pipeline

Follow the steps to create a Jenkins pipeline:

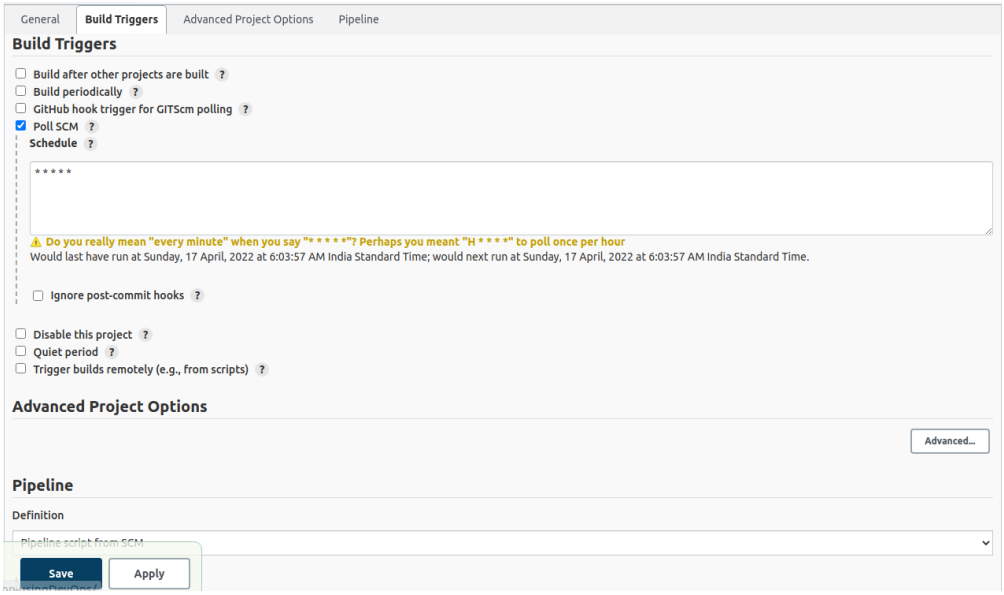
1. Create a new Jenkins pipeline as shown below:

- Dashboard -> New Item
- Give name to project
- Select pipeline project, as a project type

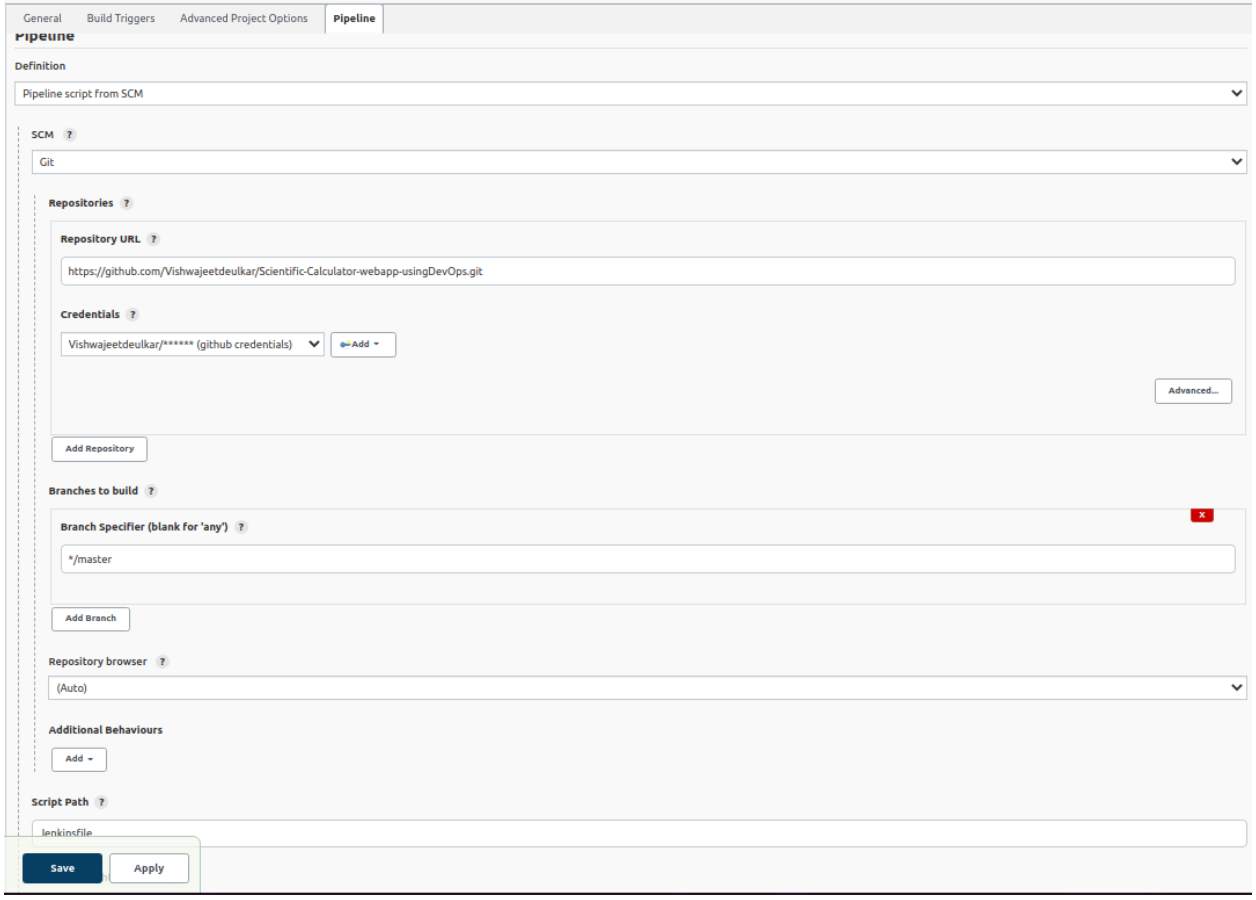


2. Add project Description and give GitHub repository URL:

3. Set a build trigger:



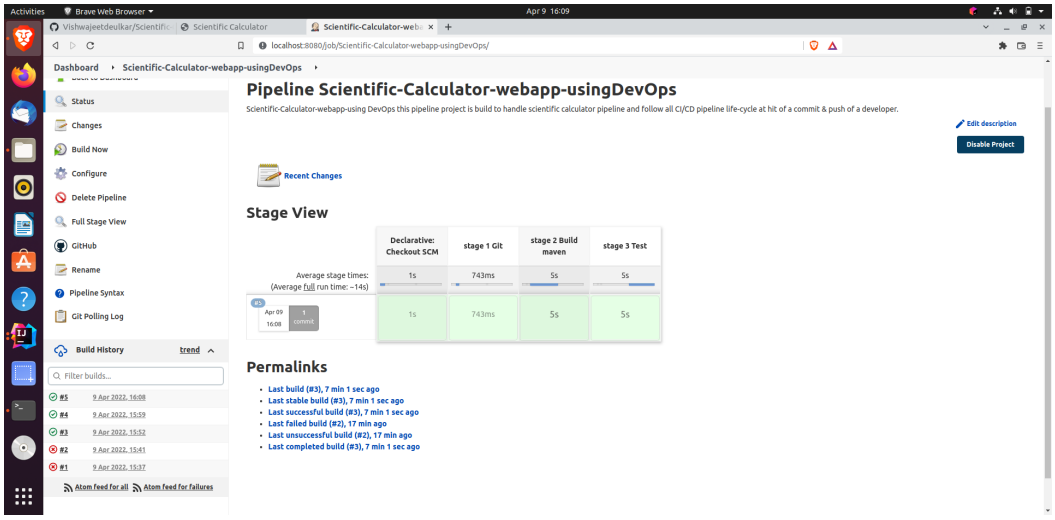
3. Add pipeline script in SCM section:



4. Add pipeline script in SCM Jenkins file:

```
pipeline {
    agent any
    stages {
        stage('stage 1 Git') {
            steps {                 // Get some code from a GitHub repository
git url: 'https://github.com/Vishwajeetdeulkar/Scientific-Calculator-webapp-usingDevOps.git/',
branch: 'master',
                credentialsId: 'gitcred'
            }
        }
        stage('stage 2 Build maven') {
            steps {
                // Compile and package code without unit testing.
                sh "mvn -B -DskipTests clean package"
            }
        }
        stage('stage 3 Test') {
            steps {
                // Unit testing on compiled source code.
                sh "mvn test"
            }
        }
```

5. Save the configuration & click on the build now option.



Until now, SCM pulling, building, testing was automated and a jar file was created in the target folder. This is continuous integration.

5.2 Continuous Delivery

Continuous Delivery (CD) is the process to build, test, configure, and deploy from a build to a production environment. Multiple testing or staging environments create a Release Pipeline to automate the creation of infrastructure and deployment of a new build.

In our project CD is achieved with help of the delivery stage of the pipeline which includes building docker images with help of a Dockerfile and the images are delivered to Dockerhub. We use Docker and Jenkins for continuous delivery.

5.2.1 Docker Installation

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. It is a tool used to create, deploy and run applications in lightweight fashion. A container of an image is an executable package of applications/software which is lightweight and includes all libraries, system tools , code and runtime.
Follow the steps to set your system for docker:

1. To install docker(in ubuntu)

```
$ apt-get install docker.io
```

2. Give permission to Jenkins to run docker commands

```
$ sudo usermod -aG docker jenkins
```

5.2.2 Docker Image Build

After compiling, testing and building code Maven gives output a JAR file with all required dependencies in the target folder. A Dockerfile is a text document a user could call on the command line to assemble an image. Pipeline script has been added to the Jenkins file for building a docker image.

```
/* The environment specifies the credentials required to push my image to dockerhub */
environment {
   registry = "vishwajeet1321/scientific-calculator-webapp-devops"
   registryCredential = 'docker-cred'
   dockerImage = ''
}
   stage('stage 4 Building docker image') {
      steps{
         script {
            dockerImage = docker.build registry + ":latest"  }          }          }
```

Dockerfile:

```
FROM openjdk:11
EXPOSE 8085
ADD target/Scientific-Calculator-webapp-usingDevOps-0.0.1-SNAPSHOT.jar
scicalculator.jar
ENTRYPOINT ["java","-jar","scicalculator.jar"]
```

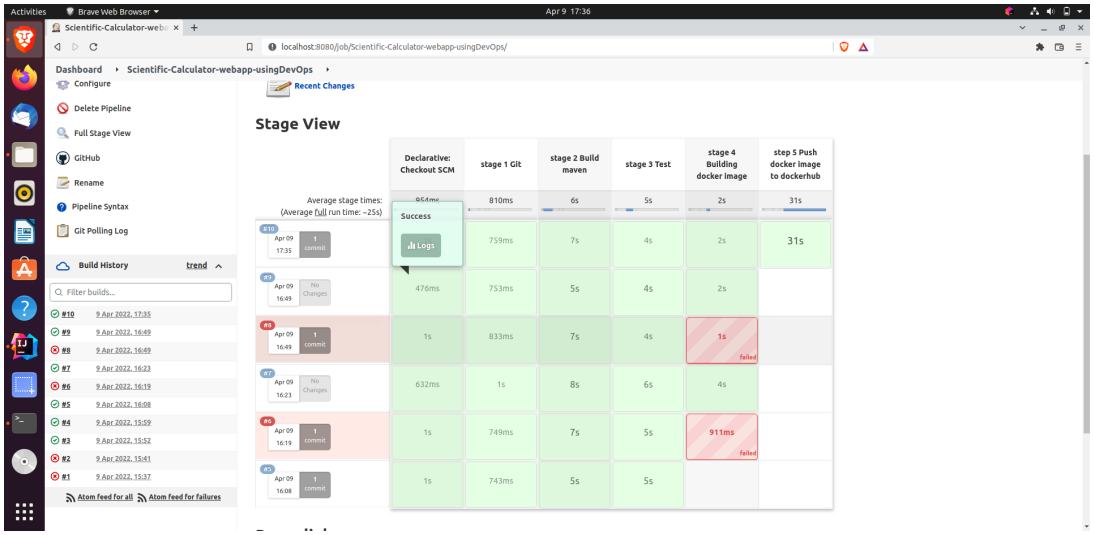
Layer 1: base image is openJDK:1 as our application is built using Maven Java.
Layer 2: EXPOSE is used to inform Docker that the container listens on the specified network ports at runtime.
Layer 3: ADD used to copy files/directories into a Docker image.
Layer 4: EXPOSE is used to set executables that will always run when the container is initiated.

5.2.3 Publish a Docker Image

After building, image vishwajeet1321/scientific-calculator-web-app-devops it will be present at the local machine on which Jenkins is running. Dockerhub is a hosted repository service provided by Docker for finding and sharing container images. To publish this docker image on docker hub, first sign up on docker hub. Add the following in the script after the image build stage.

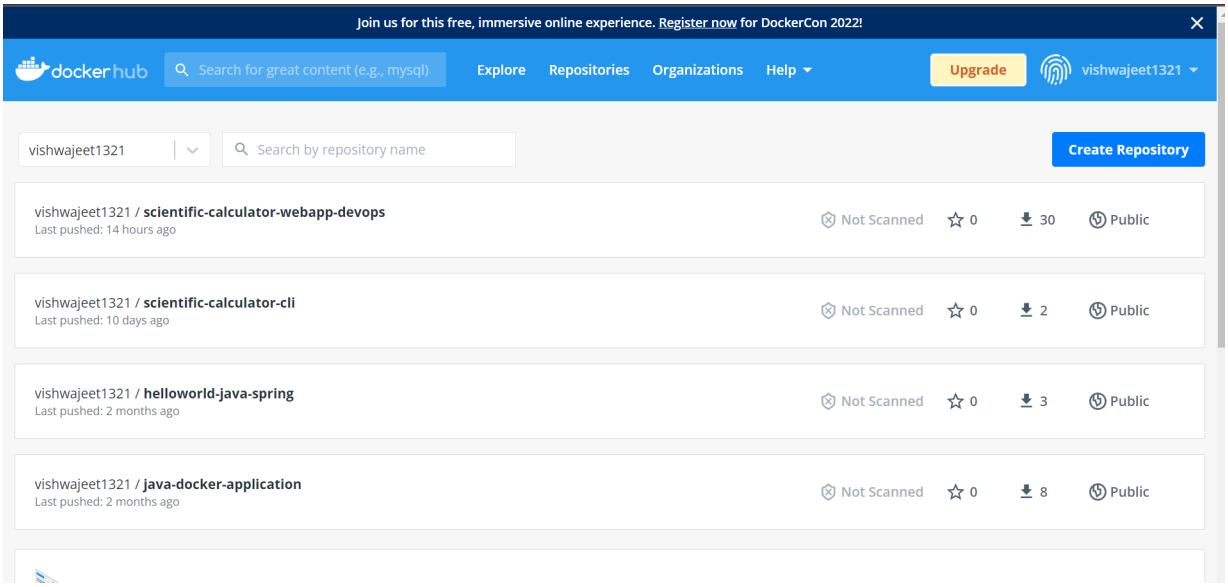
```
    stage('stage 5 Push docker image to dockerhub') {
            steps{
               script {
                  docker.withRegistry( '', registryCredential ) {
                  dockerImage.push()
                  }      }      }      }
```

Click build now to check everything is working fine until continuous delivery.



DockerHub:

Image is published on DockerHub.



***While building a docker image we faced an issue as permission denied for /var/run/docker.sock uses the following command to give permission and resolve the issue.

```
$ sudo chmod 666 /var/run/docker.sock
```

5.3 Continuous Deployment (CD)

In Continuous Deployment we take validated features from the staging environment and deploy them into production. As pipelines are triggered automatically we don't need to pause for development releases. Releases are less risky and easier to fix in case of problems as you deploy small batches of changes. Customers see a continuous stream of improvements, and quality increases every day, instead of every month, quarter or year.

After the delivery stage in our project docker image is created and published on DockerHub, We are using Ansible to fetch the image and deploy the container in the Ubuntu server, Ansible playbook is invoked by Jenkins pipeline script.

5.3.1 Ansible Installation

Ansible is an open source IT configuration management, deployment, and orchestration tool. It encourages DevOps teams to define their infrastructure as a code in a simple and declarative manner.

Benefits of using Ansible:

1. Ansible is agentless
2. Ansible is written in python

3. Ansible is Easy to learn
4. Deploy Infrastructure in record time

To install Ansible, follow the steps:

1. First we need to make a connection with the server on which we will be deploying ansible. For that we will be using SSH to connect to the remote host.
2. Install Open SSH server

```
$ sudo apt install openssh-server
```

3. Generate RSA key pair

```
$ ssh-keygen -t rsa
```

4. Copy ssh key and login to remote user

```
$ ssh-copy-id REMOTE_HOSTNAME@REMOTE_IP_ADDRESS
$ ssh REMOTE_HOSTNAME@REMOTE_IP_ADDRESS
```

5. Install Ansible

```
$ sudo apt update
$ sudo apt install ansible
```

6. Check the ansible version.

```
$ ansible --version
```

```
vishwa@vishwa-VirtualBox:~$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/vishwa/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Mar 15 2022, 12:22:08) [GCC 9.4.0]
vishwa@vishwa-VirtualBox:~$
```

5.3.2 Ansible integration with jenkins

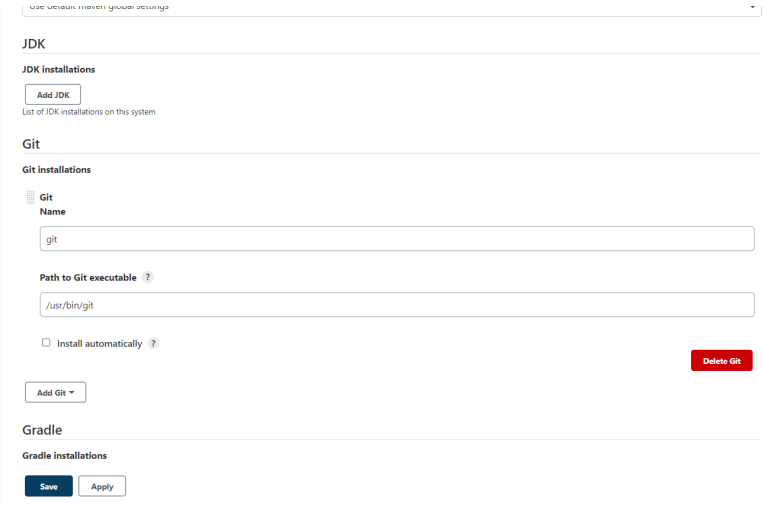
1. Start Jenkins on Control Node

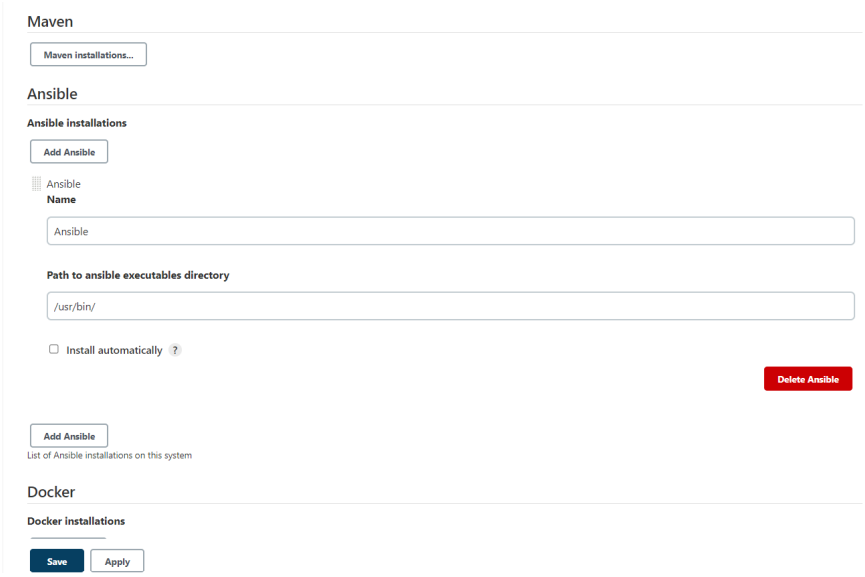
```
$ sudo sytemctl start Jenkins
```

2. Installing Ansible plugin on Jenkins
3. Adding ssh keys to Jenkins

```
$ sudo su jenkins
$ ssh-keygen -t rsa
$ ssh-copy-id REMOTEUSER@<REMOTE IP ADDRESS>
```

4. Setting up Git and Ansible on Jenkins





5.3.3 Ansible in jenkins pipeline

Below is the pipeline script used for the Asible deployment step:

```
        stage('Stage 6 Ansible image deploy'){
                steps{    ansiblePlaybook becomeUser: null, colorized: true,
disableHostKeyChecking: true, installation: 'Ansible', inventory: 'ansible-deploy/inventory',
playbook: 'ansible-deploy/deploy-image.yml', sudoUser: null   }    }
```

Inventory file used to give MetaData about deployment server:

```
[ubuntu18]
172.16.134.244 ansible_user=vishwa
```

Here, Ubuntu18 is the server name.

IP address of server and username.

Deployment script for image deploy in remote server:

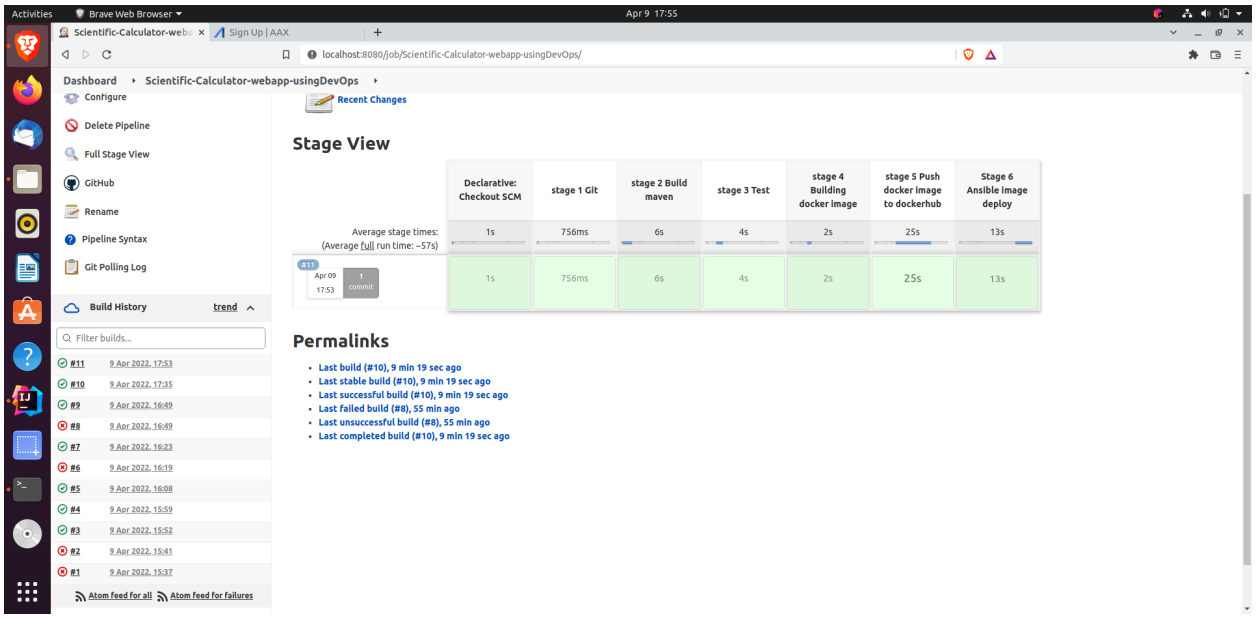
```
---
- name: Pull docker image of app
  hosts: all
  tasks:
    - name: remove stopped container
      shell: docker rm -vf $(docker ps -aq) || true
    - name: remove docker images
      shell: docker rmi -f $(docker images -aq) || true

    - name: Pull app devops image
      docker_image:
        name: vishwajeet1321/scientific-calculator-webapp-devops
        source: pull
    - name: create web container
      shell: docker run -d -p 8085:8085 --name calculator
vishwajeet1321/scientific-calculator-webapp-devops:latest
```

In this YML script tasks are mentioned to perform by a playbook:

1. Remove the stop containers
2. Remove the docker images
3. Pull the image from DockerHub
4. Create container and deploy image container on Ubuntu server

*** Here, "|| true" is provided to carry the task flow even if there are no images or containers to remove.

5.3.4 Stage view: Ansible in jenkins pipeline



It is a final pipeline view, which includes:

1. Push the code on Github repository.
2. Jenkins detects the change and builds the code.
3. Jenkins tests the compiled code.
4. Jenkins triggers the job of building docker images.
5. Push that docker image on Docker Hub
6. Jenkins triggers the Ansible playbook job to deploy a docker image on a container on a Ubuntu server.

*** While deploying on the server we got an error regarding the python3-docker not being installed. After hitting the below command the issue was resolved.

```
$ sudo apt install python3-docker
```

6. Monitoring (ELK Stack)

The proactive method of observing systems with goals of preventing downtime and outages, is monitoring. It involves measuring current behavior against predetermined baselines. Some of the commonly observed devices are CPU usage, storage capacities, request hits i.e network traffic, etc. by which we can find the root cause of the failure. Here, We used ELK stack monitoring.
Elasticsearch is a modern analytics and search engine which is based on Apache Lucene. Using this we can draw meaningful information from index logs created by applications. We can use elastic to manage, store and search data which can come handy in Logs, Metrics, Application monitoring, Endpoint security, Search backend.
We can use ELK stack locally or kibana & elasticsearch can be run on cloud and we can send logs using logstash installed on the host machine. Here, we have used cloud approaches as for installing ELK stack RAM requirements were high.

6.1 Using elastic cloud

First we need to create an account on Elasticsearch cloud.

1. Go to https://www.elastic.co/ URL and signup.
2. Create the first deployment.
3. Go with default configuration.
4. Save username & password after creation of deployment as we will need them hereafter and they will ot be shown agian.

6.2 Parse log file using logstash

We have used Apache log4j2 for log creation, when the application runs on any host respective calculator.log file is created. So after Deployment of a docker image to a container inside an Ubuntu server we need to copy the file from container to server and then server to host for monitoring and feeding to Elastic and Kibana clouds.

1. Get the docker container id

```
$ docker ps -a
```

2. Copy file from docker container to server

```
$ docker cp <container_id>:<source_location>  <destination_location>
```

3. Copy file from server to host machine i.e. Ubuntu server to Ubuntu in my case

```
$ scp <remote_user@ip_address:source_location>  <destination_location>
```

4. To parse a log file via logstash use the following command or you can upload it manually.

```
$ bin/logstash -f /home/vishwa/documents/calculatorproject/calculatorlogger.conf
```

calculatorlogger.conf is a configuration file, it contains the grok pattern , timestamp, cloud link , username and password of users.

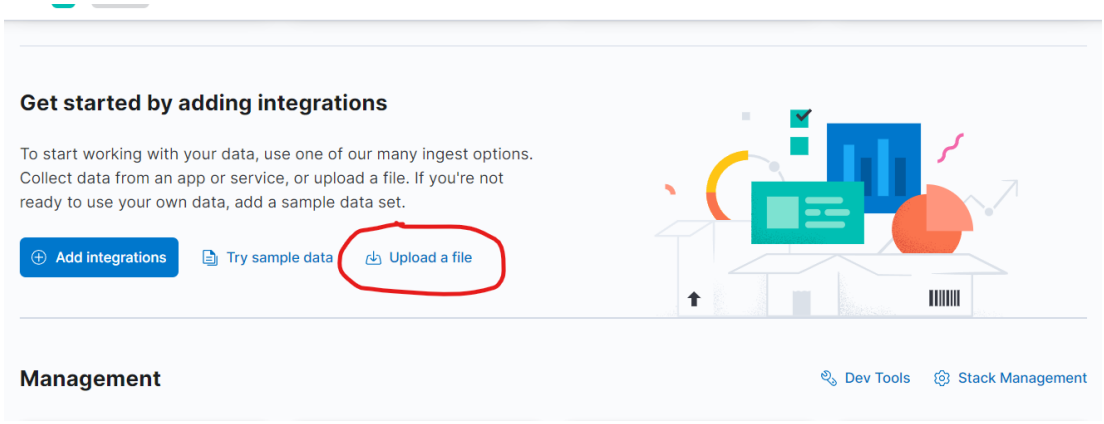
```
input {
 file {    path =>
"/home/vishwa/Documents/projects/Scientific-Calculator-webapp-usingDevOps/calculator.log"
   start_position => "beginning"
  }
}
filter {
  grok {
    match => [
      "message", "%{TIMESTAMP_ISO8601:timestamp} \[.*?\] \[%{LOGLEVEL:level}\]
%{GREEDYDATA:logger} %{GREEDYDATA:action} \- %{GREEDYDATA:input} \-
%{GREEDYDATA:output}"
    ]
  }
  date {
    match => ["timestamp", "dd/MMM/YYYY:HH:mm:ss XX"]
  }
 mutate {
    remove_field => [timestamp]
  }
}
output {
  elasticsearch {
    hosts => ["https://calculator-2b50da.kb.us-central1.gcp.cloud.es.io:9243/"]
    user => "elastic"
    password => "fqyML6kaJWU1VvU3Z5w0qHkM"
    index => "calculator_elastic"  }
  stdout {
```

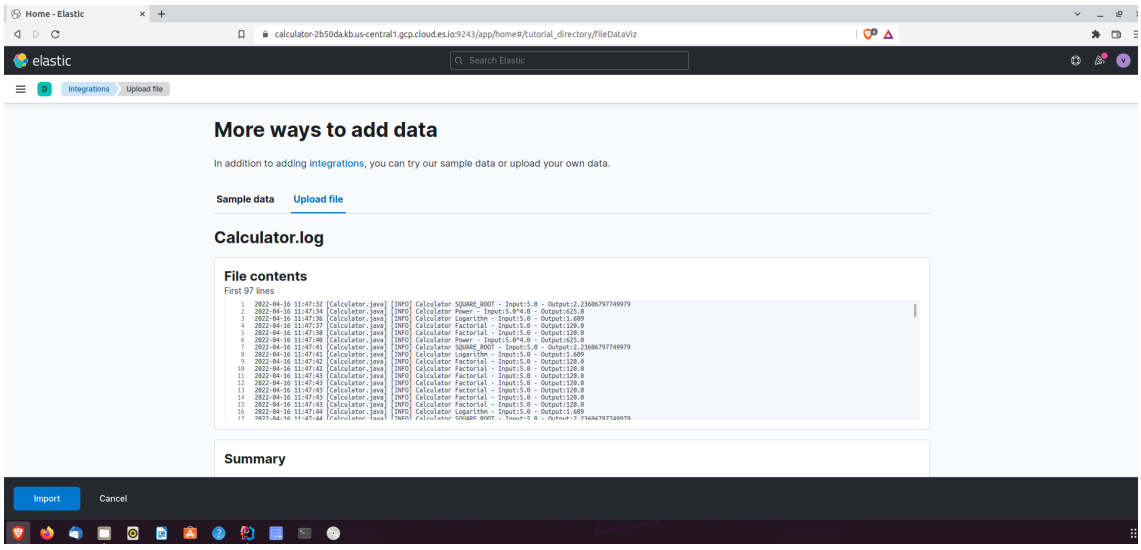
```
codec => rubydebug } }
```

6.3 Visualizing using Kibana

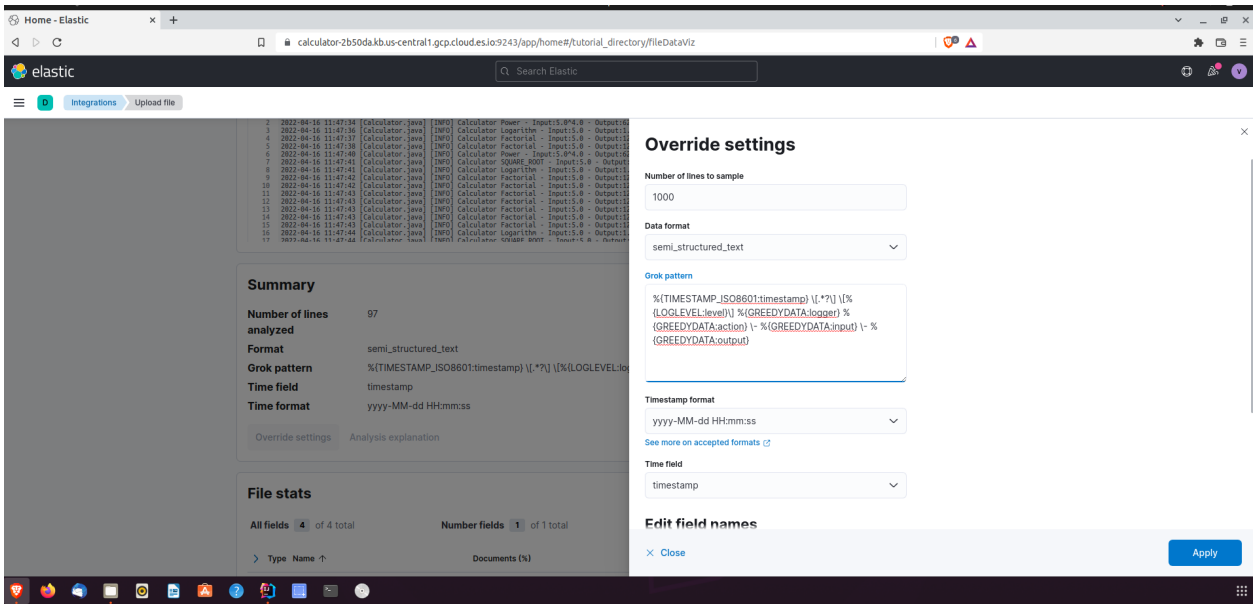
For visualizing logs on Kibana we are following the following steps:

1. On deployment, in the Home window, press on the upload data TAB.

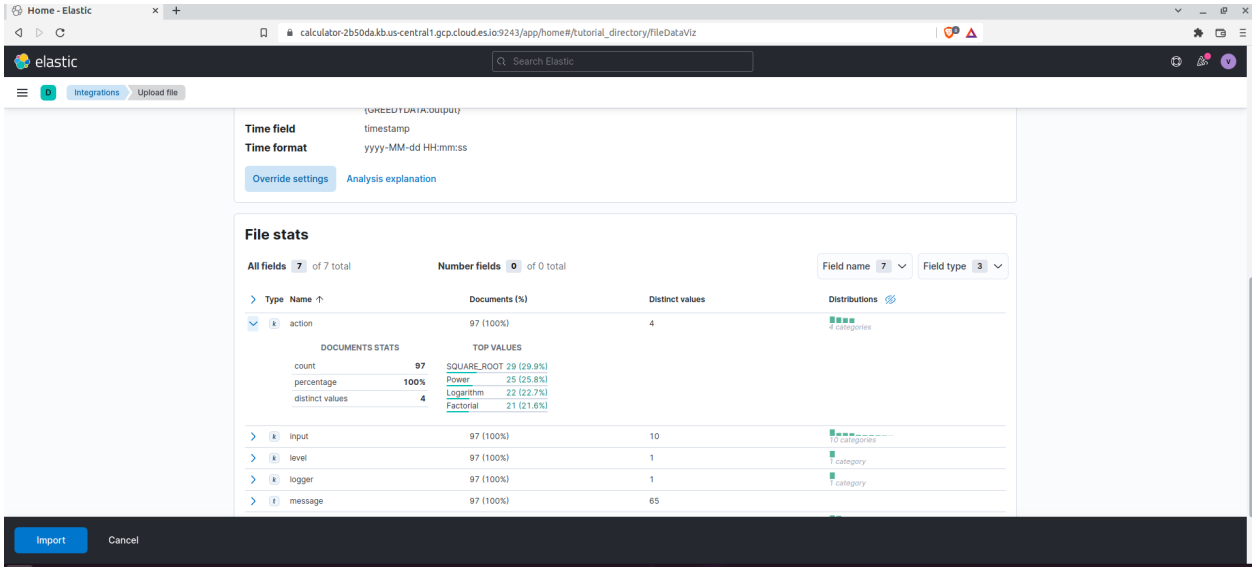




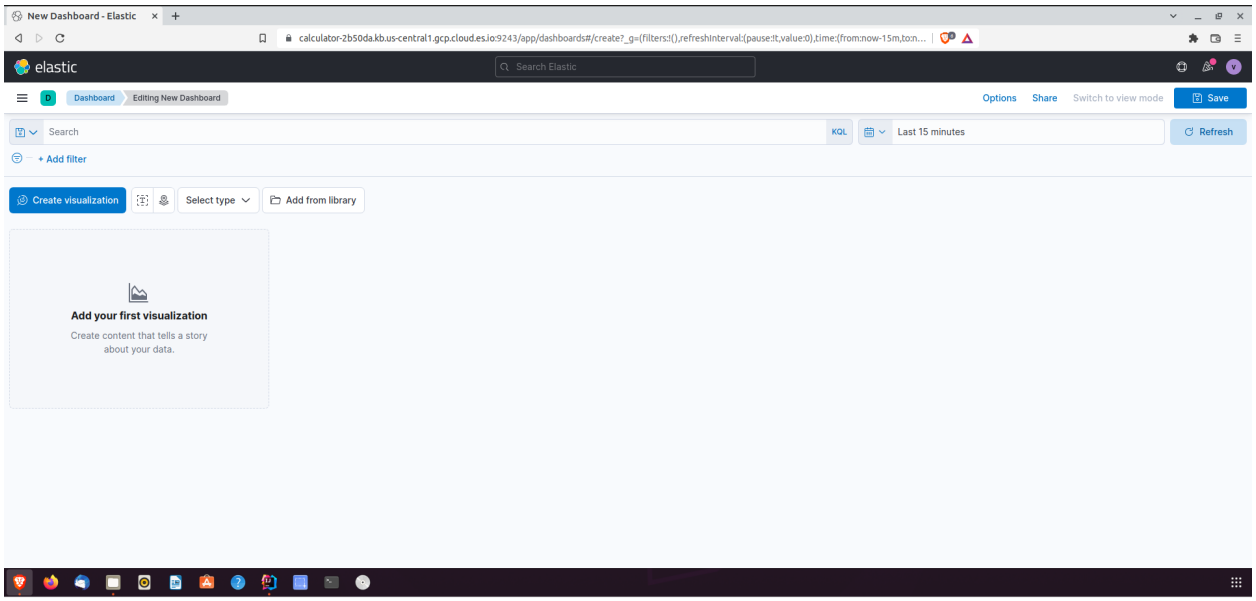
2. Go on override settings options and apply a custom grok pattern and timestamp.



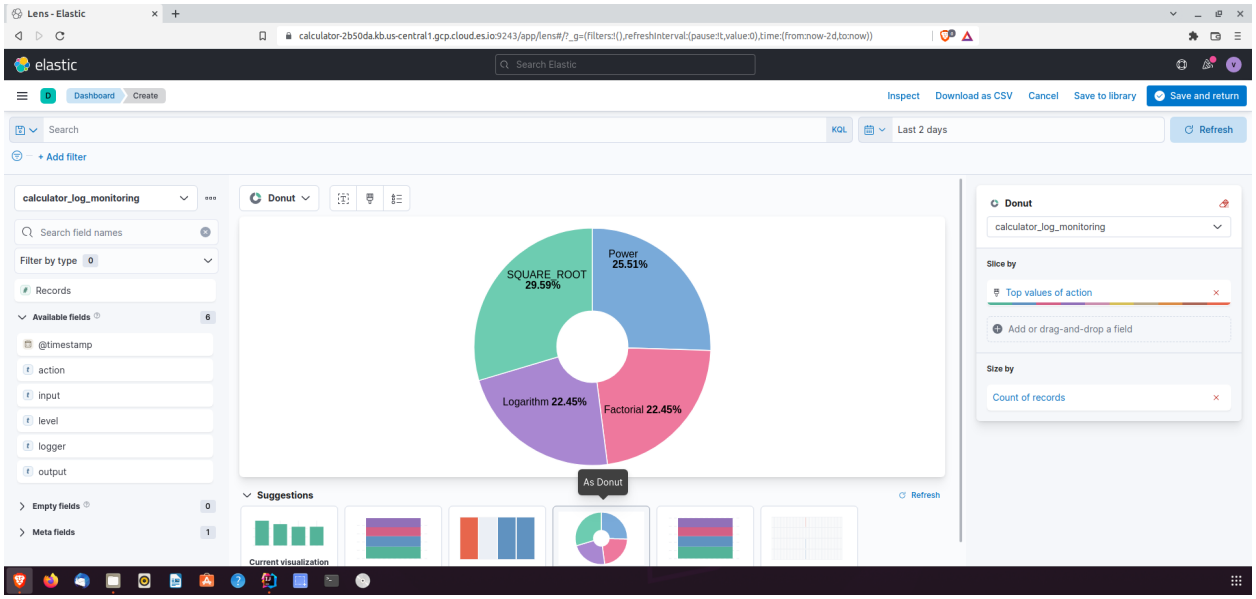
3. Action field will be created according to functions:



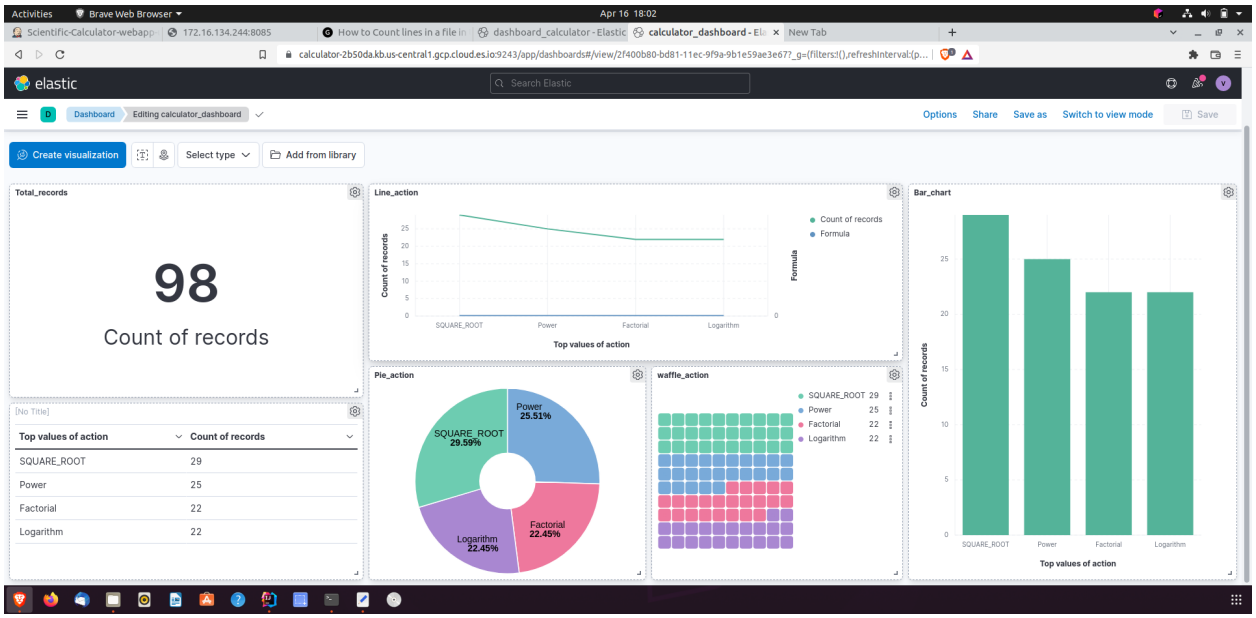
4. Go to the create dashboard optio and choose the time period of logs using calendar option.

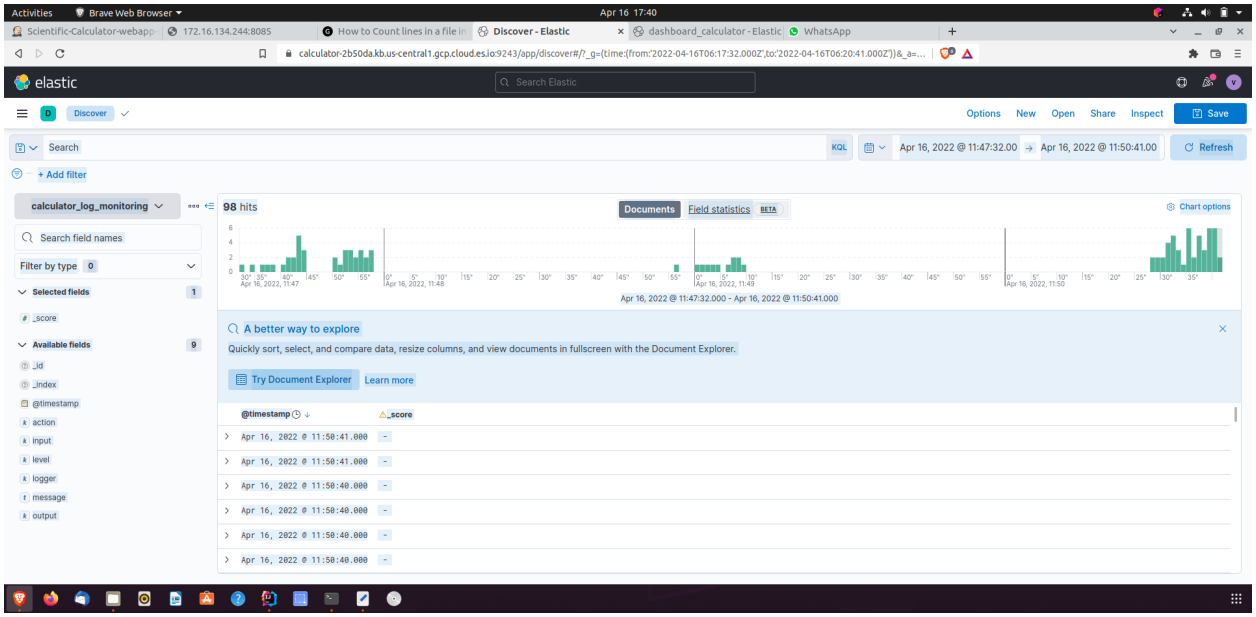


5. Create visualizations for dashboards.



6. To create a full dashboard, first create visualizations and save them by option save & return. After creating & adjusting visualizations, save to create a dashboard.





7. Challenges and errors faced

1) Permission error for /var/run/docker.run file

Solved by following command :

```
$ sudo chmod 666 /var/run/docker.sock
```

2) Log file not generated issue:

Resolved by excluding dependency on the default logging functionality of spring boot and then adding log4j2 dependency.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
        <exclusions>
          <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
          </exclusion>
        </exclusions>
      </dependency>

      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
      </dependency>
```

3) Error while creating new container after each deployment:

It is resolved by using below docker commands to stop and remove respective containers and images and the deploying new one. Note ; Here, true has been used after each command in case there is no image or container in the server at first time.

```
  - name: remove stopped container
    shell: docker rm -vf $(docker ps -aq) || true
  - name: remove docker images
    shell: docker rmi -f $(docker images -aq) || true
```

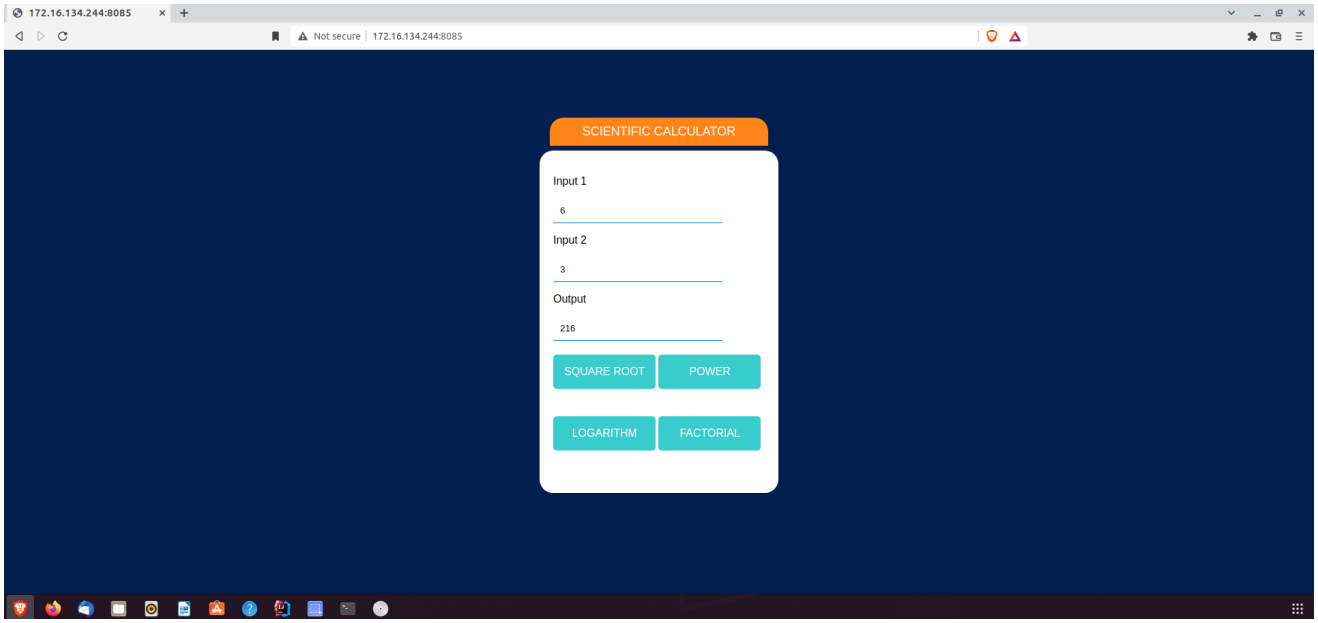
4) Installation not found of python-docker

To resolve this we install python-docker

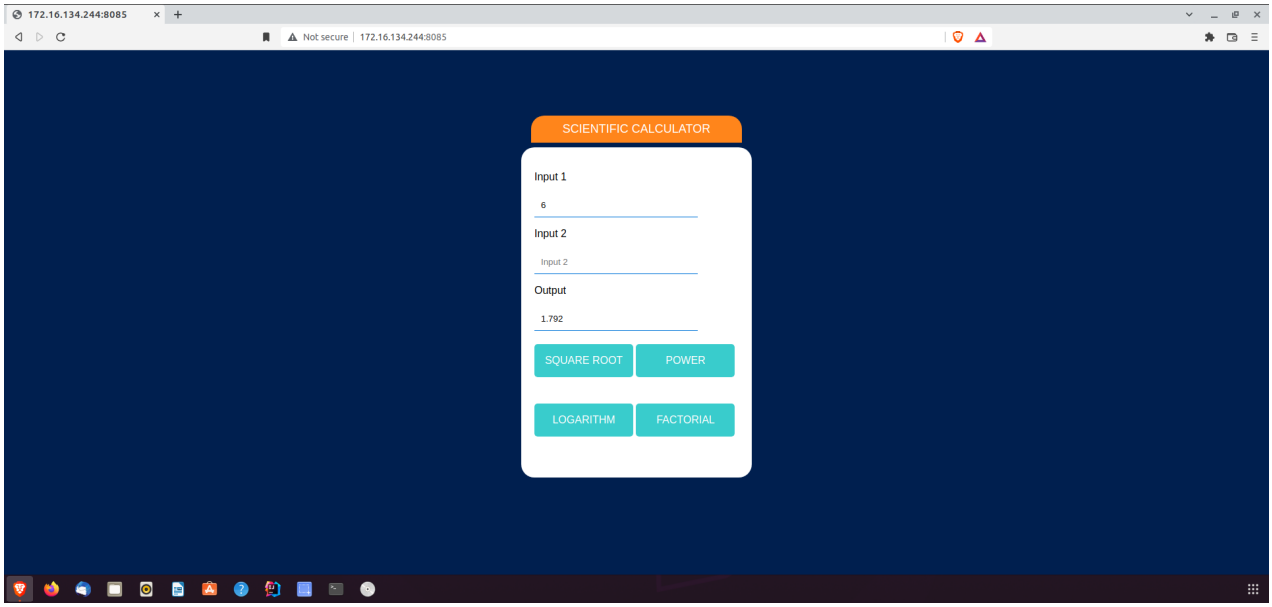
```
$ sudo apt install python3-docker
```

8. Scientific Calculator operations screenshots

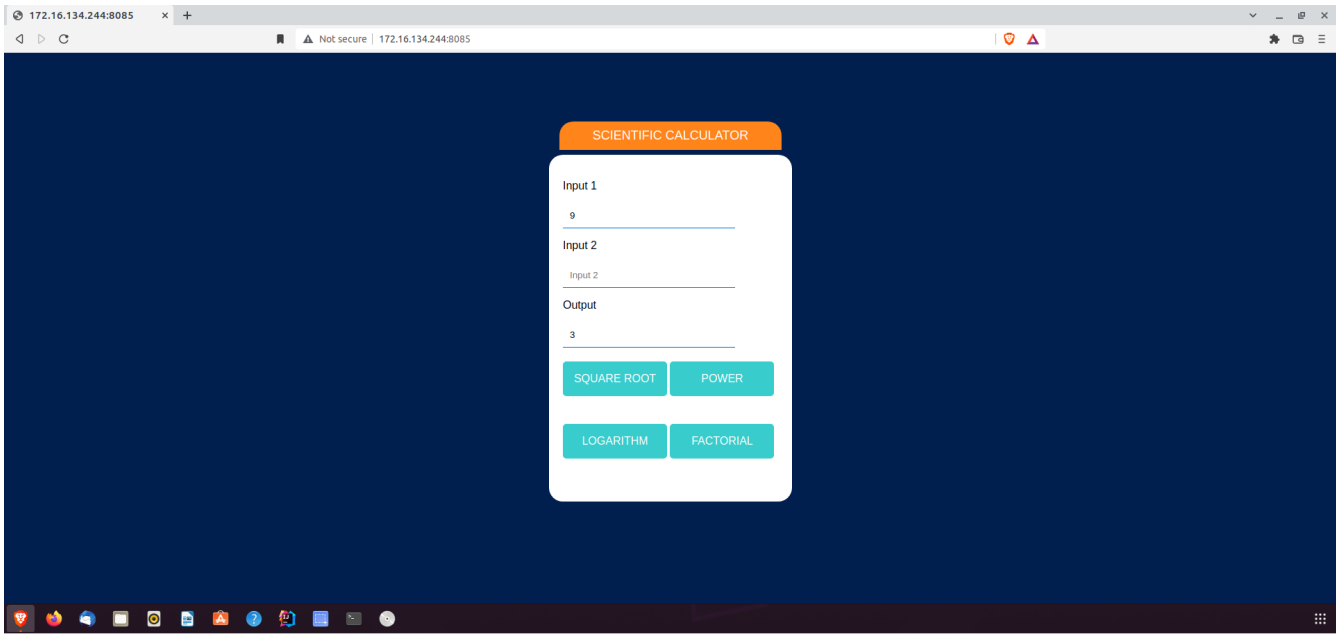
1. Power function



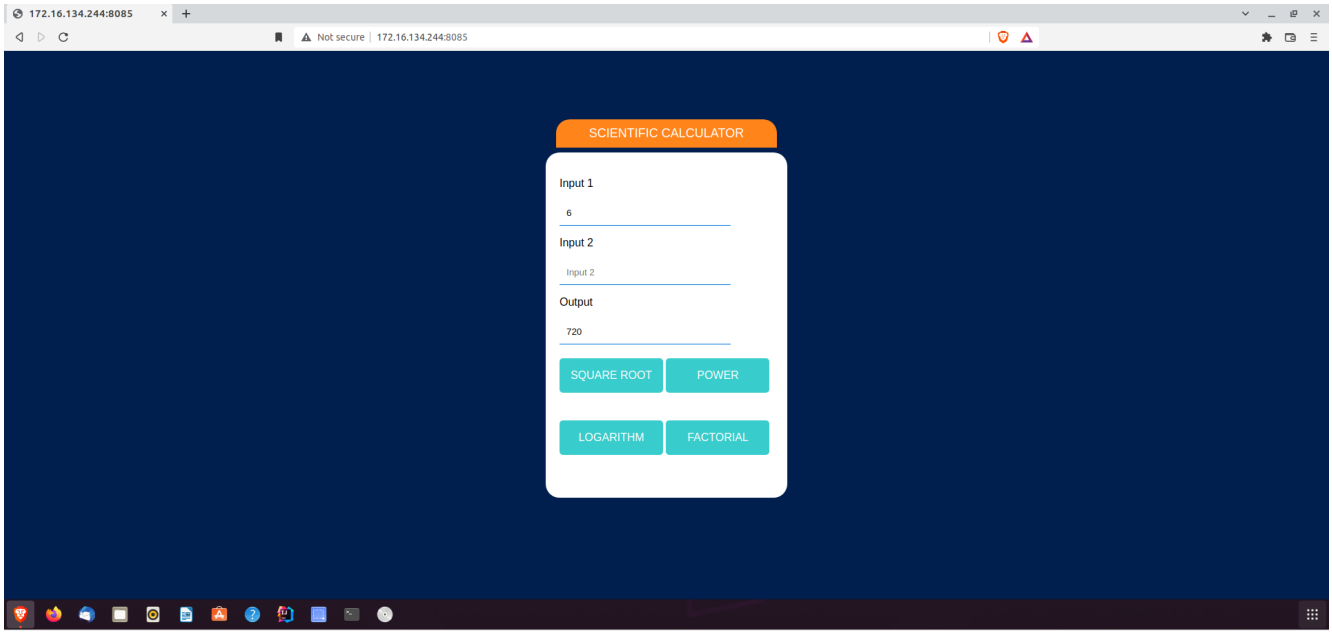
2. Logarithm function



3. Square root function



4. Factorial function



9. References

Following are references used to solve issues while developing a project:

I. https://www.digitalocean.com/community/questions/how-to-fix-docker-got-permission-denied-while-trying-to-connect-to-the-docker-daemon-socket
II. https://howtodoinjava.com/spring-boot2/logging/spring-boot-log4j2-config/
III. https://logging.apache.org/log4j/2.x/maven-artifacts.html
IV. https://stackoverflow.com/questions/12472645/how-do-i-schedule-jobs-in-jenkins
V. https://dockerlabs.collabnix.com/docker/cheatsheet/
VI. https://www.elastic.co/
VII. All Lab documents and course slides.