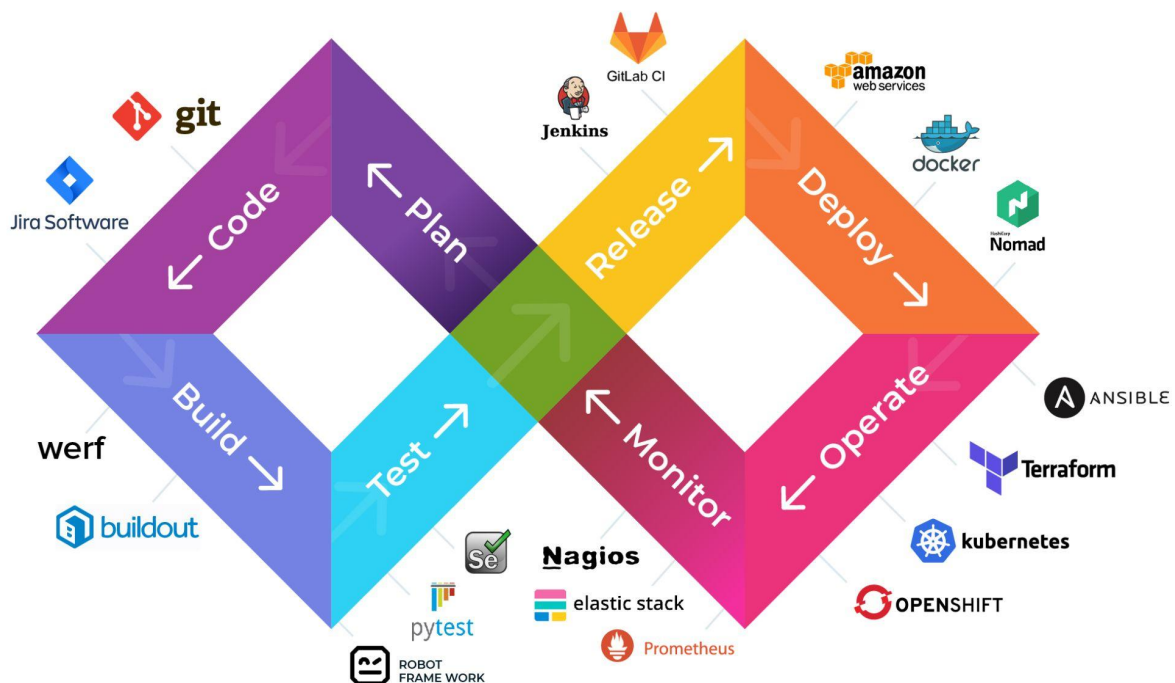


MINI PROJECT

SCIENTIFIC CALCULATOR USING DEVOPS

Akila L (DT2020002)



Github Link:

https://github.com/akilalakshmanan/ScientificCalculator_DevOps

DockerHub Link:

<https://hub.docker.com/r/akila1811/spe-mini-project>

Introduction

DevOps is the combination of practices and tools that increases the ability of an organization to deliver applications and services at high velocity which implies evolving and improving products in a quicker pace when compared to the organizations using traditional software development and infrastructure management processes.

This mini project is intended to build a Java based Scientific Calculator and integrate with DevOps tools like Jenkins, Docker and Ansible. This calculator performs square root function, natural logarithm, power function and finds factorials of two numbers.

The following are the steps involved in the overall process of building, integrating and deploying the application.

- a. Git Pull
- b. Maven Build
- c. Docker Build to image
- d. Pushing Docker image to docker hub
- e. Ansible pull Docker image

The above steps are performed as different stages in Continuous Integration – Continuous Development pipeline and Jenkins is used for the same.

Environment Set Up

In order to successfully develop and deploy our application with DevOps the following software packages, tools and plugins are required.

1. IntelliJ IDEA Community Edition v2020.3

This provides the environment for coding and testing the application. We can integrate with Git to directly commit and push the application to the GitHub repository. Download it from here using for your respective Operating Systems.

<https://www.jetbrains.com/idea/download/>

2. Git install and setting up GitHub

Git is a version control system which will contain our entire code in the form of a repository. Whatever changes are being made to our project can be directly pushed to our GitHub repository using either Git or Git plugin of IntelliJ. Command for git installation is,

```
$ sudo apt install git-all
```

3. Setting up Maven on IntelliJ

Maven is a build tool which resolves various dependencies present in our project like JUnit, log4j etc. It builds and packages the project in the form of a .jar file. Open a new Java application on IntelliJ as a Maven project.

4. Installing Jenkins and setting up the Jenkins dashboard

Jenkins is a powerful DevOps tool which streamlines our continuous integration and development in a single pipeline. To install Jenkins, use the following command.

<https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-18-04>

Make sure to install various plugins like Git, GitHub, Docker and Ansible over the Jenkins which helps in our integration.

5. Installing Docker and making an account on Docker Hub

Docker is a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from each other and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. Install it using the following link.

<https://docs.docker.com/engine/install/ubuntu/>

6. Setting up Ansible

Ansible is a configuration management tool which is used to deploy the docker image and run the .jar file inside another host specified in Ansible inventory. Refer the following link to install and setup Ansible.

https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Stages involved in Jenkins Pipeline

Jenkins is mainly used to establish the complete workflow in a smooth, efficient, and well-organized manner. It provides various plugins for each and every stage of our pipeline which helps in continuous integration and continuous development. Whenever the developer makes a push to the GitHub repository, Jenkins will automatically perform the following steps.

Stage 1: Git Pull

This stage involves creating a public GitHub repository and adding a ReadMe for the project. Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Basic Git Commands include,

Initialize the local directory as a Git repository

```
$ git init
```

Add the files to your new local repository. This stages them for the first commit.

```
$ git add .
```

Commit the files that you have staged in your local repository.

```
$ git commit -m "First commit"
```

This commits the tracked changes and prepares them to be pushed to a remote repository.

To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and add the file again.

In Terminal, add the URL for the remote repository where your local repository will be pushed.

Set the new remote.

```
$ git remote add origin <REMOTE_URL>
```

```
$ git remote -v # Verifies the new remote URL
```



Push the changes in your local repository to GitHub. Pushes the changes in your local repository up to the remote repository you specified as the origin.

```
$ git push origin main
```

Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)


Owner * **Repository name ***

 akilalakshmanan / ScientificCalculator_DevOps 

Great repository names are short and memorable. Need inspiration? How about [miniature-computing-machine?](#)

Description (optional)

☒  **Public**
Anyone on the Internet can see this repository. You choose who can commit.


☐  **Private**
You choose who can see and commit to this repository.


Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

[Create repository](#)

Fig 1.1. Creating a public Github repository

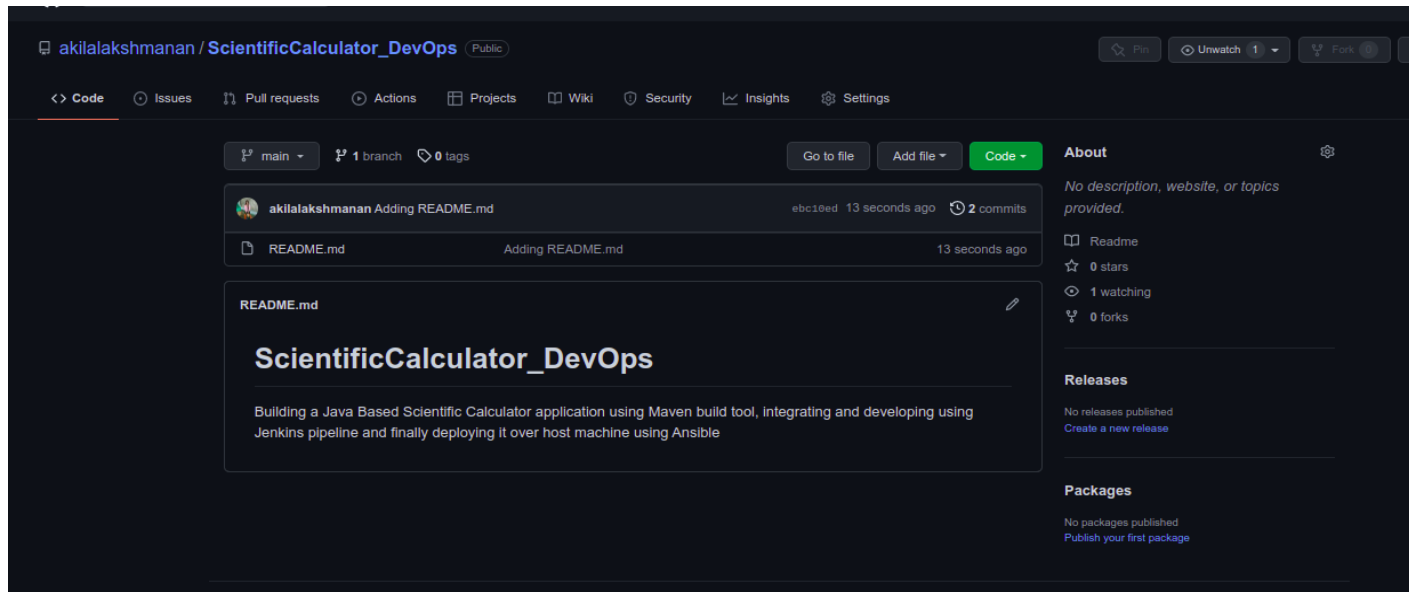


Fig 1.2. Adding ReadMe file

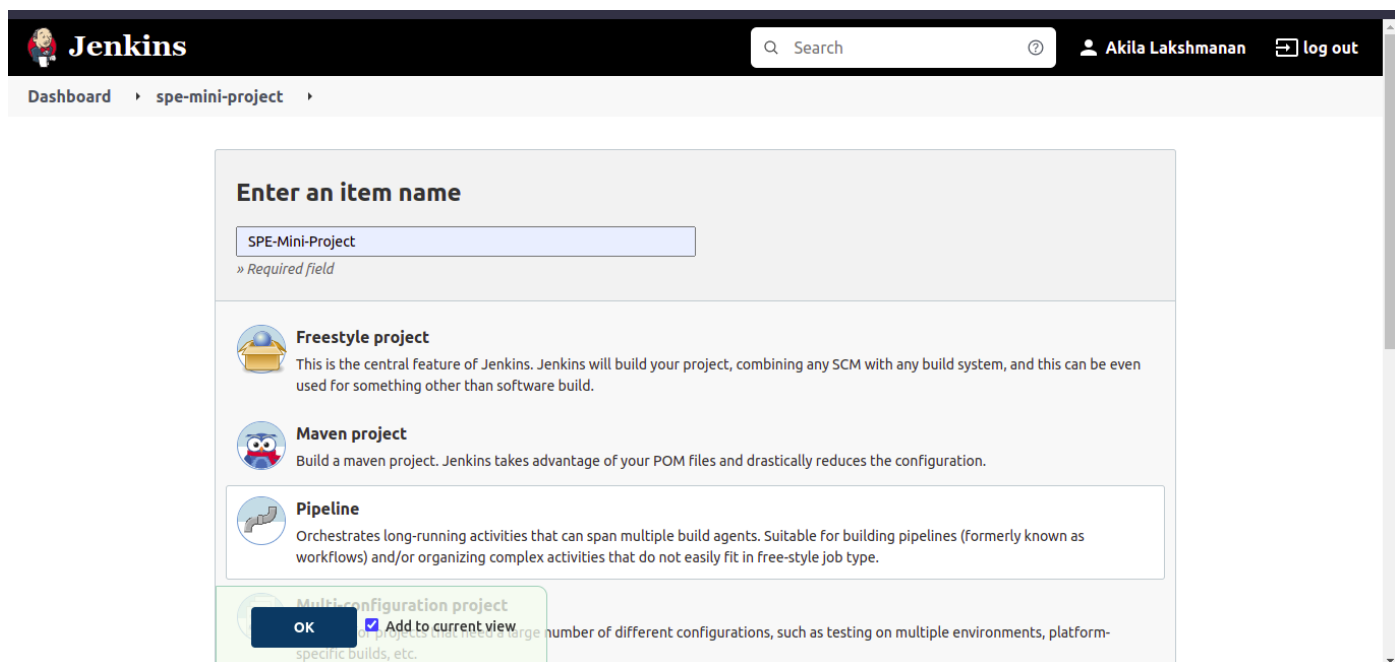


Fig 1.3. Logging into Jenkins as local host and creating a new job as pipeline

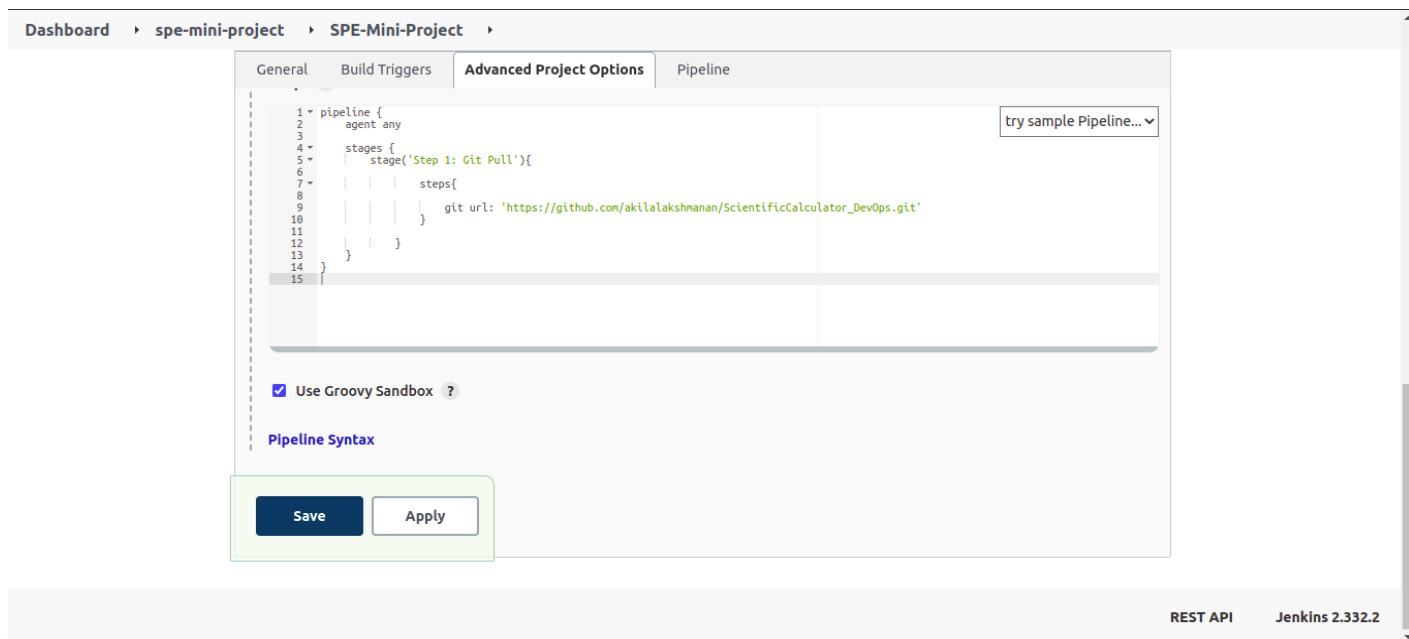


Fig 1.4. Adding Pipeline script for Git Pull

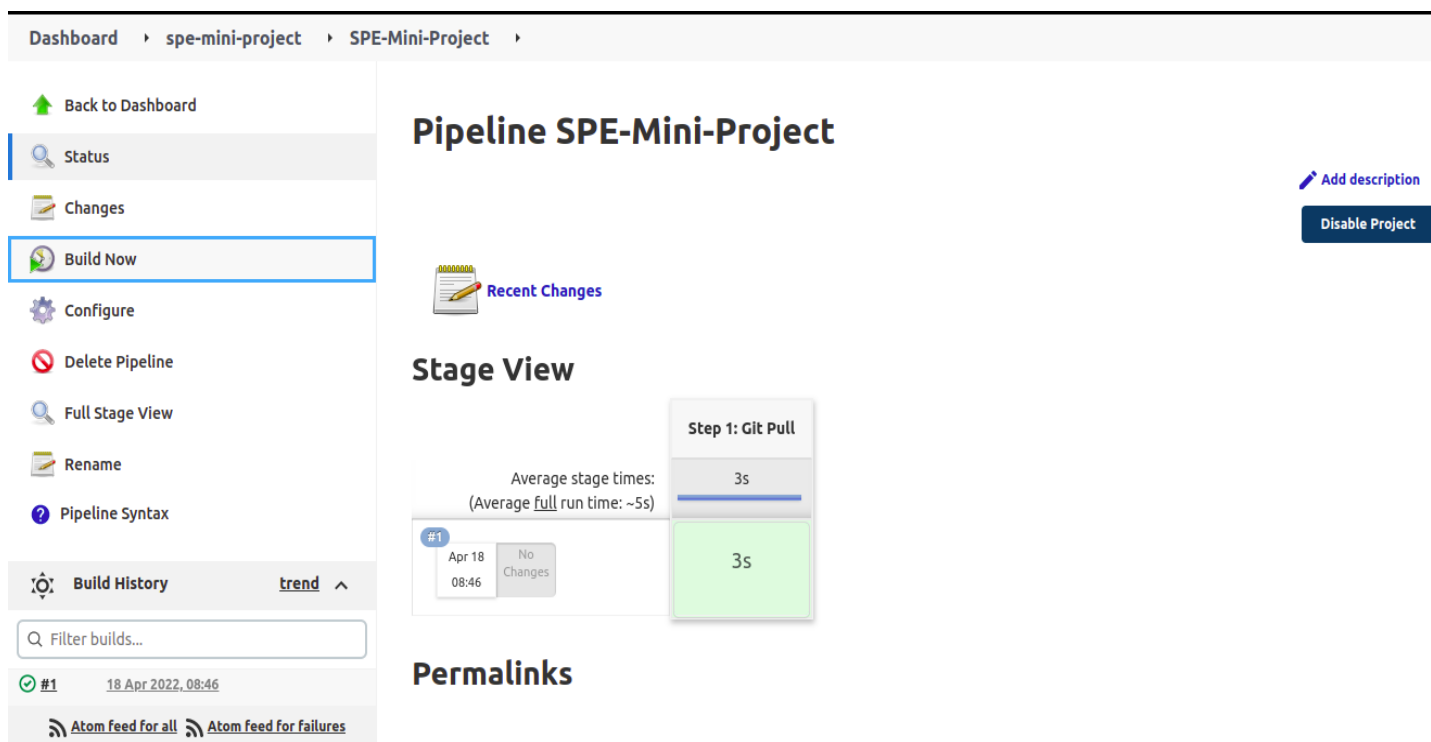


Fig 1.5. Successfully built Stage 1 of Pipeline

Pipeline Code for Git Pull is as follows.

```

pipeline {
    agent any

```

```

stages {
stage('Step 1: Git Pull'){
    steps{
        git url: 'https://github.com/akilalakshmanan/ScientificCalculator_DevOps.git'
    }
}
}

```

Adding Github Hook Trigger for Git SCM Polling for Jenkins

Whenever we commit new changes to our Github repository, Jenkins will automatically detect there are some new changes in the Github and hence Jenkins will trigger the job to build.

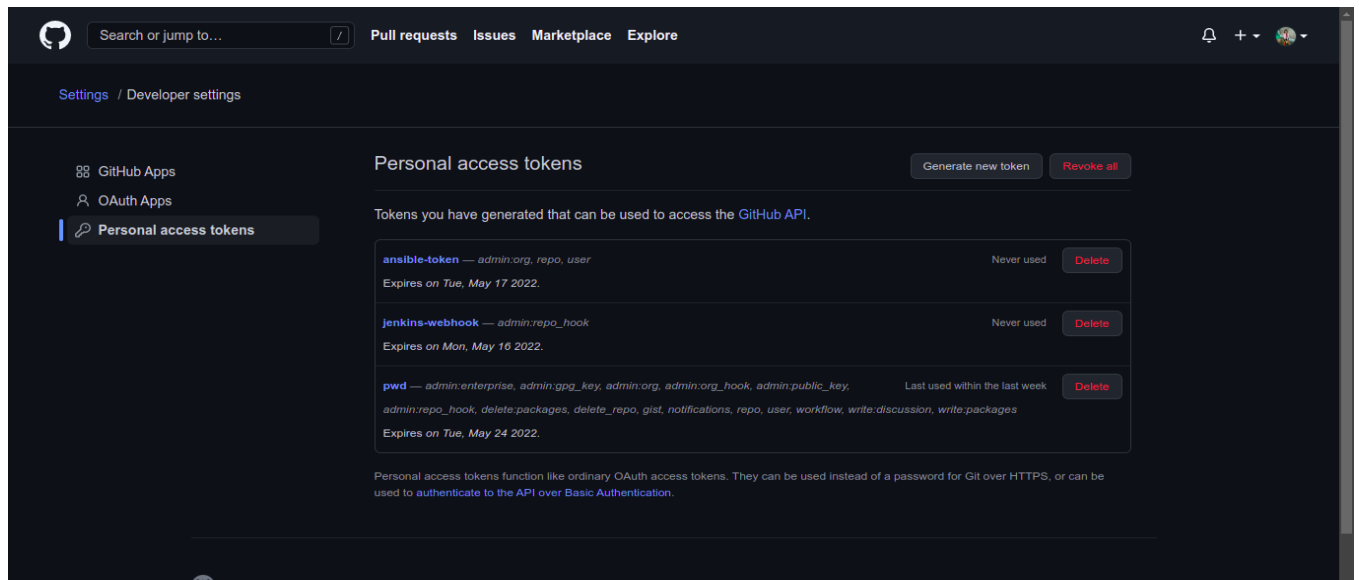


Fig 1.6. Generating new personal access token in GitHub

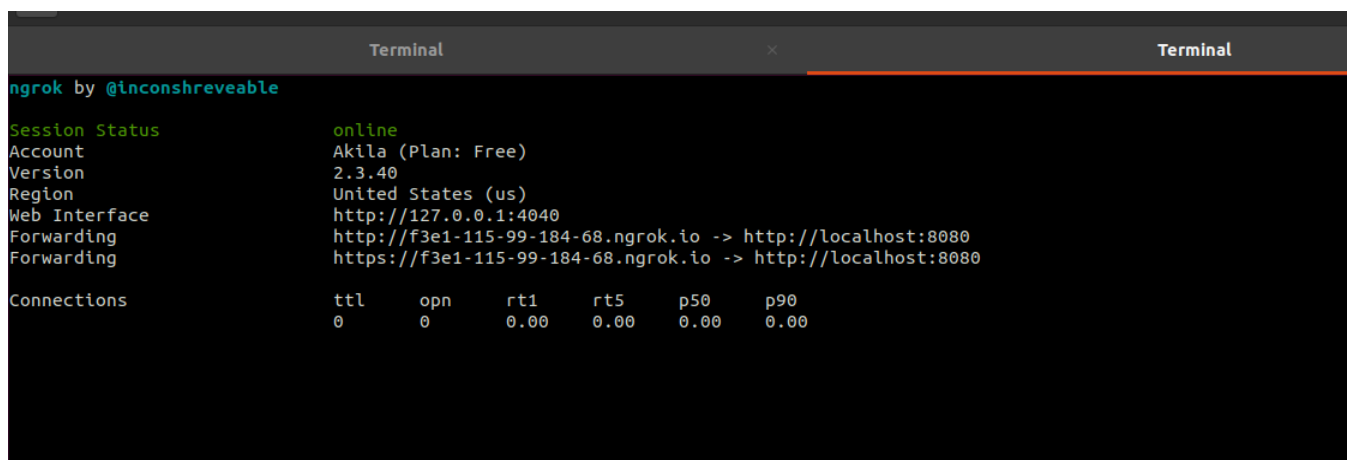


Fig 1.7. Ngrok configuration

General

Access

- Collaborators
- Moderation options

Code and automation

- Branches
- Tags
- Actions
- Webhooks**
- Environments
- Pages

Security

- Code security and analysis
- Deploy keys
- Secrets

Integrations

- GitHub apps
- Email notifications

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our [developer documentation](#).

Payload URL *

`http://f3e1-115-99-184-68.ngrok.io/github-webhook/`

Content type

`application/x-www-form-urlencoded`

Secret

`ghp_j2C3NnWnl6rv0SgH8cG0TRbgm0xyaX4aehog`

Which events would you like to trigger this webhook?

- ☒ Just the push event.
- ☐ Send me everything.
- ☐ Let me select individual events.

☒ **Active**
We will deliver event details when this hook is triggered.

Add webhook

Fig 1.8. Adding a webhook in our GitHub repository

Dashboard > configuration

☐ Restrict project naming

Jenkins Location

Jenkins URL ?

`http://f3e1-115-99-184-68.ngrok.io/`

System Admin e-mail address ?

Jenkins-Master <akilalakshman18@gmail.com>

Serve resource files from another domain

Resource Root URL ?

Without a resource root URL, resources will be served from the Jenkins URL with Content-Security-Policy set.

Global properties

- ☐ Disable deferred wipeout on this node ?
- ☐ Environment variables
- ☐ Tool Locations

Save **Apply**

Fig 1.9.1. GitHub server configuration in Jenkins

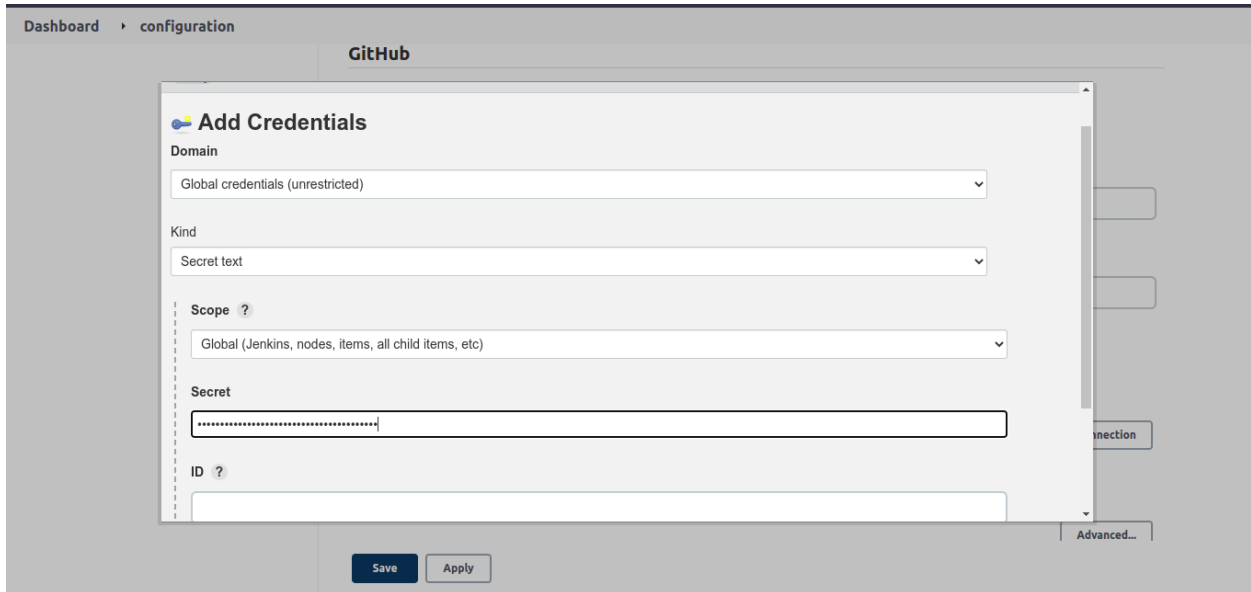


Fig 1.9.2. Adding credentials for GitHub server configuration in Jenkins

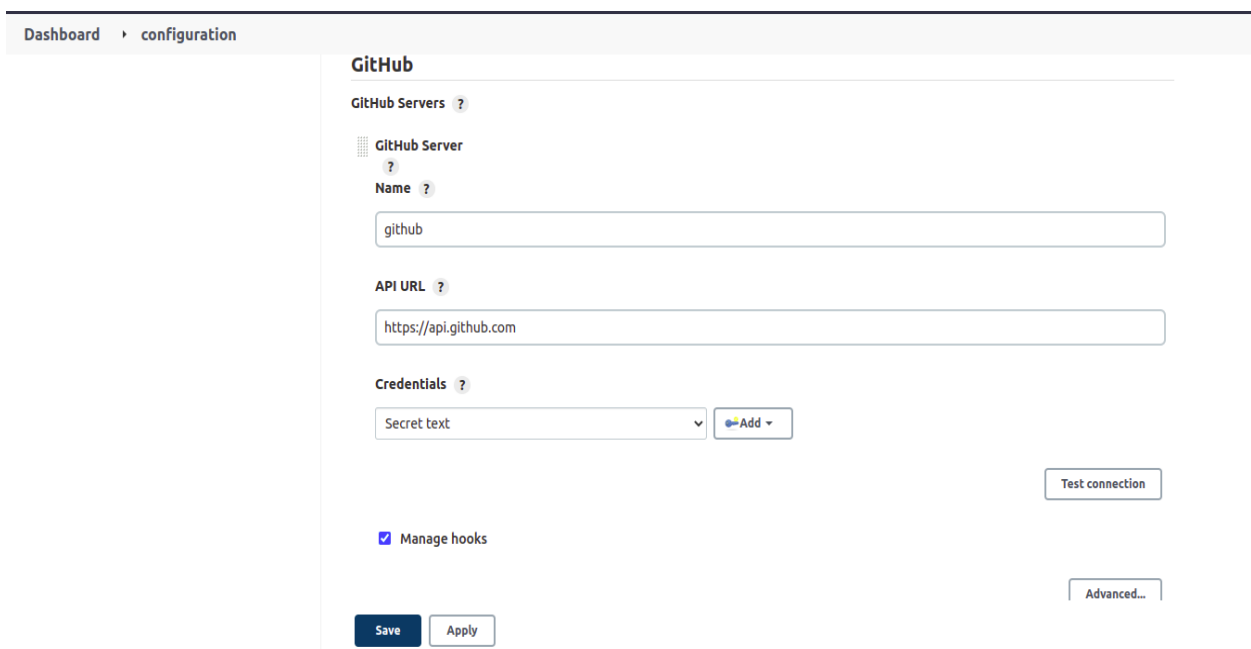


Fig 1.9.3. GitHub server configuration in Jenkins

Stage 2: Maven Build

In this the entire project is being compiled, validated and tested by Maven and Junit. All this is handled by the Jenkins pipeline which outputs a target folder containing the .jar file.

Maven is a build automation tool used primarily for Java projects. Maven can also be used to build and manage projects written in C#, Ruby, Scala, and other languages. The Maven project is hosted by the Apache Software Foundation.

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development. JUnit is linked as a JAR at compile-time.

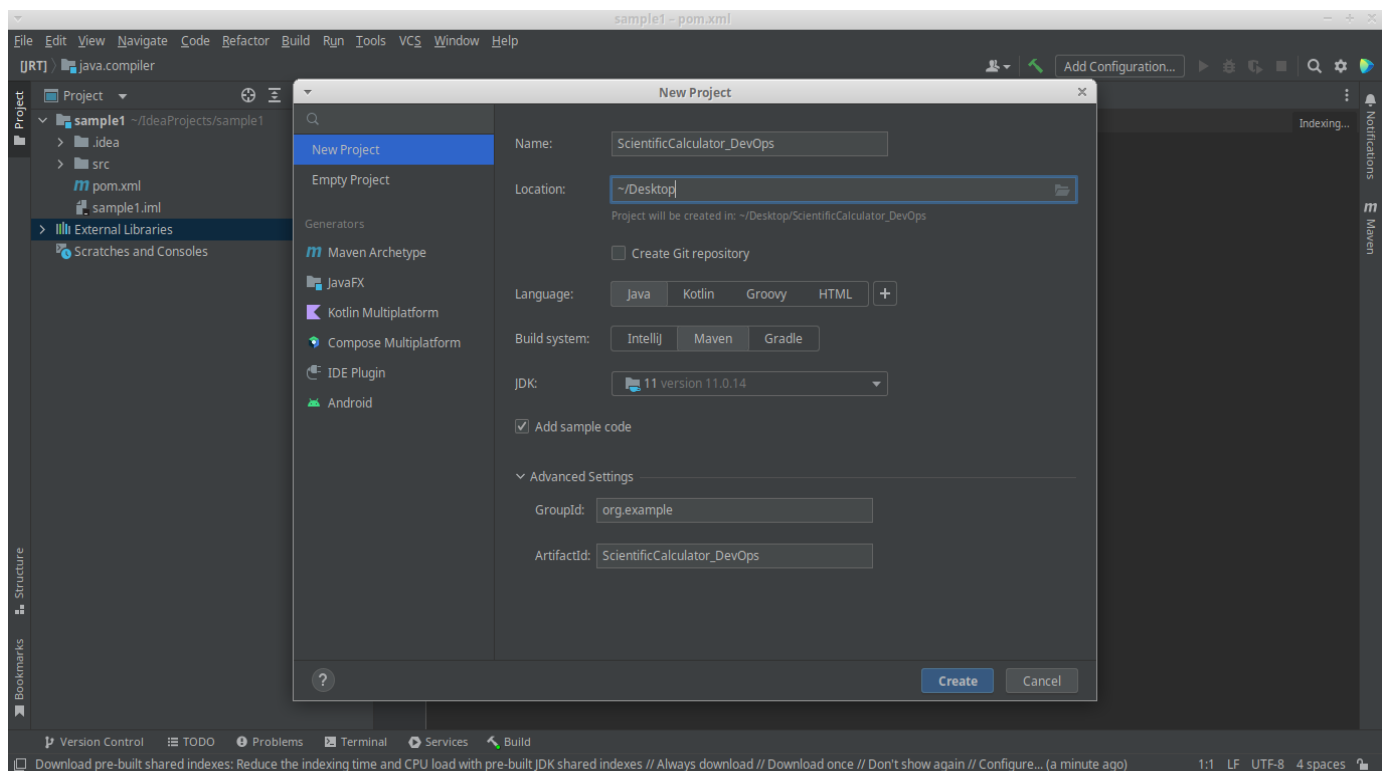


Fig 2.1. Creating a New Maven Project on IntelliJ

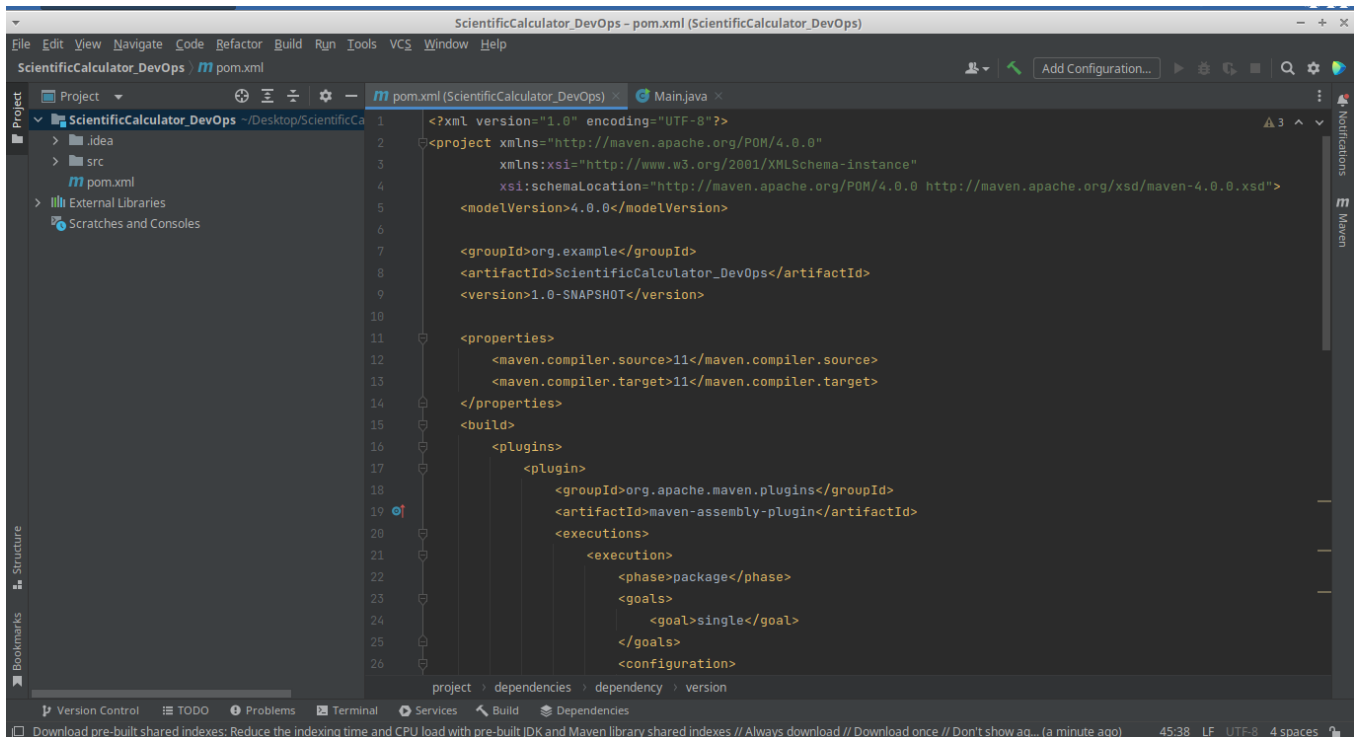


Fig 2.2. Adding Dependencies to pom.xml

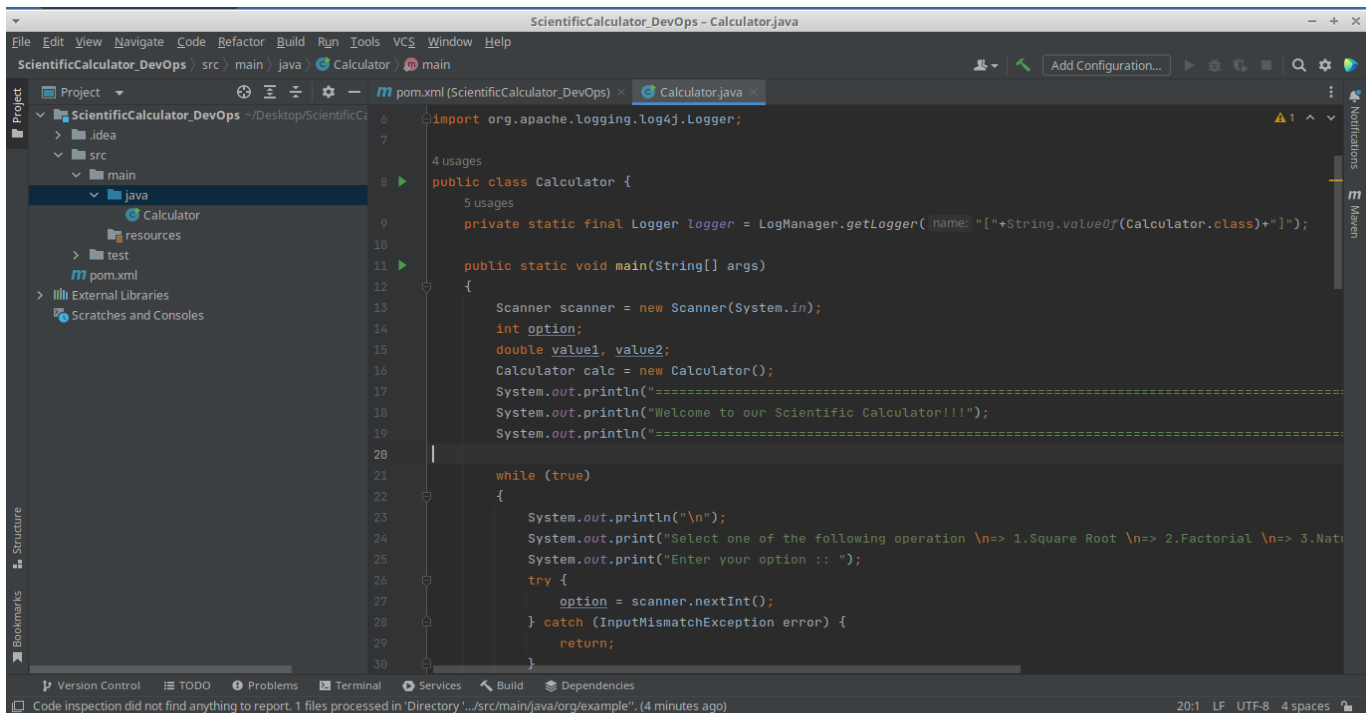


Fig 2.3. Writing the code for Scientific Calculator in Calculator.java

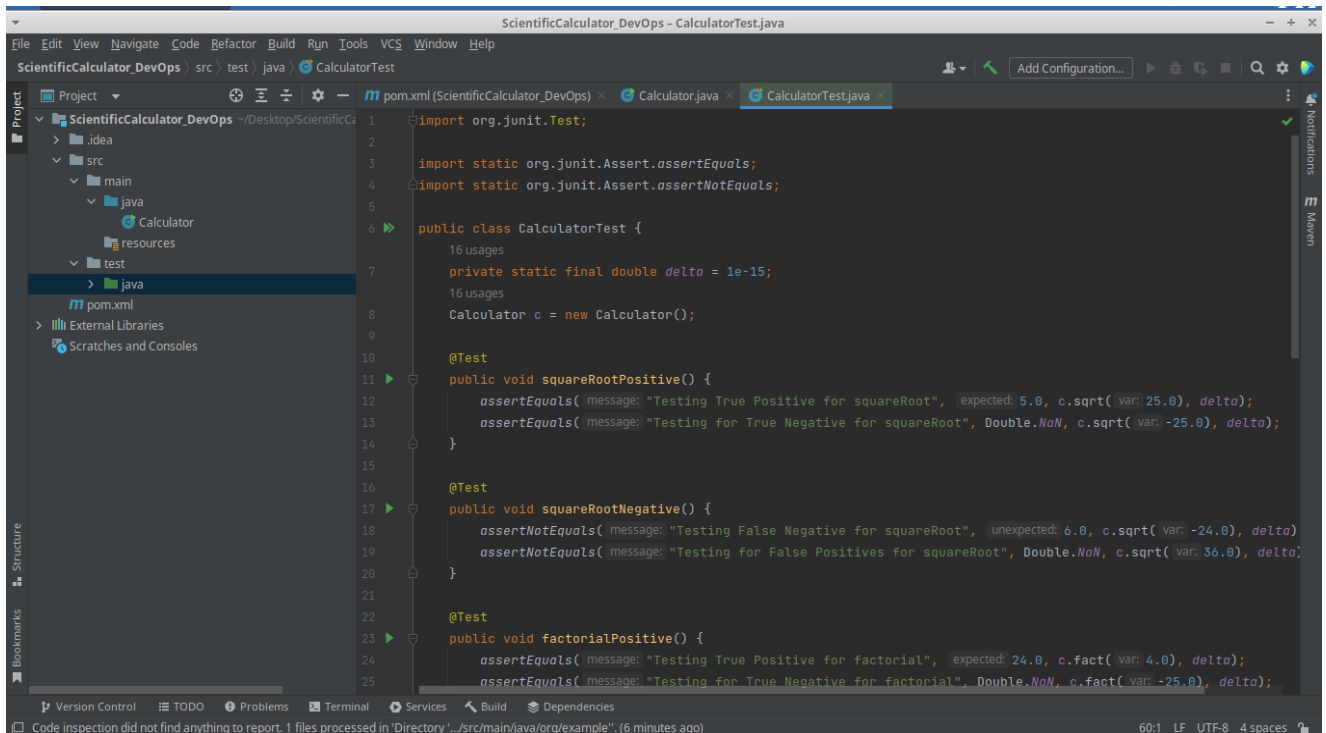


Fig 2.4. Writing Test Cases in CalculatorTest.java

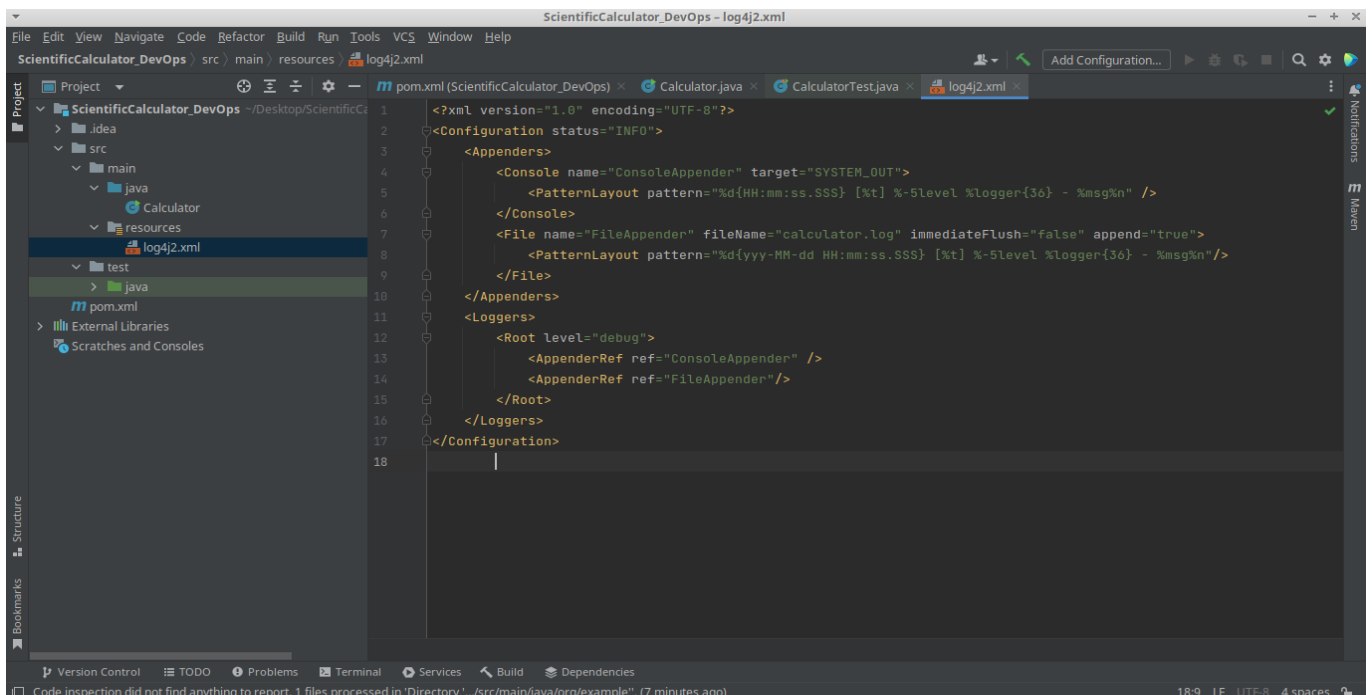


Fig 2.5. Making logger configuration file as log4j2.xml

```
sitory/org/example/ScientificCalculator_DevOps/1.0-SNAPSHOT/ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.812 s
[INFO] Finished at: 2022-04-18T10:52:04+05:30
[INFO] -----
```

Fig 2.6. Building using command mvn clean package

```
akila@akila:~$ cd Desktop/ScientificCalculator_DevOps/
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git init
Initialized empty Git repository in /home/akila/Desktop/ScientificCalculator_DevOps/.git/
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git add .
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git commit -m "Adding source code"
[master (root-commit) f1408ed] Adding source code
 21 files changed, 443 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/compiler.xml
 create mode 100644 .idea/jarRepositories.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 calculator.log
 create mode 100644 pom.xml
 create mode 100644 src/main/java/Calculator.java
 create mode 100644 src/main/resources/log4j2.xml
 create mode 100644 src/test/java/CalculatorTest.java
 create mode 100644 target/ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar
 create mode 100644 target/ScientificCalculator_DevOps-1.0-SNAPSHOT.jar
 create mode 100644 target/classes/Calculator.class
 create mode 100644 target/classes/log4j2.xml
 create mode 100644 target/maven-archiver/pom.properties
 create mode 100644 target/maven-status/maven-compiler-plugin/compile/default-compile/createdFiles.lst
 create mode 100644 target/maven-status/maven-compiler-plugin/compile/default-compile/inputFiles.lst
 create mode 100644 target/maven-status/maven-compiler-plugin/testCompile/default-testCompile/createdFiles.lst
 create mode 100644 target/maven-status/maven-compiler-plugin/testCompile/default-testCompile/inputFiles.lst
 create mode 100644 target/surefire-reports/CalculatorTest.txt
 create mode 100644 target/surefire-reports/TEST-CalculatorTest.xml
 create mode 100644 target/test-classes/CalculatorTest.class
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git remote add origin https://github.com/akilalakshmanan/ScientificCalculator_DevOps.git
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git push origin master
```

Fig 2.7.1. Committing and pushing changes to GitHub using Git commands

```
fatal: Remote origin already exists.
akila@akila:~/Desktop/ScientificCalculator_DevOps$ git push origin master
Username for 'https://github.com': akilalakshmanan
Password for 'https://akilalakshmanan@github.com':
Enumerating objects: 40, done.
Counting objects: 100% (40/40), done.
Delta compression using up to 4 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (40/40), 1.76 MiB | 1.46 MiB/s, done.
Total 40 (delta 0), reused 0 (delta 0)
To https://github.com/akilalakshmanan/ScientificCalculator_DevOps.git
 * [new branch] master -> master
akila@akila:~/Desktop/ScientificCalculator_DevOps$
```

Fig 2.7.2. Committing and pushing changes to GitHub using Git commands

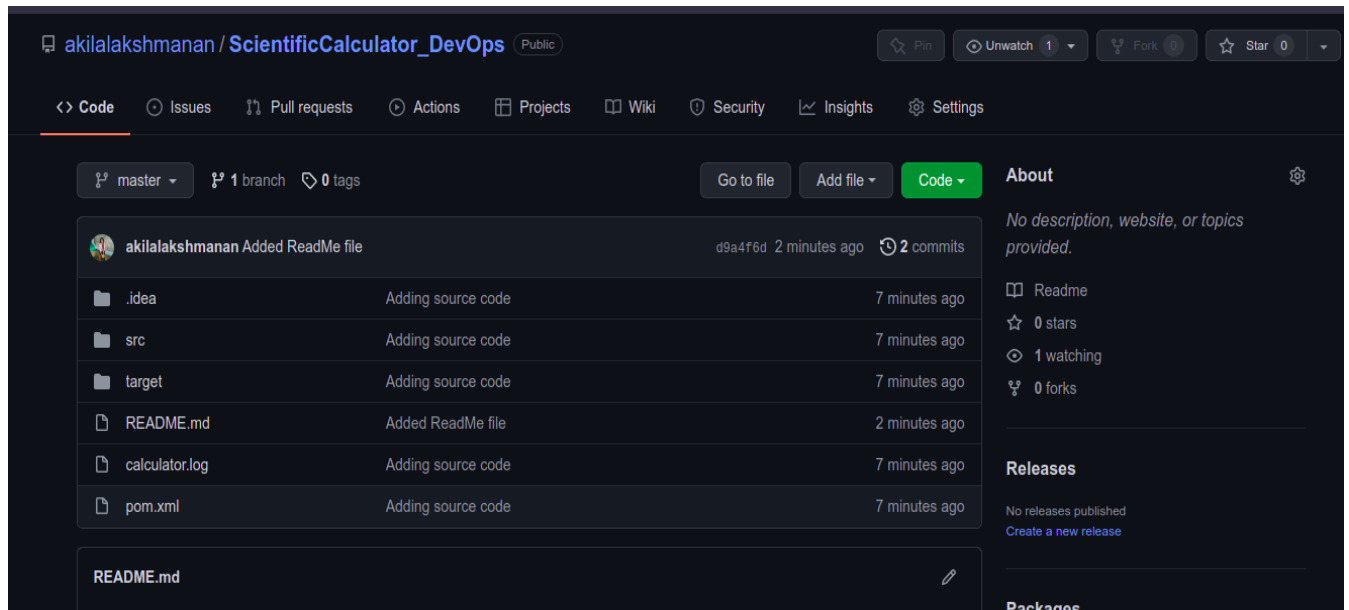


Fig 2.8. Changes visible on GitHub

Pipeline script for Maven is as follows.

```
stage('Step 2: Maven Build'){
    steps{
        script{
            sh 'mvn clean install'
            sh 'sudo cp
/var/lib/jenkins/workspace/SPE-Mini-Project/target/ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar /home/akila/Desktop/ScientificCalculator_DevOps/target/'
        }
    }
}
```

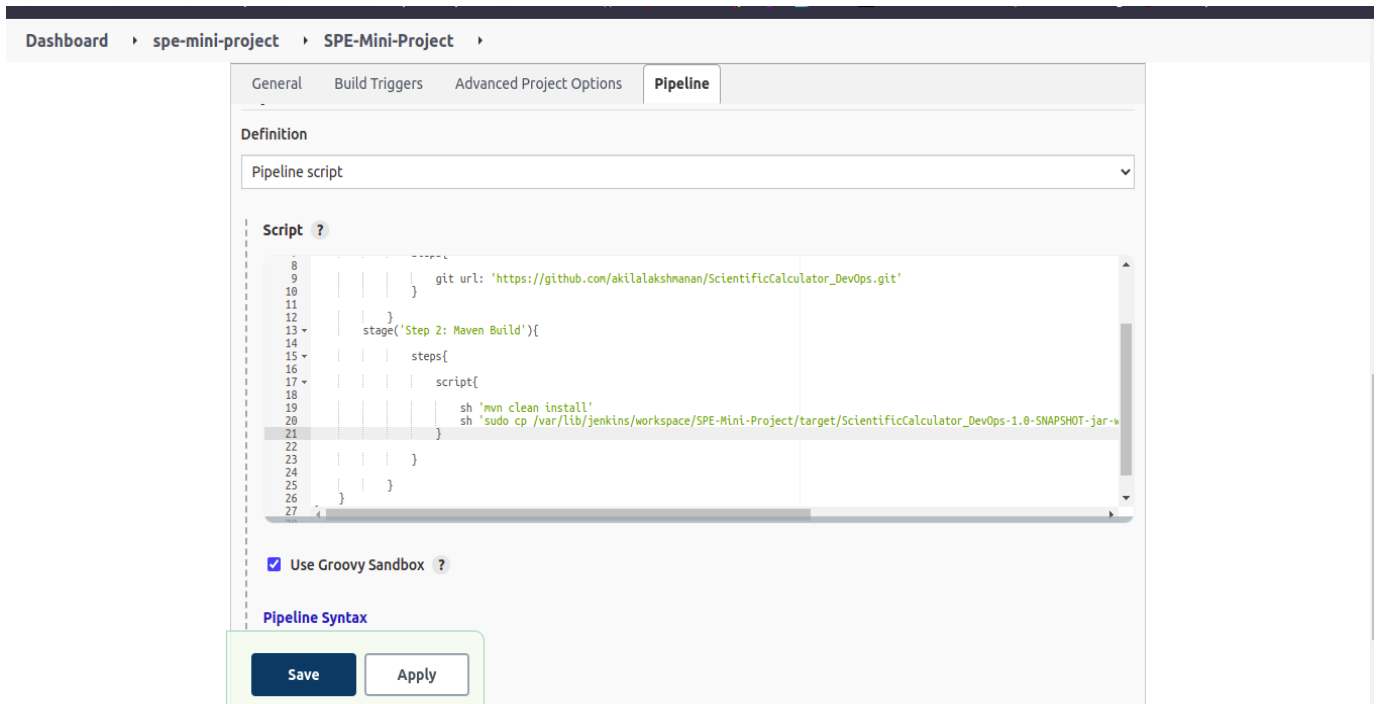


Fig 2.9. Pipeline script included in Jenkins

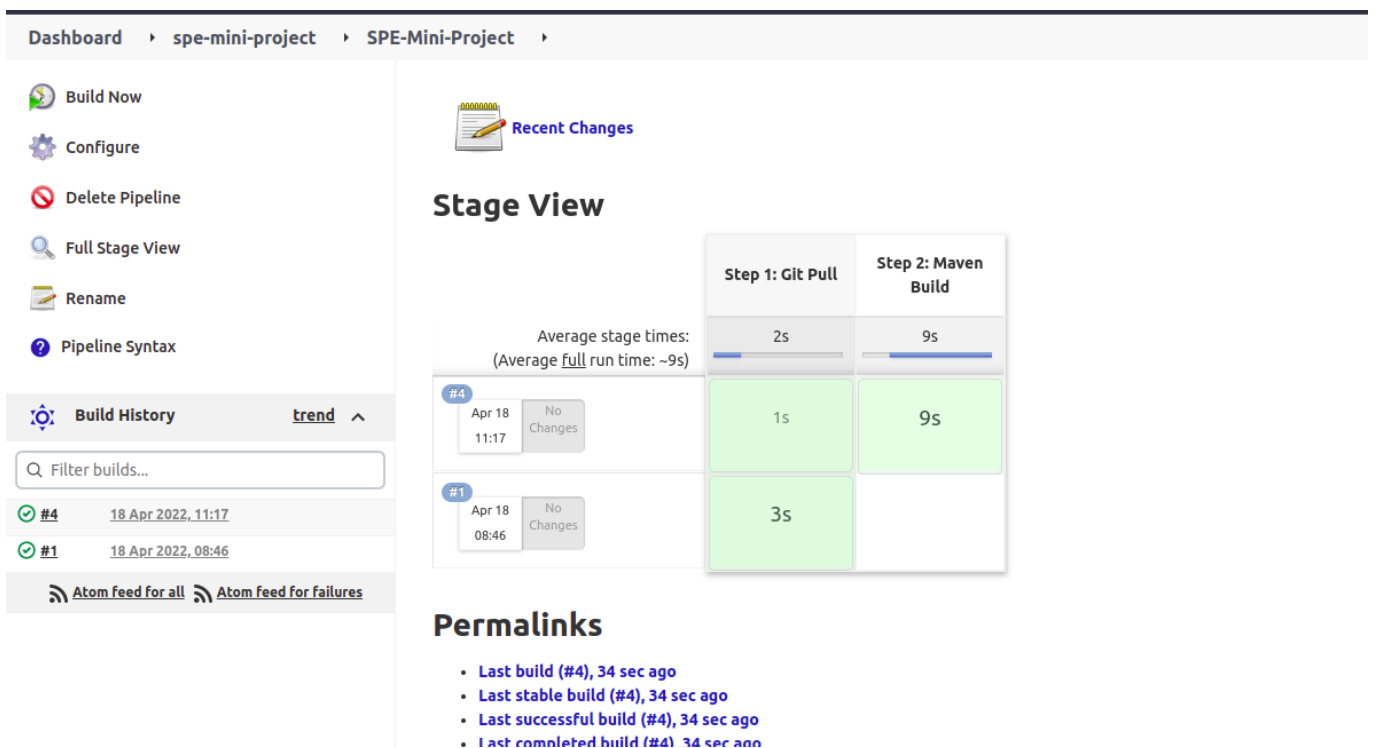


Fig 2.10. Successfully built Stage 2 of Pipeline

Stage 3: Docker Build to image

This step involves containerizing a .jar file in a Docker image. For this we first create a Dockerfile in our project and add the name of our .jar file accordingly.

Dockerfile

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

Then push this Dockerfile to our Github account

Pipeline script for building docker image is as follows.

```
stage('Step 3: Build docker image')
{
    steps{
        script{
            dockerImage = docker.build 'akila1811/spe-mini-project'
        }
    }
}
```

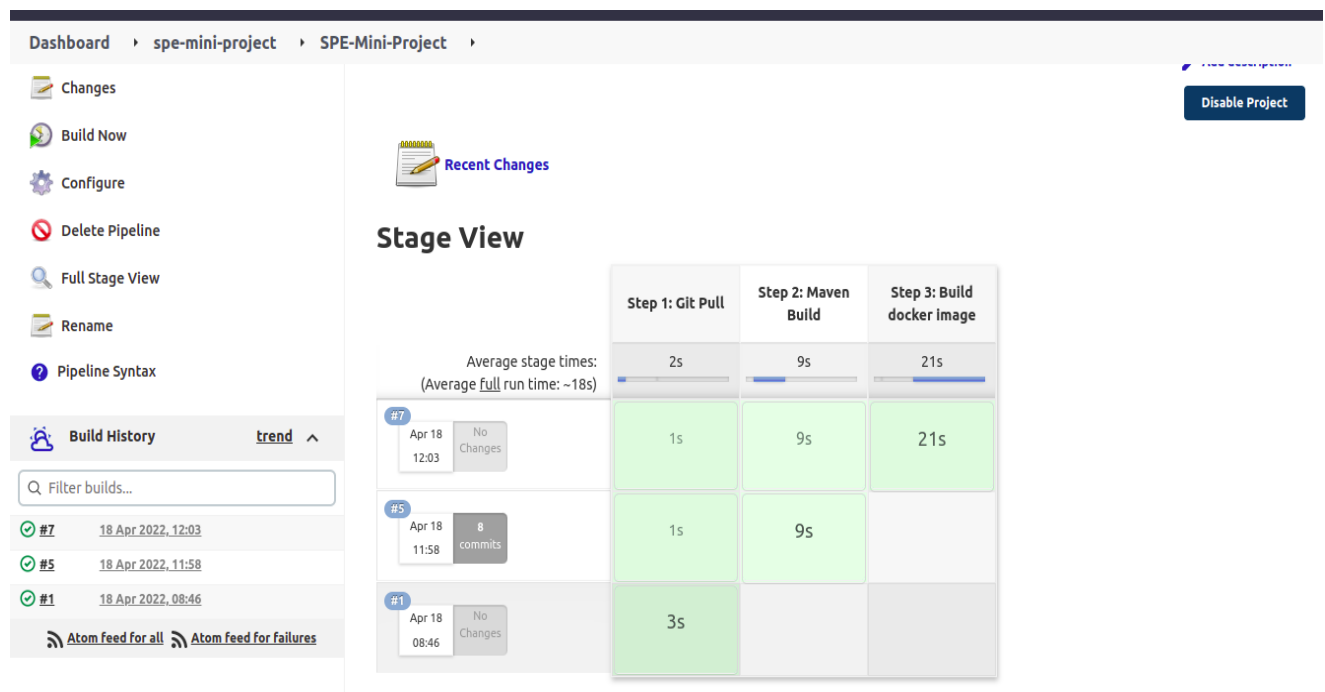


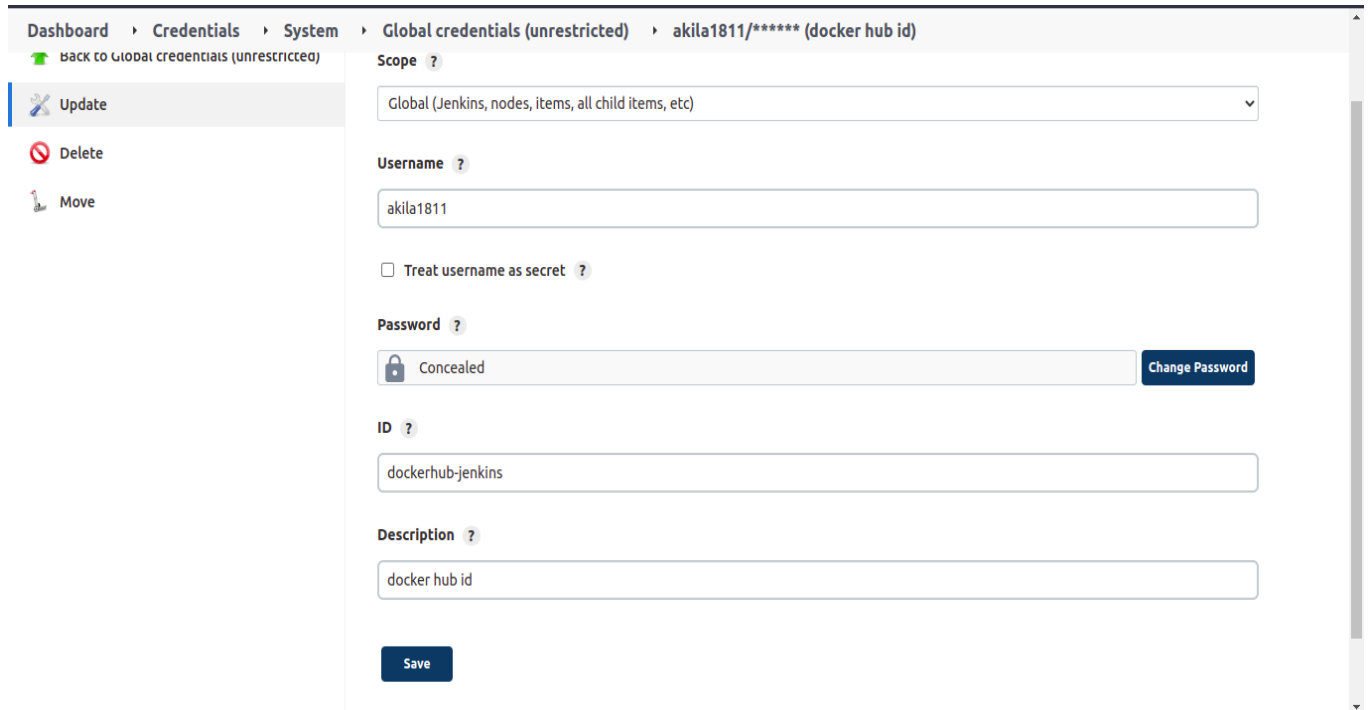
Fig 3.1: Successfully containerized .jar file to Docker Image

Stage 4: Pushing Docker image

In this step, we push our docker image to our public Docker Hub repository.

Pipeline script for this stage is as follows.

```
stage('Step 4: Pushing docker image')
{
    steps{
        script {
            docker.withRegistry("registryCredential") {
                dockerImage.push()
            }
        }
    }
}
```



The screenshot shows the Jenkins web interface for configuring a global credential. The breadcrumb trail at the top is: Dashboard > Credentials > System > Global credentials (unrestricted) > akila1811/***** (docker hub id). On the left sidebar, there are three actions: 'Update' (with a wrench icon), 'Delete' (with a red circle and slash icon), and 'Move' (with a hand icon). The main form area contains the following fields:

- Scope**: A dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'.
- Username**: A text input field containing 'akila1811'.
- Treat username as secret**: An unchecked checkbox.
- Password**: A text input field with a lock icon and the text 'Concealed'. To its right is a 'Change Password' button.
- ID**: A text input field containing 'dockerhub-jenkins'.
- Description**: A text input field containing 'docker hub id'.

At the bottom of the form is a 'Save' button.

Fig 4.1. Configuring Docker Hub Credentials in Jenkins

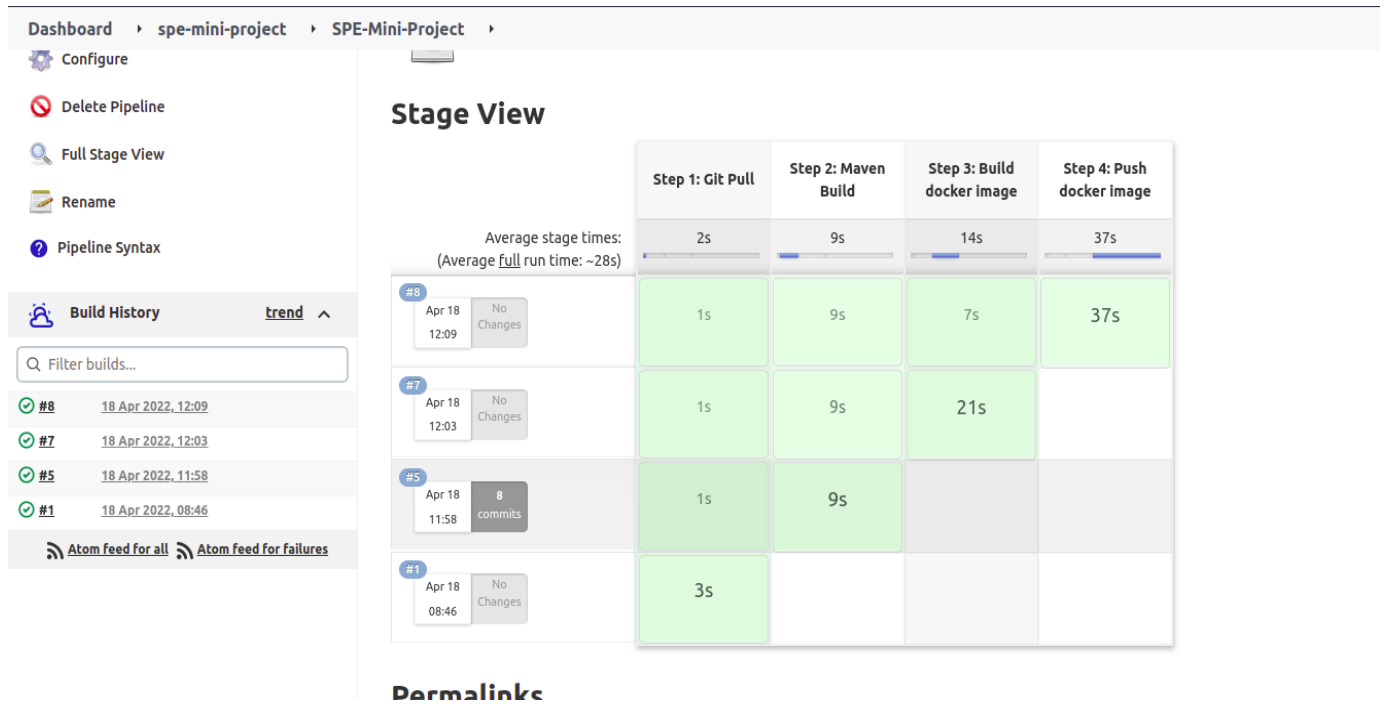


Fig 4.2. Pushed Docker image to Docker Hub

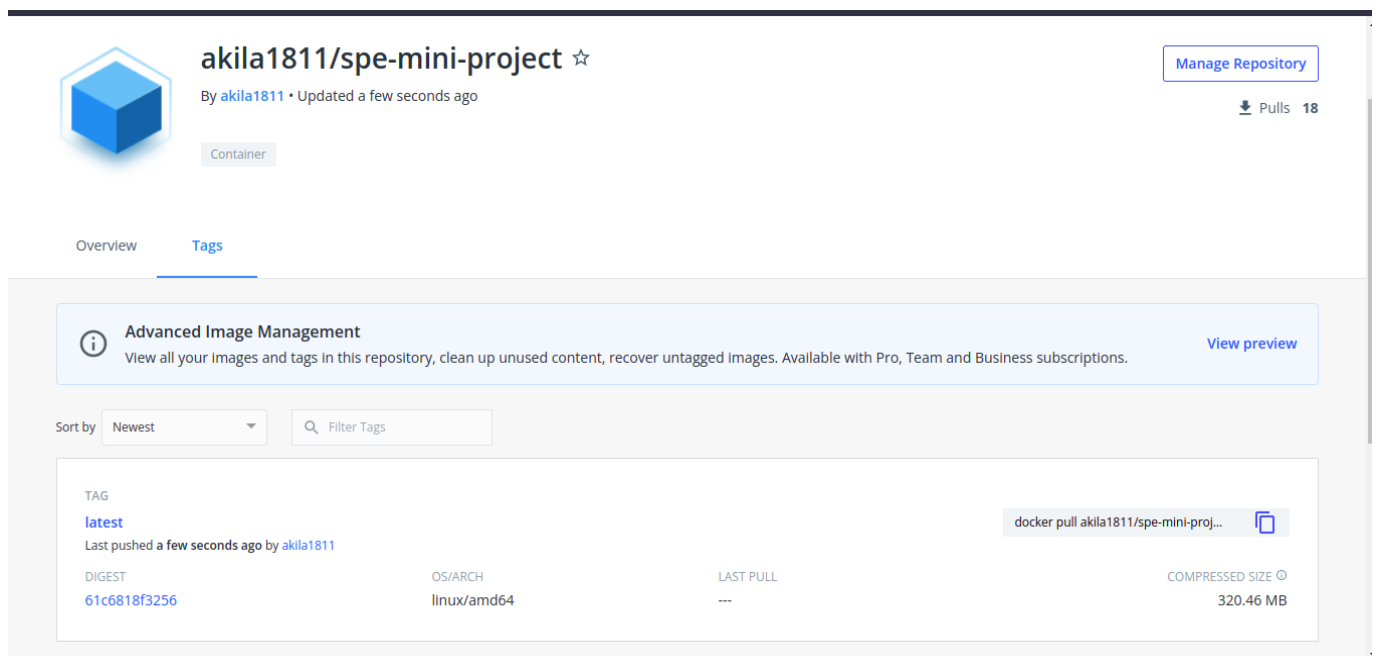


Fig 4.3. Changes reflected to Docker Hub Repository

Stage 5: Ansible pull Docker image

In this step we install Ansible on the controller node and deploy our docker image in all our hosts. In this step we first create a file named calculator-playbook.yml which contains the instructions required to deploy the docker image to our hosts.

calculator-playbook.yml

```
---
- hosts: all
  tasks:
    - name: stop container
      command: docker stop calc-container
      ignore_errors: yes
    - name: remove container
      command: docker rm calc-container
      ignore_errors: yes
    - name: Pull image from dockerhub
      command: docker pull akila1811/spe-mini-project:latest
    - name: Building docker container from image
      command: docker run -d -it --name calc-container akila1811/spe-mini-project:latest
```

The inventory file is added which contains the list of managed hosts where our application needs to be deployed. An inventory can also organize managed nodes, creating and nesting groups for easier scaling. The inventory file used in this project is as follows.

inventory

```
[xubuntu]
192.168.0.105 ansible_pass=1811 ansible_user=gokul
```

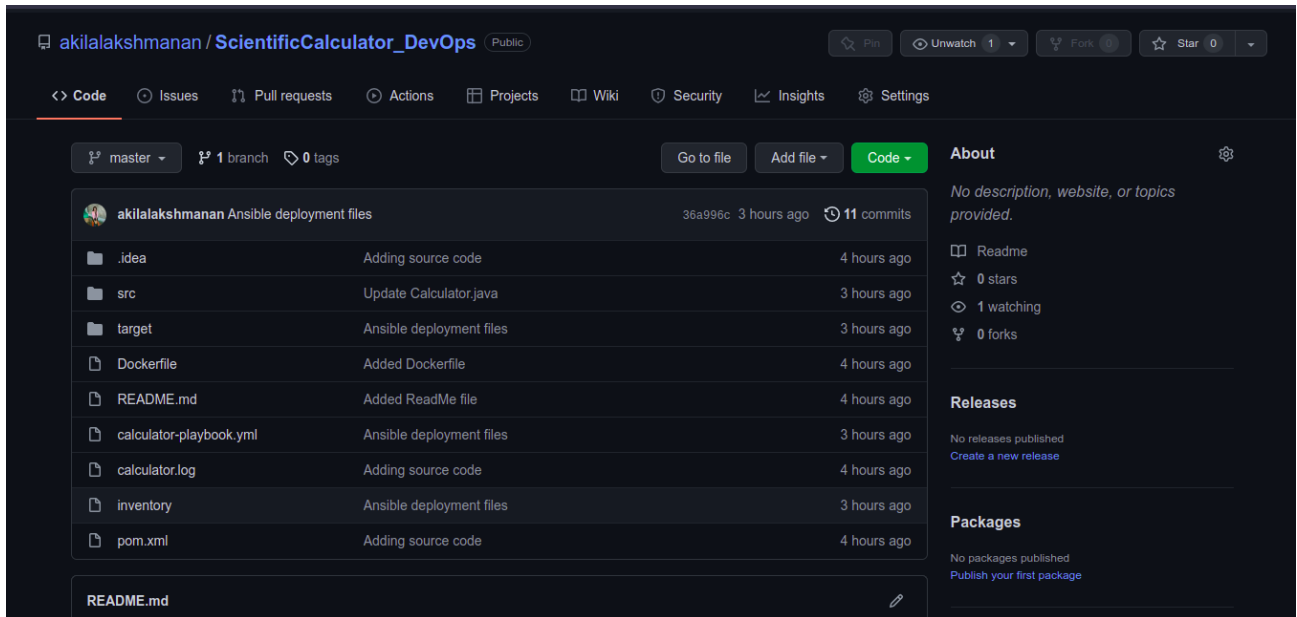


Fig 5.1. Added .yml and inventory file to the Github account

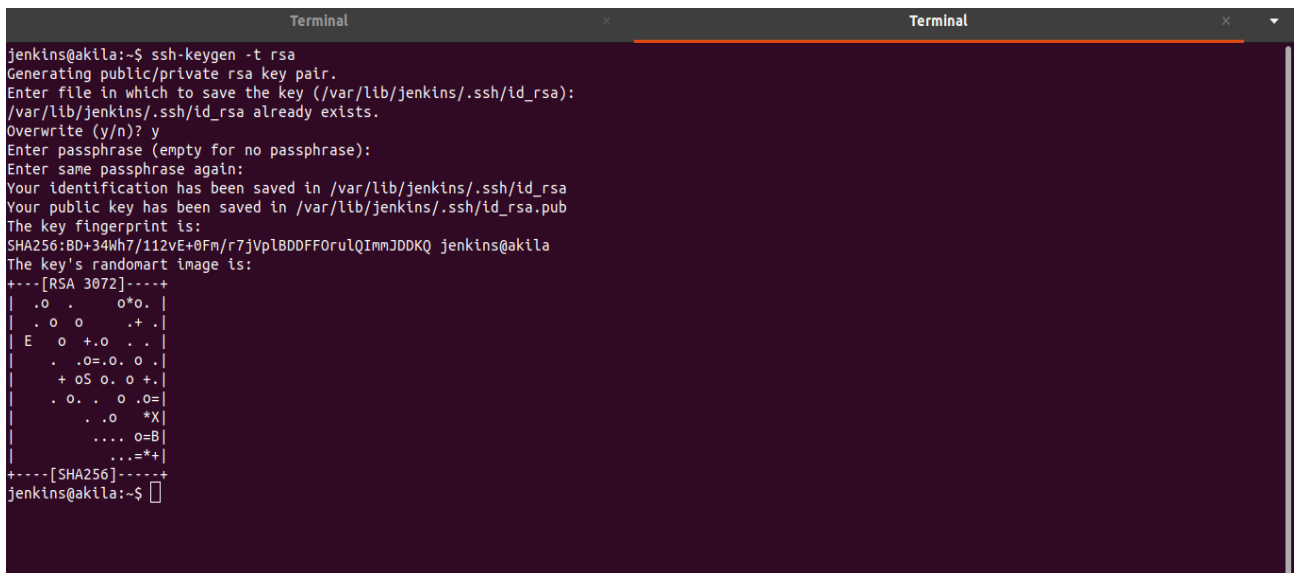


Fig 5.2. Generate RSA Key-pair

```
Terminal
gokul@gokul-xubuntu: ~

jenkins@akila:~$ ssh-copy-id gokul@192.168.0.105
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/var/lib/jenkins/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
gokul@192.168.0.105's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'gokul@192.168.0.105'"
and check to make sure that only the key(s) you wanted were added.

jenkins@akila:~$
jenkins@akila:~$ ssh gokul@192.168.0.105
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-39-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 updates can be applied immediately.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Mon Apr 18 07:21:43 2022 from 192.168.43.160
gokul@gokul-xubuntu:~$
```

Fig 5.3. Copy ssh key and login to remote user

Now we add our Jenkins pipeline syntax for Ansible,

stage('Step 5: Ansible pull docker image')

```
{
    steps{
        ansiblePlaybook colorized: true, disableHostKeyChecking: true, installation:
'Ansible', inventory: 'inventory', playbook: 'calculator-playbook.yml'
    }
}
```

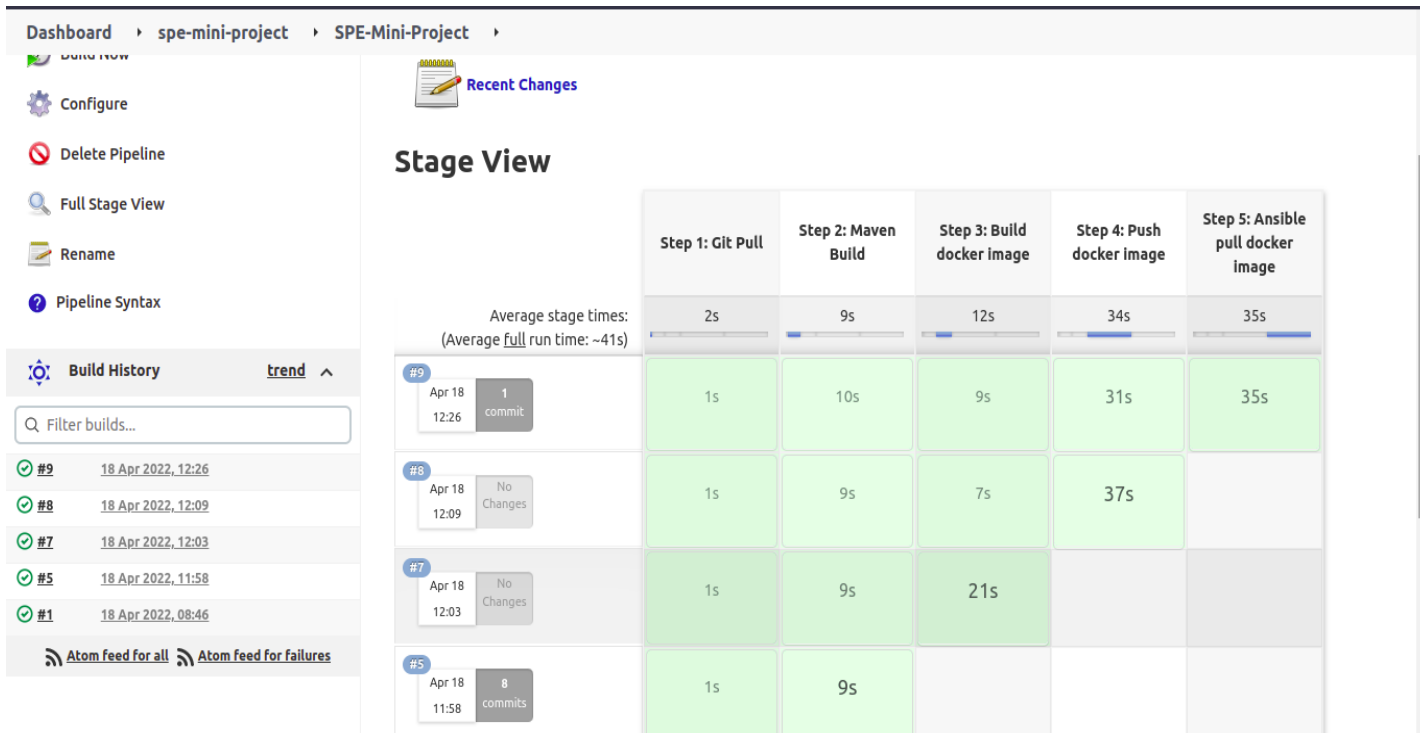


Fig 5.4. Docker image successfully deployed to our host machine

```

Terminal
gokul@gokul-xubuntu: ~
gokul@gokul-xubuntu:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS   NAMES
304f650c55bd   akila1811/spe-mini-project:latest   "java -jar Scientifi..." About a minute ago Up About a minute           calc-container
gokul@gokul-xubuntu:~$ docker exec -it calc-container bash
root@304f650c55bd:/# ls
ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar boot dev home lib64 mnt proc run srv tmp var
bin calculator.log etc lib media opt root sbin sys usr
root@304f650c55bd:/# java -jar ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
=====
Welcome to Scientific Calculator!!!
=====

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 1

===== [ Square Root ] =====
Enter the number: 81
07:01:25.321 [main] INFO [class Calculator] - Square Root of [ 81.0 ]
Result [ 9.0 ]
Square Root of 81.0 = 9.0

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 2

```

Fig 5.5.1. .jar file running on Docker image pulled from the Docker hub on our host machine

```
Terminal gokul@gokul-xubuntu: ~
Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 2

=====[ Factorial ]====
Enter the number: 55
07:01:39.160 [main] INFO [class Calculator] - Factorial of [ 55.0 ]
Result is [ 1.2696403353658276E73 ]
Factorial of 55.0 = 1.2696403353658276E73

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 3

=====[ Natural Logarithm ]====
Enter the number: 16
07:01:53.310 [main] INFO [class Calculator] - Natural Logarithm of [ 16.0 ]
Result [ 2.772588722239781 ]
Natural Logarithm of 16.0 = 2.772588722239781

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
```

Fig 5.5.2. Calculator output: .jar file running on Docker image pulled from the Docker hub on our host machine

```
Terminal gokul@gokul-xubuntu: ~
=====[ Natural Logarithm ]====
Enter the number: 16
07:01:53.310 [main] INFO [class Calculator] - Natural Logarithm of [ 16.0 ]
Result [ 2.772588722239781 ]
Natural Logarithm of 16.0 = 2.772588722239781

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 4

=====[ Power ]====
Enter the first number:
2
Enter the second number :
3
07:01:58.632 [main] INFO [class Calculator] - Power [ 2.0 ]^[ 3.0 ]
Result [ 8.0 ]
Power : 2.0 ^ 3.0 = 8.0

Select one of the following operation
=> 1.Square Root
=> 2.Factorial
=> 3.Natural Logarithm
=> 4.Power function
=> 5.Exit
Enter your option :: 5

***** Exiting *****
root@304f650c55bd:/#
```

Fig 5.5.3. Calculator output: .jar file running on Docker image pulled from the Docker hub on our host machine


```

gokul@gokul-xubuntu:~$ docker exec -it calc-container bash
root@304f650c55bd:/# ls
ScientificCalculator_DevOps-1.0-SNAPSHOT-jar-with-dependencies.jar  boot      dev  home  lib64  mnt  proc  run  srv  tmp  var
bin                        calculator.log  etc  lib   media  opt  root  sbin  sys  usr
root@304f650c55bd:/# cat calculator.log
2022-04-18 06:59:31.802 [main] INFO [class Calculator] - Square Root of [ 12.0 ]
Result [ 3.4641016151377544 ]
2022-04-18 06:59:41.305 [main] INFO [class Calculator] - Power [ 5.0 ]^[ 3.0 ]
Result [ 125.0 ]
2022-04-18 07:01:25.321 [main] INFO [class Calculator] - Square Root of [ 81.0 ]
Result [ 9.0 ]
2022-04-18 07:01:39.160 [main] INFO [class Calculator] - Factorial of [ 55.0 ]
Result is [ 1.2696403353658276E73 ]
2022-04-18 07:01:53.310 [main] INFO [class Calculator] - Natural Logarithm of [ 16.0 ]
Result [ 2.772588722239781 ]
2022-04-18 07:01:58.632 [main] INFO [class Calculator] - Power [ 2.0 ]^[ 3.0 ]
Result [ 8.0 ]
root@304f650c55bd:/#

```

Fig 5.6. Viewing the logs saved in calculator.log file

Continuous Monitoring using Elasticsearch – Logstash – Kibana (ELK stack)

ELK stands for Elasticsearch – Logstash – Kibana which are three open source projects. Elastic search serves as a search engine. Logstash server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana helps users visualize data with charts and graphs in Elasticsearch. Sign Up for a 15 days trial on Elastic Cloud and create your first deployment.

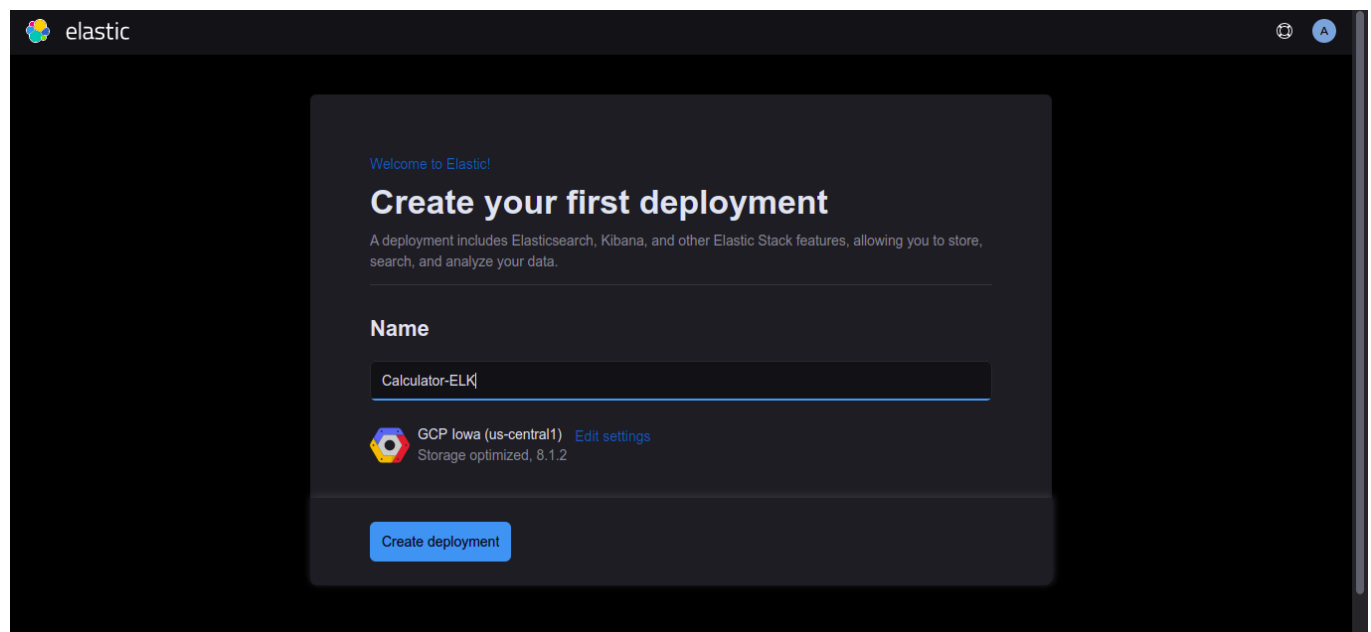


Fig 1. Creating deployment

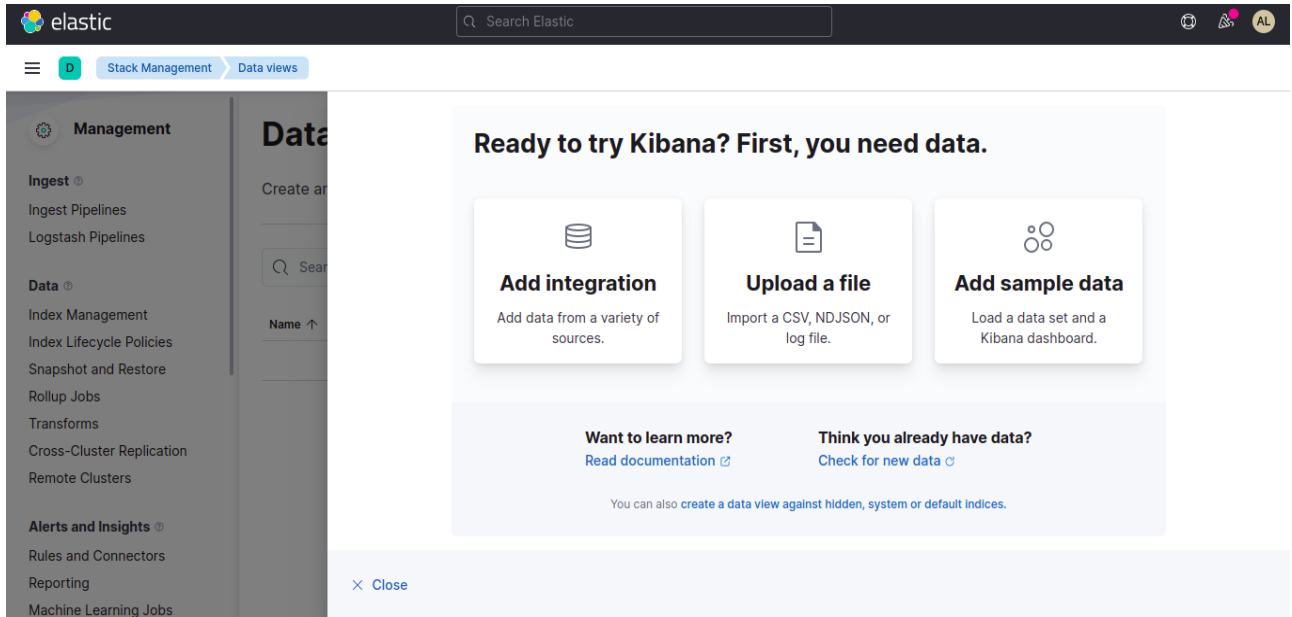


Fig 2. Deployment Created

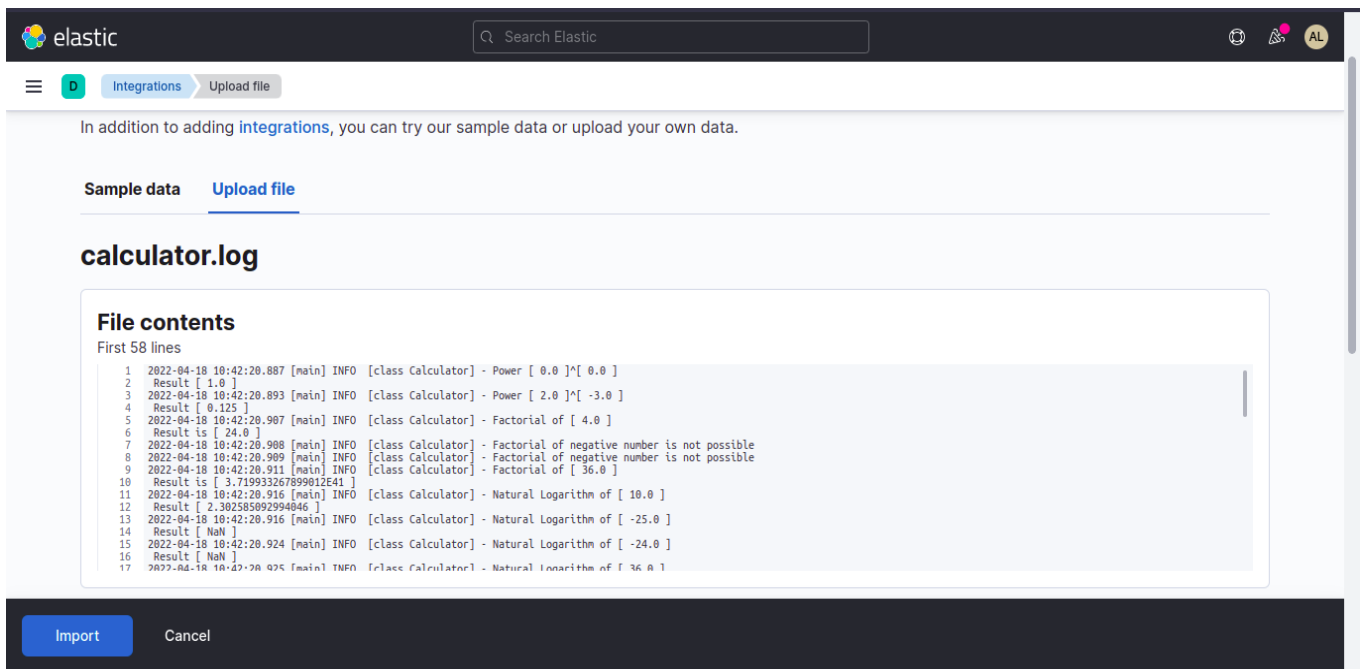


Fig 3. Upload the calculator.log file from target folder

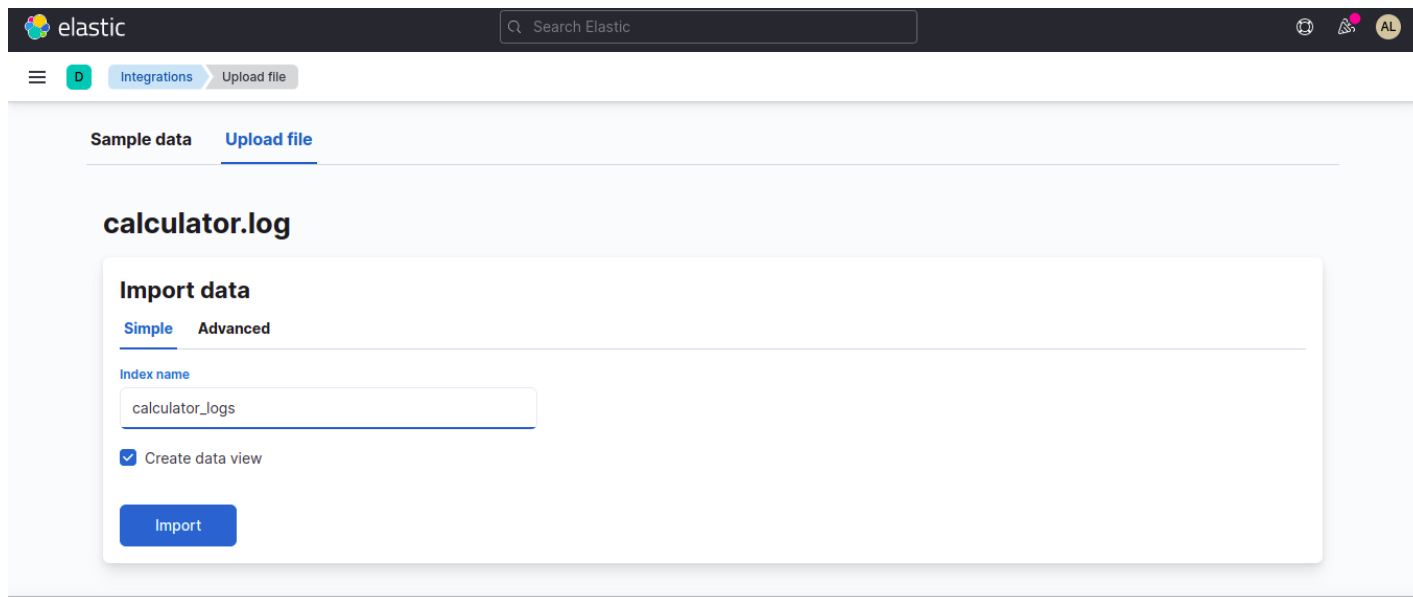


Fig 4. Assign an index name

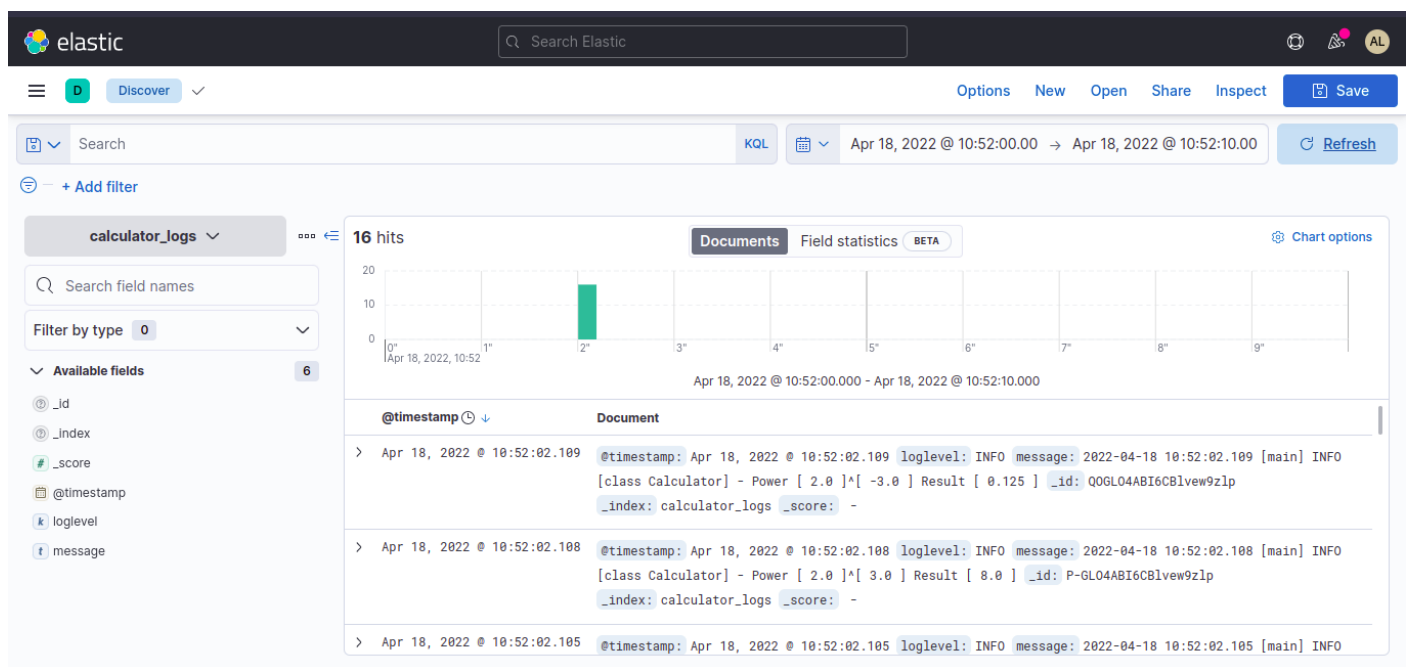


Fig 5. Visualization using Kibana