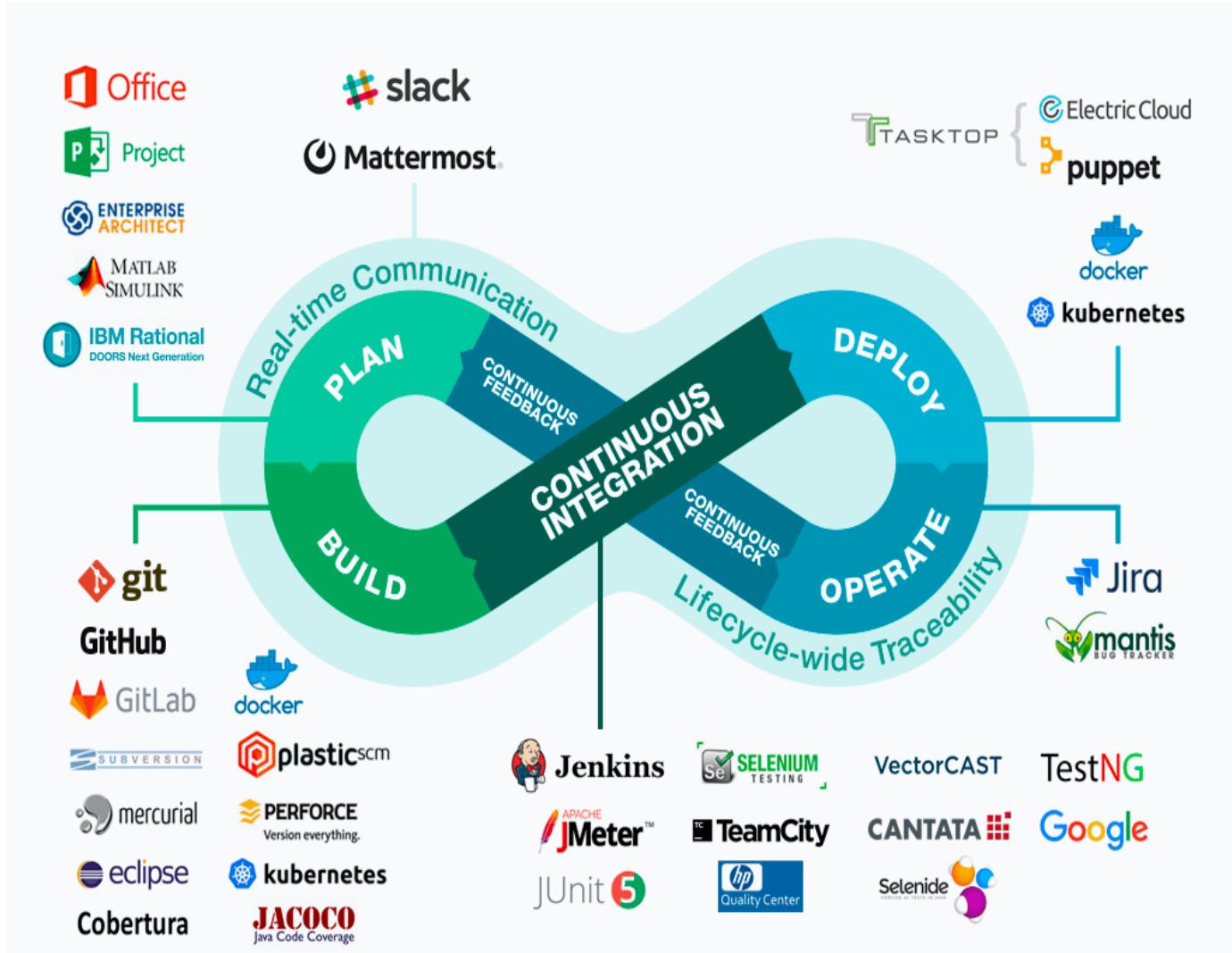


## BASIC SCIENTIFIC CALCULATOR- REPORT



By Mohd Asad Ansari  
(MT2020147)  
SOFTWARE PRODUCTION ENGINEERING

## **PROJECT STATEMENT**

Creating a terminal based basic calculator program with operations such calculating square root, factorial, natural log, and power, adding each functionality incrementally using devops tools like CircleCI, Docker, Git, and Github. The main objective of the project is to learn devops concept CI/CD which is achieved by creating a circlegit pipeline.

## **DevOps:**

DevOps is the combination of software development and operations. This practice allows a single team to handle the entire application lifecycle-from development, to testing, to deployment, and operations.

## **DevOps Pipeline :**

Here we are automating the integration chain which includes source control management, and continuous integration. The cycle would include pushing latest changes or code to git (GitHub), building code includes using tools, pushing build docker image onto docker hub, this part of continuous automation is called continuous integration.

Then we integrate circleci with docker and github here does the continuous integration. CircleCI pulls the pushed docker image from docker hub and deploys it on a docker container. Tools used includes

- GitHub - <https://github.com/asadiiitb/Dev-Ops-MiniProject>
- Docker image -  
[https://hub.docker.com/repository/docker/imasadansari/devops\\_calculator](https://hub.docker.com/repository/docker/imasadansari/devops_calculator)
- Continuous Integration – CircleCI
- Kibana (For Visualizing and Monitoring)

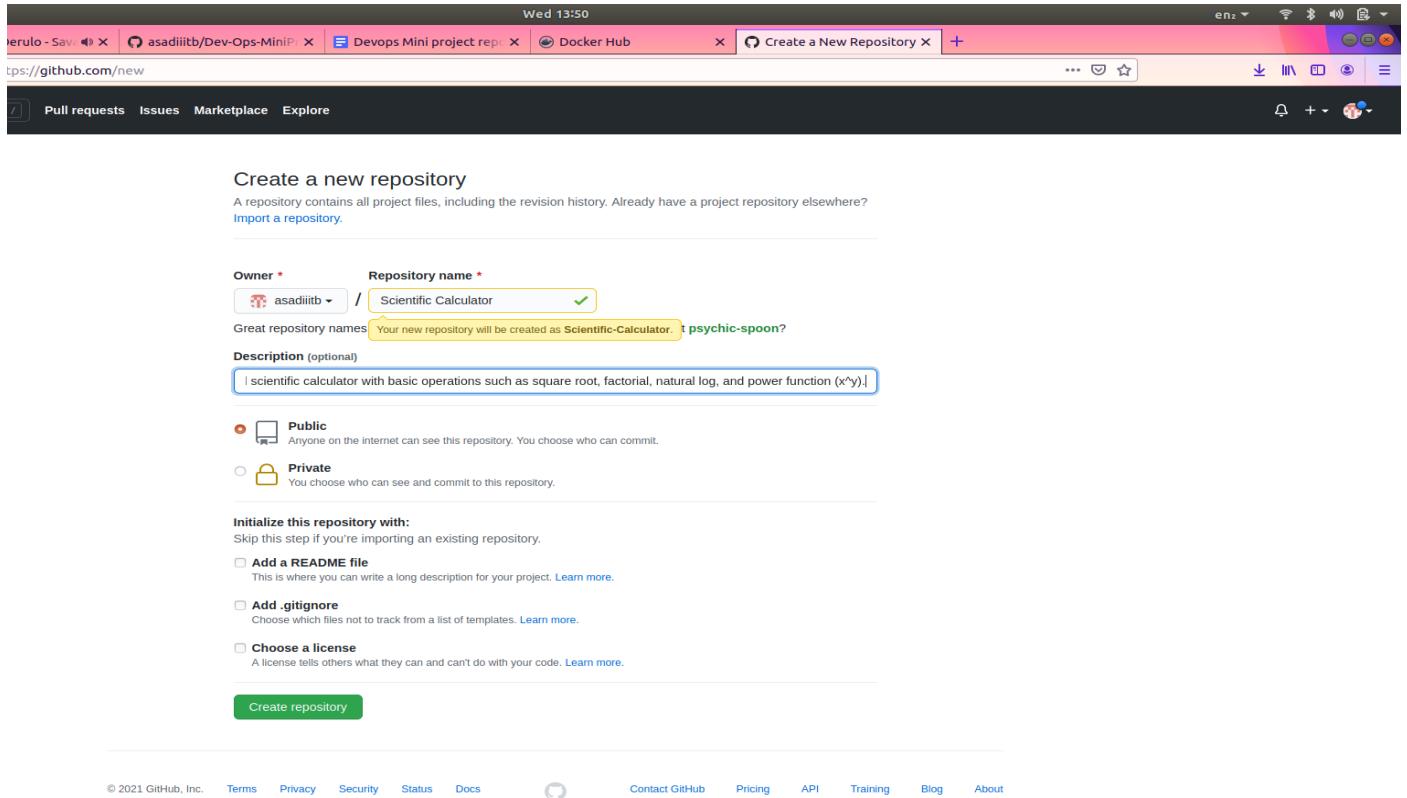


Fig 1 :- Creating a Repository on github

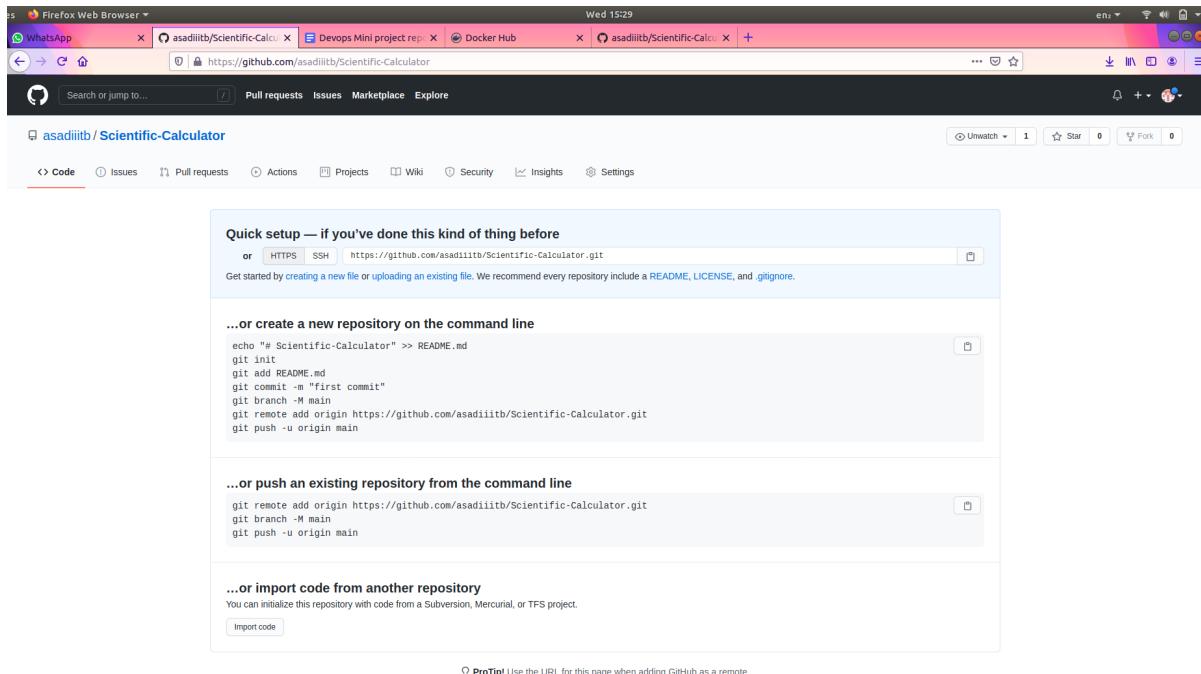
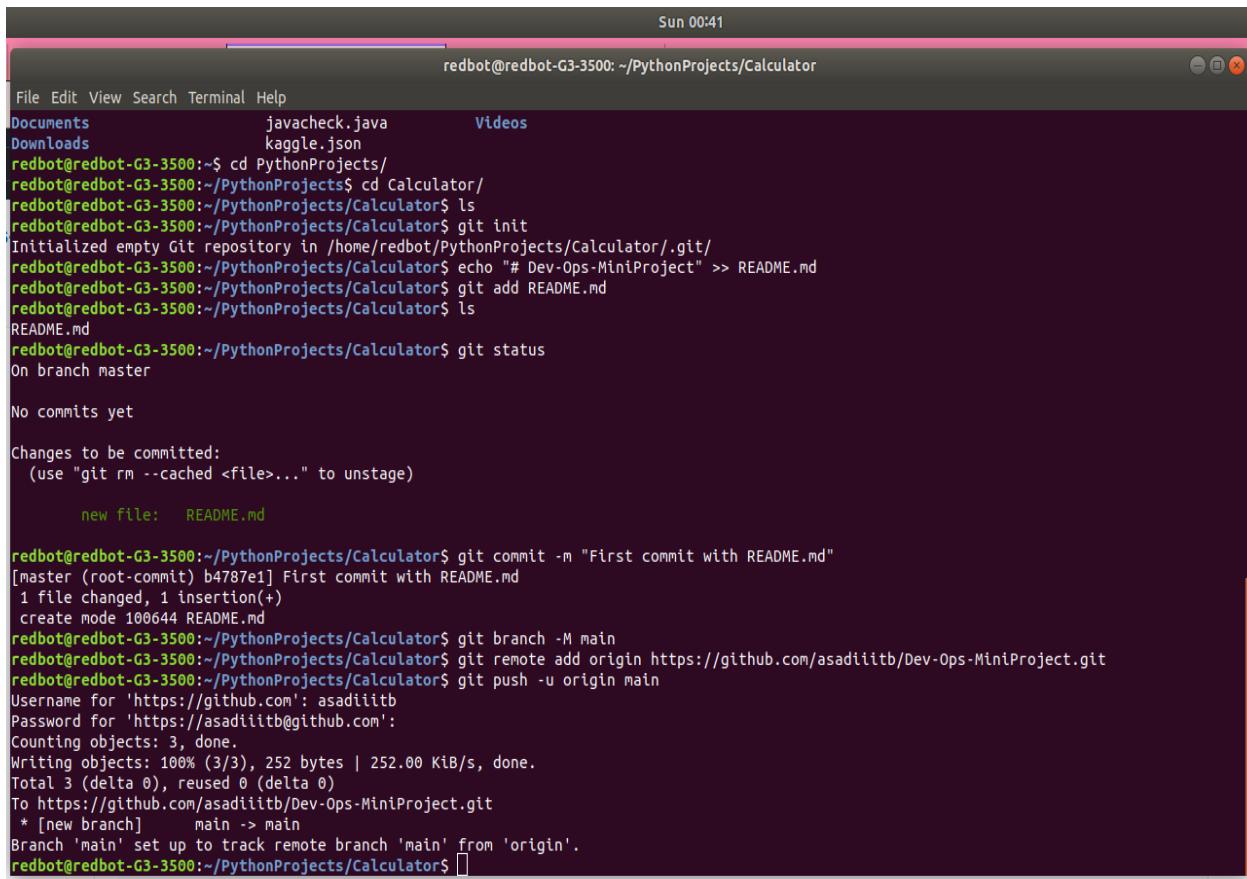


Fig 2:- Empty repository Created

## Source Control Management :-

Creating a new repository on to <https://www.github.com>. This includes adding repository name and description. The repository name should be unique for the signed in user. The similar is done and one such repository is created for the calculator dev-ops project and can be found at <https://github.com/asadiitb/Dev-Ops-MiniProject>. The SCM handles our code and is used to connect as input to CircleCI. Other SCM are gitlab, bitbucket.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, it says "Sun 00:41". The title bar reads "redbot@redbot-G3-3500: ~/PythonProjects/Calculator". The terminal content is as follows:

```
File Edit View Search Terminal Help
Documents          javacheck.java      Videos
Downloads         kaggle.json
redbot@redbot-G3-3500:~$ cd PythonProjects/
redbot@redbot-G3-3500:~/PythonProjects$ cd Calculator/
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ ls
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git init
Initialized empty Git repository in /home/redbot/PythonProjects/Calculator/.git/
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ echo "# Dev-Ops-MiniProject" >> README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git add README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ ls
README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file: README.md

redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git commit -m "First commit with README.md"
[master (root-commit) b4787e1] First commit with README.md
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git branch -M main
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git remote add origin https://github.com/asadiitb/Dev-Ops-MiniProject.git
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git push -u origin main
Username for 'https://github.com': asadiitb
Password for 'https://asadiitb@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 252 bytes | 252.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/asadiitb/Dev-Ops-MiniProject.git
 * [new branch]   main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
redbot@redbot-G3-3500:~/PythonProjects/Calculator$
```

Fig 3: Initializing SCM

## Installing Git and its Commands:

To Install Git use commands:

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

```
z c++-5.4.1
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git --version
git version 2.17.1
```

We need to tell git who are we, so the username will be used for pushing into repository

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git config --global user.name "Mohammed Asad Ansari"
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git config --global user.email "mohammadasad.ansari@iiitb.org"
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
```

Now if you have already developed the project and want to push to yours create repo, you can initiate the current directory –

- git init :- To initiate the current directory
- git remote add origin <git\_repo\_url>
- git push -f origin <branch\_name>

Or if you creating a new project I suggest doing a

- git clone <git\_repo\_url>

```
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git init
Initialized empty Git repository in /home/redbot/PythonProjects/Calculator/.git/
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ echo "# Dev-Ops-MiniProject" >> README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git add README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ ls
README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   README.md

redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git commit -m "First commit with README.md"
[master (root-commit) b4787e1] First commit with README.md
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git branch -M main
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git remote add origin https://github.com/asadiiitb/Dev-Ops-MiniProject.git
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ git push -u origin main
Username for 'https://github.com': asadiiitb
Password for 'https://asadiiitb@github.com':
Counting objects: 3, done.
Writing objects: 100% (3/3), 252 bytes | 252.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/asadiiitb/Dev-Ops-MiniProject.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
```

To push the code on to repository follow following commands

- git add <changed\_files\_path>
- git commit -m “Commit message name”
- git push (only if you are on master branch, otherwise It is suggested merging your branch with master first, resolving conflicts and then do git push).
- git status :- To check status of the tracked and untracked file.
- git add <File\_name> :- to add file to staging area.

## Setting Up Python and Working with Virtual Environment:

Here we are using python so we will install python using command:

- sudo apt-get install python3

Then to check whether it is installed or not check version using command:

- python3 -V

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip3 -V
pip 9.0.1 from /home/redbot/PythonProjects/Calculator/venv/lib/python3.6/site-packages (python 3.6)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip -V
pip 9.0.1 from /home/redbot/PythonProjects/Calculator/venv/lib/python3.6/site-packages (python 3.6)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ python3 -V
Python 3.6.9
```

Since we are working on the local system currently, and to cope up with the updates it is always advised and better to use a virtual environment for your project. Advantage of it will be that whenever in future if we want to use this project and we upgrade a certain library or package or it got deleted we don't need to cope up with environment we will save the libraries and their versions we are working with while developing this code, in a '**requirements.txt**' file. Using this file we can install those libraries anytime in future and work with it (Virtual Environment also has certain more advantages).

To manage software packages for Python, install **pip**, a tool that will install and manage libraries or modules to use in your projects.

- sudo apt install -y python3-pip

To check pip version use command:

- pip -V or pip3 -V

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip3 -V
pip 9.0.1 from /home/redbot/PythonProjects/Calculator/venv/lib/python3.6/site-packages (python 3.6)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip -V
pip 9.0.1 from /home/redbot/PythonProjects/Calculator/venv/lib/python3.6/site-packages (python 3.6)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ python3 -V
Python 3.6.9
```

To install any Package use :

- pip3 install <package name>

Installing Venv: Virtual environments enable you to have an isolated space on your server for Python projects. We'll use **venv**, part of the standard Python 3 library, which we can **install** by:

- sudo apt install -y python3-venv

Creating a Virtual Environment : You can create a new environment with the **pyvenv** command. Here, we can call new environment by any name, whatever you want:

- python3.6 -m venv <Environment Name>

Activate the environment using the command below, where my\_env is the name of your programming environment.

- Source <Environment Name>/bin/activate [This works for linux and mac users]

```
rebot@rebot-G3-3500:~/PythonProjects/Calculator$ 
rebot@rebot-G3-3500:~/PythonProjects/Calculator$ source venv/bin/activate
(venv) rebot@rebot-G3-3500:~/PythonProjects/Calculator$ 
```

Now, we can start working in this environment, isolated from our actual environment.

Note that we have installed external packages (pytest) and others might not have these installed. The other developers need the correct version of libraries and packages we have used in our current environment to replicate our work. This is where requirements.txt comes into play. If we provide a file listing the necessary packages, others can simply pip install -r requirements.txt and all the program's required packages and libraries will be downloaded, installed, and ready to go in one shot.

To save current environment use

- pip3 freeze > requirements.txt

```
(venv) rebot@rebot-G3-3500:~/PythonProjects/Calculator$ pip3 freeze
appdirs==1.4.4
attrs==20.3.0
coverage==5.5
distlib==0.3.1
filelock==3.0.12
flake8==3.8.4
importlib-metadata==3.7.2
importlib-resources==5.1.2
iniconfig==1.1.1
mccabe==0.6.1
packaging==20.9
pkg-resources==0.0.0
pluggy==0.13.1
py==1.10.0
pycodestyle==2.6.0
pyflakes==2.2.0
pyparsing==2.4.7
pytest==6.2.2
pytest-cov==2.11.1
six==1.15.0
toml==0.10.2
typing-extensions==3.7.4.3
virtualenv==20.4.2
zipp==3.4.1
```

Fig : Pip freeze output showing used libraries and corresponding versions

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip3 freeze > requirements.txt
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ ls
Calculator.log      Dockerfile          'Logs last 5 hours.png'  README.md      Scientific_Calculator.py      venv
Calculator_main.py   'Logs last 3 days.png'  __pycache__           requirements.txt  test_ScientificCalculator.py
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
```

Fig: requirements.txt now present in directory

To install the saved libraries use

- Pip3 install -r requirements.txt

```
LINKEPOINT python3 calculator_main.py
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip3 install -r requirements.txt
Requirement already satisfied: appdirs==1.4.4 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 1))
Requirement already satisfied: attrs==20.3.0 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 2))
Requirement already satisfied: coverage==5.5 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 3))
Requirement already satisfied: distlib==0.3.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 4))
Requirement already satisfied: filelock==3.0.12 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 5))
Requirement already satisfied: flake8==3.8.4 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 6))
Requirement already satisfied: importlib-metadata==3.7.2 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 7))
Requirement already satisfied: importlib-resources==5.1.2 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 8))
Requirement already satisfied: iniconfig==1.1.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 9))
Requirement already satisfied: mccabe==0.6.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 10))
Requirement already satisfied: packaging==20.9 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 11))
Requirement already satisfied: pluggy==0.13.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 12))
Requirement already satisfied: py==1.10.0 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 13))
Requirement already satisfied: pycodestyle==2.6.0 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 14))
Requirement already satisfied: pyflakes==2.2.0 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 15))
Requirement already satisfied: pyparsing==2.4.7 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 16))
Requirement already satisfied: pytest==6.2.2 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 17))
Requirement already satisfied: pytest-cov==2.11.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 18))
Requirement already satisfied: six==1.15.0 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 19))
Requirement already satisfied: toml==0.10.2 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 20))
Requirement already satisfied: typing-extensions==3.7.4.3 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 21))
Requirement already satisfied: virtualenv==20.4.2 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 22))
Requirement already satisfied: zipp==3.4.1 in ./venv/lib/python3.6/site-packages (from -r requirements.txt (line 23))
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
```

Note: Since I'm already in my virtual environment it says the requirements are already satisfied but for others it will start downloading that package and installing it if that package is not already installed.

To exit the virtual Environment use command:

- deactivate

```
Requirement already satisfied: backports-weakref in ./venv/lib/python3.6/site-packages (from -r requirements.txt)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ deactivate
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 
```

Since this will be a python terminal-based calculator, I am using ubuntu's editor, a text editor to write source code and test cases. First, I created a python file called Scientific\_Calculator.py which contains the main functions of the calculator. The reason for this is that while testing, we will not have to input from stdin and there will not be errors or warnings during testing.

The screenshot shows a code editor window with the following details:

- Title Bar:** Open ▾
- File Path:** ~/PythonProjects/Calculator
- File Name:** \*Scientific\_Calculator.py
- Code Content:** The code defines several mathematical functions using Python's math and logging modules. It includes:
  - square\_root(num):** Returns the square root of a number, handling negative numbers by printing an error message.
  - factorial(num):** Returns the factorial of a number. It handles decimal numbers by rounding them to integers before calculating the factorial, and it prevents ValueError for negative numbers.
  - natural\_log(num):** Returns the natural logarithm of a number. It prevents taking the log of zero or negative numbers.
  - power(num1, num2):** Returns the power of one number to another.

Fig 4: Scientific\_Calculator functions

Then in a separate python file, we write the driving code.

```

1 import Scientific_Calculator
2 import os
3
4 os.system('cls' if os.name == 'nt' else 'clear')
5
6 while True:
7     print("Select Operation.")
8     print("1: Sqaure Root - √x")
9     print("2: Factorial - x!")
10    print("3: Natural Log - ln(x)")
11    print("4: Power - x^y")
12    print("e: Exit")
13
14    operation = input("Enter your choice : ")
15
16    if operation in ('1', '2', '3', '4'):
17        num = float(input("Enter the number: "))
18
19        if operation == '1':
20            print("Answer : ", Scientific_Calculator.square_root(num))
21
22        elif operation == '2':
23            print("Answer : ", Scientific_Calculator.factorial(num))
24
25        elif operation == '3':
26            print("Answer : ", Scientific_Calculator.natural_log(num))
27
28        elif operation == '4':
29            exp = float(input("Enter the power : "))
30            print("Answer : ", Scientific_Calculator.power(num, exp))
31            # Not using Clear screen here as it will also clear the result as well
32            # os.system('cls' if os.name == 'nt' else 'clear')
33
34    elif operation == 'e' or 'E':
35        break
36
37    else:
38        print("Please Enter a valid Input")
39    print("\n")
40

```

Fig 5 : Code of Caluclator\_main.py which uses the Scientific Calculator functions

## **Testing:**

For unit tests in python, we use pytest. A unit test is designed to check a single function, or unit, of code. Pytest supports execution of unittest test cases. The real advantage of pytest comes by writing pytest test cases. pytest test cases are a series of functions in a Python file starting with the name `test_`. To install pytest, we simply do

- pip install -U pytest

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip3 install -U pytest
Requirement already up-to-date: pytest in ./venv/lib/python3.6/site-packages
Requirement already up-to-date: iniconfig in ./venv/lib/python3.6/site-packages (from pytest)
Requirement already up-to-date: toml in ./venv/lib/python3.6/site-packages (from pytest)
Requirement already up-to-date: attrs>=19.2.0 in ./venv/lib/python3.6/site-packages (from pytest)
Collecting importlib-metadata>=0.12; python_version < "3.8" (from pytest)
  Downloading https://files.pythonhosted.org/packages/80/5d/0bbcab2b16e01313cf0343167d4cfb90f6fade747cd4d10d368094b2883a/importlib_metadata-3.7.3-py3-none-any.whl
Requirement already up-to-date: packaging in ./venv/lib/python3.6/site-packages (from pytest)
Requirement already up-to-date: py>=1.8.2 in ./venv/lib/python3.6/site-packages (from pytest)
Requirement already up-to-date: pluggy<1.0.0a1,>=0.12 in ./venv/lib/python3.6/site-packages (from pytest)
Requirement already up-to-date: typing-extensions>=3.6.4; python_version < "3.8" in ./venv/lib/python3.6/site-packages (from importlib-metadata>=0.12; python_version < "3.8">->pytest)
Requirement already up-to-date: zipp>=0.5 in ./venv/lib/python3.6/site-packages (from importlib-metadata>=0.12; python_version < "3.8">->pytest)
Requirement already up-to-date: pyParsing>=2.0.2 in ./venv/lib/python3.6/site-packages (from packaging->pytest)
Installing collected packages: importlib-metadata
  Found existing installation: importlib-metadata 3.7.2
    Uninstalling importlib-metadata-3.7.2:
      Successfully uninstalled importlib-metadata-3.7.2
Successfully installed importlib-metadata-3.7.3
```

pytest has some other great features:

- Support for the built-in assert statement instead of using special self.assert\*() methods
- Support for filtering for test cases
- Ability to rerun from the last failing test
- An ecosystem of hundreds of plugins to extend the functionality

A standard practice that goes hand in hand with testing is calculating code coverage. Code coverage is the percentage of source code that is “covered” by your tests. pytest has an extension, pytest-cov, that helps us understand the code coverage. To install this, we do pip install pytest-cov. Now we can run the tests.

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pip install pytest-cov
Requirement already satisfied: pytest-cov in ./venv/lib/python3.6/site-packages
Requirement already satisfied: pytest>=4.6 in ./venv/lib/python3.6/site-packages (from pytest-cov)
Requirement already satisfied: coverage>=5.2.1 in ./venv/lib/python3.6/site-packages (from pytest-cov)
Requirement already satisfied: importlib-metadata>=0.12; python_version < "3.8" in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: pluggy<1.0.0a1,>=0.12 in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: toml in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: iniconfig in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: packaging in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: attrs>=19.2.0 in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: py>=1.8.2 in ./venv/lib/python3.6/site-packages (from pytest>=4.6->pytest-cov)
Requirement already satisfied: typing-extensions>=3.6.4; python_version < "3.8" in ./venv/lib/python3.6/site-packages (from importlib-metadata>=0.12; python_version < "3.8">->pytest>=4.6->pytest-cov)
Requirement already satisfied: zipp>=0.5 in ./venv/lib/python3.6/site-packages (from importlib-metadata>=0.12; python_version < "3.8">->pytest>=4.6->pytest-cov)
Requirement already satisfied: pyParsing>=2.0.2 in ./venv/lib/python3.6/site-packages (from packaging->pytest>=4.6->pytest-cov)
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$
```

Writing the TestSum test case example for pytest would look like this, It will consist of both positive to false negative results as well:

```

1 """
2 Unit tests for the calculator library
3 """
4 # Import the Scientific Calculator Function File
5
6 import Scientific_Calculator
7
8
9 class TestCalculator:
10
11     def test_sqrt(self):
12         assert Scientific_Calculator.square_root(196) == 14.0
13         assert Scientific_Calculator.square_root(0) == 0.0
14         assert Scientific_Calculator.square_root(-9) == -9.0
15         assert Scientific_Calculator.square_root(1) == 1
16         assert Scientific_Calculator.square_root(3) == 1.7320508075688772
17
18     def test_factorial(self):
19         assert Scientific_Calculator.factorial(7) == 5040
20         assert Scientific_Calculator.factorial(5) == 120
21         assert Scientific_Calculator.factorial(0) == 1
22         assert Scientific_Calculator.factorial(1) == 1
23         assert Scientific_Calculator.factorial(-5) == 0.0
24         assert Scientific_Calculator.factorial(-5.5) == 0.0
25         assert Scientific_Calculator.factorial(5.5) == 6
26
27     def test_log(self):
28         assert Scientific_Calculator.natural_log(0) == 0
29         assert Scientific_Calculator.natural_log(-1) == 0
30         assert Scientific_Calculator.natural_log(-10) == 0
31         assert Scientific_Calculator.natural_log(2.718281828459045) == 1.0
32         assert Scientific_Calculator.natural_log(20) == 2.995732273553991
33
34     def test_power(self):
35         assert Scientific_Calculator.power(2,9) == 512
36         assert Scientific_Calculator.power(0, 12) == 0
37         assert Scientific_Calculator.power(20, 0) == 1
38         assert Scientific_Calculator.power(20, 1) == 20.0
39         assert Scientific_Calculator.power(2, 3) == 8.0
40         assert Scientific_Calculator.power(-2,3) == -8.0
41         assert Scientific_Calculator.power(2,-3) == 0.125
42         assert Scientific_Calculator.power(-2, -3) == -0.125
43         assert Scientific_Calculator.power(-2, 4) == 16.0
44         assert Scientific_Calculator.power(1, 23) == 1.0
45

```

Fig 6: Unit Test cases for scientific calculator

We are using the assert keyword to write our test cases. It lets us test the condition whether both sides give equal results (or same), else it will raise an AssertionError.

After creating the unit test cases, it's time to check whether our code passes out those test cases or not. To check run command :

- pytest -v --cov=<Name of main file> [Like i runned pytest -v --cov=Scientific\_Calculator as ‘Scientific\_Calculator.py’ is name of file which holds the functions of my calculator].

```
=====
short test summary info =====
FAILED test_ScientificCalculator.py::TestCalculator::test_sqrt - UnboundLocalError: local variable 'x' referenced before assignment
FAILED test_ScientificCalculator.py::TestCalculator::test_factorial - NameError: name 'x' is not defined
FAILED test_ScientificCalculator.py::TestCalculator::test_log - NameError: name 'x' is not defined
===== 3 failed, 1 passed in 0.80s =====
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ pytest -v --cov
=====
test session starts =====
platform linux -- Python 3.6.9, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- /home/redbot/PythonProjects/Calculator/venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/redbot/PythonProjects/Calculator
plugins: cov-2.11.1
collected 4 items

test_ScientificCalculator.py::TestCalculator::test_sqrt PASSED
test_ScientificCalculator.py::TestCalculator::test_factorial PASSED
test_ScientificCalculator.py::TestCalculator::test_log PASSED
test_ScientificCalculator.py::TestCalculator::test_power PASSED
[ 25%]
[ 50%]
[ 75%]
[100%]

..... coverage: platform linux, python 3.6.9-final-0 .....
```

Fig 7: Pytest results

## Continuous Integration:

One of the most important part of Project. Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day. In this development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. We can use CircleCI for Continuous Integration or continue till Continuous Delivery using the pipeline project.

The steps we did above were all manual. What if we want several developers to add hundreds of features every day, doing all these steps manually? Think about the time we would lose in this labor. And enter Continuous Integration. The idea is to automate the build. Now python is an interpreted language so it's 'builds' revolve around test execution, rather than compilation.

The point of this step is to have everyone working on a known stable base. When the build is automated, we are encouraged to commit frequently, usually multiple times per day. It allows people to quickly find out about changes and notice if there's a conflict between two developers. If there are numerous small changes instead of a few massive updates, it's much easier to locate where the error originated. It will also encourage you to break your work down into smaller chunks, which is easier to track and test.

Step 1: To use CircleCI we need to go to : <https://circleci.com/> , then create an account there.

Step 2: Connect CircleCI account to Github or Bitbucket wherever you store your code.

Step 3: Then go to Projects > Setup Projects > setup “config.yml”.

Now, Whenever we push change code in the github repository it will get build and tested.

The screenshot shows the CircleCI 'Projects' dashboard. At the top, there's a header with a back arrow, forward arrow, refresh icon, and a lock icon indicating a secure connection to <https://app.circleci.com/projects/project-dashboard/github/asadiiitb/>. Below the header is a sidebar with various icons for navigation. The main area is titled 'Projects' and contains a section for setting up new projects or following existing ones. A search bar labeled 'Repo name' is present. A 'Follow All' button and a three-dot menu button are located in the top right of this section. The main list displays five GitHub repositories:

- Dev-Ops-MiniProject**: Includes a 'Unfollow Project' button.
- Scientific-Calculator**: Includes a 'Set Up Project' button.
- ScientificCalculator**: Includes a 'Unfollow Project' button.
- CalculatorDevops**: Includes a 'Set Up Project' button.
- September11TutLec**: Includes a 'Set Up Project' button.

Fig 8 : CircleCI projects page of your account (already connected with Github or Bitbucket)  
Before setting up a project in CircleCI we need to write the **config.yml** file in which we write the steps and commands instructing CircleCI server how to build and test our code.

1. The first step is to create a .circleci folder in our project directory. Inside this .circleci folder, we create a config.yml file. So its structure will be like ./Parent Repo/.circleci/config.yml
2. Now in the config.yml file, we need to define or state the following steps.

```

1 # Python CircleCI 2.1 configuration file
2 version: 2.1
3 jobs:
4   build:
5     # executor: docker-publisher
6     environment:
7       IMAGE_NAME: devops_calculator
8     docker:
9       - image: circleci/python:3.7
10    working_directory: ~/repo
11
12  steps:
13    # Step 1: obtain repo from GitHub
14    - checkout
15    # Step 2: setup remote docker engine
16    - setup_remote_docker
17    # Step 3: create virtual env and install dependencies
18    - run:
19      name: install dependencies
20      command: |
21        python3 -m venv v_env
22        . v_env/bin/activate
23        pip install -r requirements.txt
24    # Step 3: run linter and tests
25    - run:
26      name: run tests
27      command: |
28        . v_env/bin/activate
29        pytest -v --cov=Scientific_Calculator
30    # Step 4: Build Docker Image
31    - run:
32      name: Build and Push Docker Image
33      command: |
34        docker build -t $IMAGE_NAME:latest .
35        echo "$DOCKERHUB_PASSWORD" | docker login -u "$DOCKERHUB_USERNAME" --password-stdin
36        docker tag $IMAGE_NAME $DOCKERHUB_USERNAME/$IMAGE_NAME:latest
37        docker push $DOCKERHUB_USERNAME/$IMAGE_NAME:latest
38

```

Fig 9: Config.yml file code

Ignoring the docker part for now, we see the steps as:

- Checkout - This checks our github repo for the latest version of our code.
- Install dependencies – This creates a virtual environment and installs the packages from requirements.txt which we used in our local machine.
- Run tests – This runs unit tests in the similar fashion as we did it locally.

Now, we add this .circleci folder to our github repository. Back to setting up our CircleCI server.

We select our project repo and set up the project. Once we click on set up project, CircleCI automatically fetches the config.yml from the .circleci folder in our github repository.

The screenshot shows the CircleCI configuration interface for a project named "DevOps-Mini-Project". At the top, there is a message: "We detected an existing config in your project at `.circleci/config.yml`. Click 'Start Building' below to kickoff your pipeline." Below this is a "Start Building" button. The main area contains the content of `config.yml`:

```

1  # Python CircleCI 2.0 configuration file
2  version: 2
3  jobs:
4    build:
5      docker:
6        - image: circleci/python:3.7
7
8      working_directory: ~/repo
9
10     steps:
11       # Step 1: obtain repo from GitHub
12       - checkout
13       # Step 2: create virtual env and install dependencies
14       - run:
15         name: install dependencies
16         command: |
17           python3 -m venv venv

```

At the bottom, a green bar indicates that `config.yml` is valid and ready to commit, with a "show in JSON" link.

Fig 10: CircleCI fetched config.yml, presenting it and ready to start building

Now we will click on Start Building Button

The screenshot shows the CircleCI pipeline history for the "Dev-Ops-MiniProject" branch. The sidebar on the left shows the user profile and navigation links. The main area lists the following builds:

- Build 12: Success, 17s ago
- Build 11: Success, 19s ago
- Build 10: Success, 15s ago
- Build 9: Failed, 24s ago
- Build 8: Failed, 24s ago
- Build 7: Failed, 12s ago
- Build 6: Failed, 8s ago
- Build 5: Failed, 12s ago

Each build entry includes a "Details" link and a "Logs" link.

Fig 11: CircleCI builds success and fails pipeline

As it can be seen my certain number of builds failed as there were some virtual environment naming conflicts, also there were build fails because of spacing and indentation, as it is sensitive to spacing and indentation. But finally at the end it all worked

## Workflow of CircleCI server in this successful build:

The screenshot shows the CircleCI web interface for a successful build. At the top, it displays the URL: https://app.circleci.com/pipelines/github/asadiiitb/Dev-Ops-MiniProject/10/workflows/4646aff8-bcd1-4827-a968-a36fc50a0d68/jobs/10. The main header includes tabs for Dashboard, Project, Branch, Workflow, Job, and a success status indicator. Below the header, the pipeline navigation shows All Pipelines > Dev-Ops-MiniProject > main > workflow > build (10). The build summary card provides details like Duration / Finished (21s / 4d ago), Queued (0s), Executor (Docker Medium), Branch (main), Commit (a0f2a47), and Author & Message (Updated config.yml). The build steps section lists five steps: Spin up environment, Preparing environment variables, Checkout code, install dependencies, and run tests, each marked as successful with green checkmarks and execution times (14s, 0s, 0s, 4s, 0s) and download icons.

Fig 12: All steps of CircleCI build and test

This screenshot focuses on the 'Checkout code' step of the build process. The URL is the same as Fig 12: https://app.circleci.com/pipelines/github/asadiiitb/Dev-Ops-MiniProject/10/workflows/4646aff8-bcd1-4827-a968-a36fc50a0d68/jobs/10. The step details show the command being run: 'Using environment variables from project settings and/or contexts: CIRCLE\_JOB="REDACTED"'. It also notes that redacted variables will be masked in run step output. The step status is marked as successful with a green checkmark and a 0s execution time. The terminal output shows the git fetch and clone process, including the RSA host key addition and cloning into the repository. The final message indicates the branch is up to date with 'origin/main'.

Fig 13: Checksout - pulls the latest code from the Github repo

The screenshot shows a CircleCI pipeline interface. On the left, there's a sidebar with navigation links: Dashboard, Projects, Insights, Organization Settings, Plan, Status (set to MAINTENANCE), Help, and a GitHub icon. The main area displays a terminal window with the following output:

```

remote: Compressing objects: 100% (1345/1345), done.
remote: Total 1559 (delta 157), reused 1551 (delta 154), pack-reused 0
Receiving objects: 100% (1559/1559), 10.35 MiB | 49.28 MiB/s, done.
Resolving deltas: 100% (157/157), done.
Checking out abf2a47...
Reset branch 'main'
Your branch is up to date with 'origin/main'.
HEAD is now at abf2a47 Updated config.yml

```

Below this, a section titled "Install dependencies" shows the progress of pip installations:

```

Collecting pluggy==0.13.1
  Downloading pluggy-0.13.1-py2.py3-none-any.whl (18 kB)
Collecting py==1.10.0
  Downloading py-1.10.0-py2.py3-none-any.whl (97 kB)
Collecting pycodestyle==2.6.0
  Downloading pycodestyle-2.6.0-py2.py3-none-any.whl (41 kB)
Collecting pyflakes==2.2.0
  Downloading pyflakes-2.2.0-py2.py3-none-any.whl (66 kB)
Collecting pyrsarang==2.4.7
  Downloading pyrsarang-2.4.7-py2.py3-none-any.whl (67 kB)
Collecting pytest==6.2.2
  Downloading pytest-6.2.2-py3-none-any.whl (280 kB)
Collecting pytest-cov==2.11.1
  Downloading pytest_cov-2.11.1-py2.py3-none-any.whl (20 kB)
Collecting six==1.15.0
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting tomli==0.10.2
  Downloading tomli-0.10.2-py2.py3-none-any.whl (16 kB)
Collecting typing_extensions==3.7.4.3
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting virtualenv==20.4.2
  Downloading virtualenv-20.4.2-py2.py3-none-any.whl (7.2 MB)
Collecting zipp==3.4.1
  Downloading zipp-3.4.1-py3-none-any.whl (5.2 kB)
Installing collected packages: appdirs, attrs, coverage, distlib, filelock, typing-extensions, zipp, importlib-metadata, pycodestyle, pyflakes, mccabe, flake8, importlib-resources
Successfully installed appdirs-1.4.4 attrs-20.3.0 coverage-5.5 distlib-0.3.1 filelock-3.0.12 flake8-3.8.4 importlib-metadata-3.7.2 importlib-resources-5.1.2 inconfig-1.1.1 mccabe
WARNING: You are using pip version 20.1.1; however, version 21.0.1 is available.
You should consider upgrading via the '/home/circleci/repo/v_env/bin/python3 -m pip install --upgrade pip' command.
CircleCI received exit code 0

```

At the bottom, a section titled "run tests" shows the command and its execution:

```

#!/bin/bash -eo pipefail
. v_env/bin/activate
pytest -v --cov=Scientific_Calculator

===== test session starts =====
platform linux -- Python 3.7.10, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- /home/circleci/repo/v_env/bin/python3
cachedir: ./pytest_cache
rootdir: /home/circleci/repo
plugins: cov-2.11.1
collecting ... collected 4 items

test_Scientificcalculator.py::TestCalculator::test_sqrt PASSED [ 25%]
test_Scientificcalculator.py::TestCalculator::test_factorial PASSED [ 50%]
test_Scientificcalculator.py::TestCalculator::test_log PASSED [ 75%]
test_Scientificcalculator.py::TestCalculator::test_power PASSED [100%]

----- coverage: platform linux, python 3.7.10-final-0 -----
Name           Stmts   Miss  Cover
-----
Scientific_Calculator.py      25      0   100%
TOTAL                      25      0   100%
===== 4 passed in 0.05s =====
CircleCI received exit code 0

```

Fig 14: Installs dependencies, with the help of requirements.txt

This screenshot shows the continuation of the CircleCI pipeline from Fig 14. The terminal output continues from the previous step:

```

Collecting pytest-cov==2.11.1
  Downloading pytest_cov-2.11.1-py2.py3-none-any.whl (20 kB)
Collecting six==1.15.0
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting tomli==0.10.2
  Downloading tomli-0.10.2-py2.py3-none-any.whl (16 kB)
Collecting typing_extensions==3.7.4.3
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting virtualenv==20.4.2
  Downloading virtualenv-20.4.2-py2.py3-none-any.whl (7.2 MB)
Collecting zipp==3.4.1
  Downloading zipp-3.4.1-py3-none-any.whl (5.2 kB)
Installing collected packages: appdirs, attrs, coverage, distlib, filelock, typing-extensions, zipp, importlib-metadata, pycodestyle, pyflakes, mccabe, flake8, importlib-resources
Successfully installed appdirs-1.4.4 attrs-20.3.0 coverage-5.5 distlib-0.3.1 filelock-3.0.12 flake8-3.8.4 importlib-metadata-3.7.2 importlib-resources-5.1.2 inconfig-1.1.1 mccabe
WARNING: You are using pip version 20.1.1; however, version 21.0.1 is available.
You should consider upgrading via the '/home/circleci/repo/v_env/bin/python3 -m pip install --upgrade pip' command.
CircleCI received exit code 0

```

Below this, the "run tests" section shows the command and its execution:

```

#!/bin/bash -eo pipefail
. v_env/bin/activate
pytest -v --cov=Scientific_Calculator

===== test session starts =====
platform linux -- Python 3.7.10, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- /home/circleci/repo/v_env/bin/python3
cachedir: ./pytest_cache
rootdir: /home/circleci/repo
plugins: cov-2.11.1
collecting ... collected 4 items

test_Scientificcalculator.py::TestCalculator::test_sqrt PASSED [ 25%]
test_Scientificcalculator.py::TestCalculator::test_factorial PASSED [ 50%]
test_Scientificcalculator.py::TestCalculator::test_log PASSED [ 75%]
test_Scientificcalculator.py::TestCalculator::test_power PASSED [100%]

----- coverage: platform linux, python 3.7.10-final-0 -----
Name           Stmts   Miss  Cover
-----
Scientific_Calculator.py      25      0   100%
TOTAL                      25      0   100%
===== 4 passed in 0.05s =====
CircleCI received exit code 0

```

Fig 15: Runs Unit tests

One of features of CircleCI, as we push the code or made changes in repository it checks them and builds, and whatever is the result it sends the report / result to the developer via email, so developer dont need to look at the pipeline instantly and wait for result.

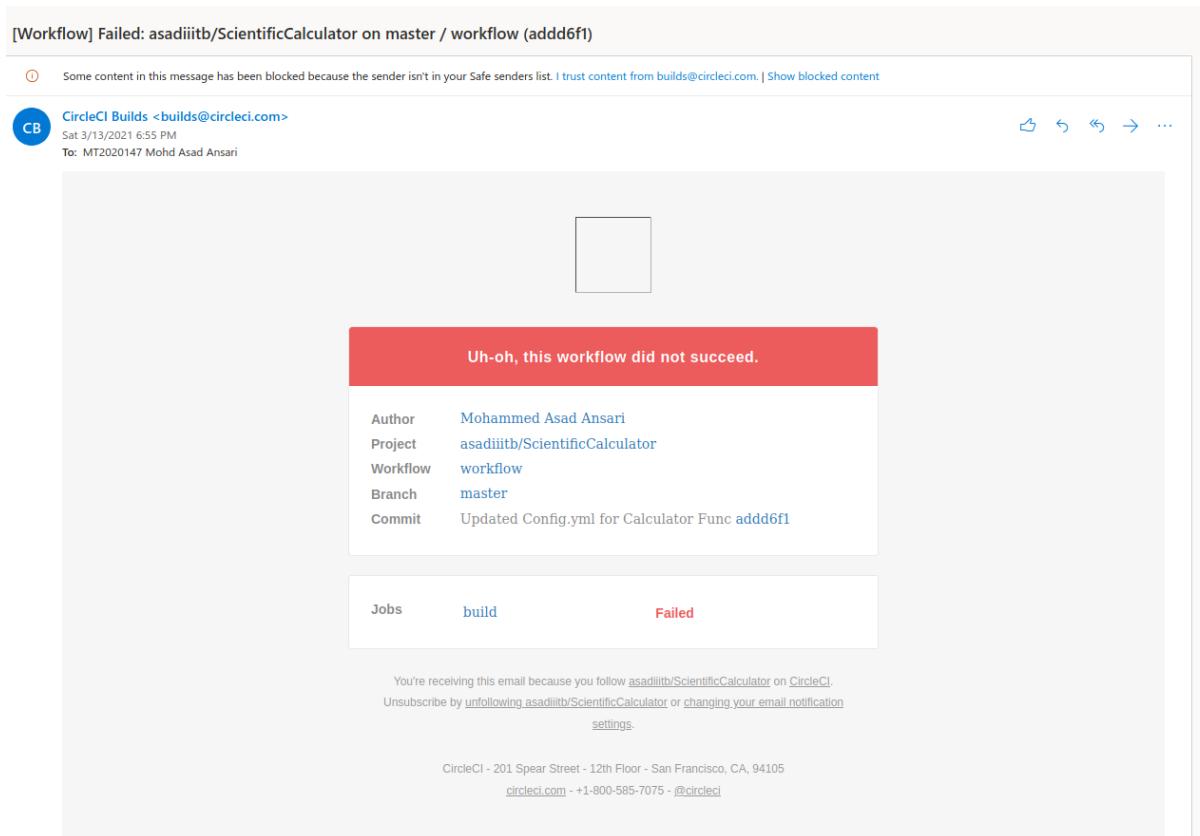


Fig 16: CircleCI build fail notification via mail

## Docker:

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

## The Docker platform:

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share

containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

## **Installing Docker:**

First things first, let's install docker engine on our local machine. Since I was using Ubuntu at the moment of this project, I followed this guide [How to Install Docker on Ubuntu](#).

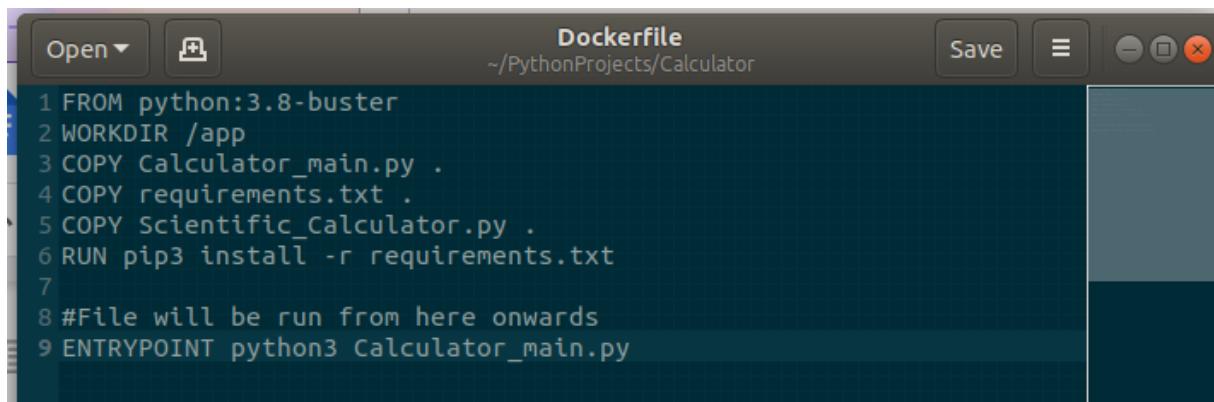
{ Keep in mind docker image need to be run in su mode so we need to use **sudo** with all the commands }.

Need of Docker : We simply want to create a docker image of our calculator application which can run on any Linux machine (Note that since I am working on Linux, the docker image created will be a Linux based image and cannot be run on Windows ).

## **DOCKERFILE:**

To build our docker image, we first need to make a Dockerfile in our project's top level directory. A **Dockerfile** is a **text file** written in an easy-to-understand syntax that includes the instructions to build a Docker image. It specifies the operating system that will underlie the container, along with the languages, environmental variables, file locations, network ports, and other components it needs—and, of course, what the container will be doing once we run it.

Let's write the docker file for our calculator application:



The screenshot shows a code editor window with a dark theme. The title bar says "Dockerfile" and the path is "~/PythonProjects/Calculator". The editor contains the following Dockerfile code:

```
1 FROM python:3.8-buster
2 WORKDIR /app
3 COPY Calculator_main.py .
4 COPY requirements.txt .
5 COPY Scientific_Calculator.py .
6 RUN pip3 install -r requirements.txt
7
8 #File will be run from here onwards
9 ENTRYPOINT python3 Calculator_main.py
```

Fig 17 : Dockerfile

{Keep in mind u need to save this file as “Dockerfile”, I saved it in Camel Case and wasted 2-3 hrs to solve the error before realizing}

Since, it's easy to understand let's try to get what each line means:

1. **FROM python:3.8-buster** - This line specifies that from the python repository at Docker Hub, use the version 3.8. There can be multiple versions available for an image and we are free to choose at our own will.
2. **WORKDIR /app** - Now, we create a working directory. This instructs Docker to use this path as the default location for all subsequent commands. By doing this, we do not have to type out full file paths but can use relative paths based on the working directory.
3. **COPY Calculator\_main.py .** - The COPY command takes two parameters. The first parameter tells Docker what file(s) you would like to copy into the image. The second parameter tells Docker where you want that file(s) to be copied to. We will copy the Calculator\_main.py file into our working directory /app. Similarly, we do it for our remaining .py files and requirements.txt.
4. **RUN pip3 install -r requirements.txt** - Next, we can use the RUN command to execute the command pip3 install. This works the same as if we were running pip3 install locally on our machine, but this time the modules are installed into the image.
5. **ENTRYPOINT python3 Calculator\_main.py** - This tells the Docker what to do with the image. In our case, we want to run our Calculator\_main.py using the command **python3 Calculator\_main.py**.

## **DOCKER IMAGE BUILD:**

With our Dockerfile ready, we are ready to build our image. By command:

- **sudo docker build -t <Name/tag of image generated by Dockerfile>**

```

redbot@redbot-G3-3500: ~/PythonProjects/Calculator
File Edit View Search Terminal Help

(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker build . -t devops_mini_project_version2
Sending build context to Docker daemon 38.91MB
Step 1/7 : FROM python:3.8-buster
--> 53da5c105f01
Step 2/7 : WORKDIR /app
--> Using cache
--> 56282f7537f2
Step 3/7 : COPY Calculator_main.py .
--> Using cache
--> 169effe9ba8b
Step 4/7 : COPY requirements.txt .
--> 32d2f27b386e
Step 5/7 : COPY Scientific_Calculator.py .
--> 63a3a65be8fd
Step 6/7 : RUN pip3 install -r requirements.txt
--> Running in a6f4a3c4d1f6
Collecting appdirs==1.4.4
  Downloading appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Collecting attrs==20.3.0
  Downloading attrs-20.3.0-py2.py3-none-any.whl (49 kB)
Collecting coverage==5.5
  Downloading coverage-5.5-cp38-cp38-manylinux2010_x86_64.whl (245 kB)
Collecting distlib==0.3.1
  Downloading distlib-0.3.1-py2.py3-none-any.whl (335 kB)
Collecting filelock==3.0.12

```

Fig 18: Building Docker image from Dockerfile

Now check in docker images on your system the generated image using command:

- **sudo docker images**

```

(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
devops_mini_project_version2    latest   56bd6ba43508  About a minute ago  913MB
<none>              <none>   e8c176720aa3   3 minutes ago   883MB
imasadansari/devops_calculator    latest   195693810cf9   32 hours ago   913MB
devops_mini_project    latest   55ba07a3b5fe   32 hours ago   913MB
devops_mini_project/testimage    version1  0e9d2dc76ce2   32 hours ago   561MB
ubuntu18.04/testimage    version1  0e3e786c7d30   4 days ago    560MB
python                3.8-buster  53da5c105f01   5 days ago    883MB
python                3           2c31ca135cf9   5 days ago    885MB
python                latest    2c31ca135cf9   5 days ago    885MB
ubuntu                18.04     329ed837d508  13 days ago   63.3MB
redis                 latest    eb0ab2d55fdf   5 weeks ago   104MB
busybox               latest    22667f53682a  6 weeks ago   1.23MB
ubuntu                 latest    f63181f19b2f   7 weeks ago   72.9MB
hello-world            latest    bf756fb1ae65  14 months ago  13.3kB
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ 

```

Fig 19: Build image “devops\_mini\_project\_version2” can be seen in docker images.

As we have successfully built the docker image for our terminal based calculator. Let's run it by command :

- Sudo docker run -i <tag/name of Image> {-i tag is used to run the image in an interactive terminal such that it expects a user}.

```
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker run -i devops_mini_project
TERM environment variable not set.
Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 2
Enter the number: 6
Answer : 720

Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 1
Enter the number: 169
Answer : 13.0

Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : e
(venv) redbot@redbot-G3-3500:~/PythonProjects/Calculator$
```

Fig 20: Running docker image

Now we are ready to push our image to our Docker Hub repository using below steps:

1. Login into Docker Hub from command line - docker login --username=yourhubusername --email=youremail
2. Attaching a tag to our docker image:

```
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ ls
og      Dockerfile          'Logs last 5 hours.png'  README.md      Scientific_Calculator.py    venv
ain.py  'Logs last 3 days.png'  __pycache__           requirements.txt  test_ScientificCalculator.py
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker tag 69s547321xa2 imasadansari/devops_mini_project:firsttry
```

3. Pushing the image to our repository:

```
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker push imasadansari/devops_mini_project:firsttry
```

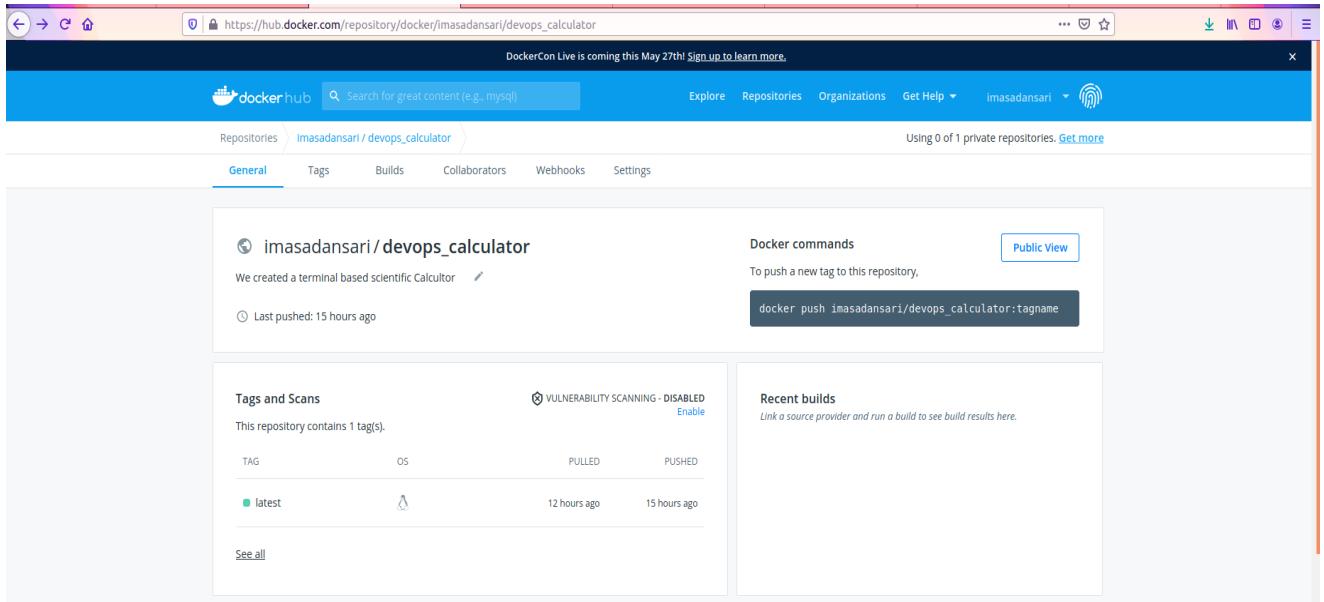


Fig 21: Docker hub

As we can see, our image was pushed to Docker Hub from our local machine. This is great but we do not want to manually build and push every time we make some changes or add new features. Once again, we will go with CircleCI for automating this.

### **Docker Hub and CircleCI Automation builds:**

Since we have already written the Dockerfile and successfully pushed the image on Docker Hub, half of our work is already done.

To make CircleCI do this job for us, we need to again edit the config.yml file which was present in the .circleci folder of our project directory.

```

1 # Python CircleCI 2.1 configuration file
2 version: 2.1
3 jobs:
4   build:
5     # executor: docker-publisher
6     environment:
7       IMAGE_NAME: devops_calculator
8     docker:
9       - image: circleci/python:3.7
10    working_directory: ~/repo
11  steps:
12    # Step 1: obtain repo from GitHub
13    - checkout
14    # Step 2: setup remote docker engine
15    - setup_remote_docker
16    # Step 3: create virtual env and install dependencies
17    - run:
18      name: install dependencies
19      command: |
20        python3 -m venv venv
21        . venv/bin/activate
22        pip install -r requirements.txt
23    # Step 3: run linter and tests
24    - run:
25      name: run tests
26      command: |
27        . venv/bin/activate
28        pytest -v --cov=Scientific_Calculator
29    # Step 4: Build Docker Image
30    - run:
31      name: Build and Push Docker Image
32      command: |
33        docker build -t $IMAGE_NAME:latest .
34        echo "$DOCKERHUB_PASSWORD" | docker login -u "$DOCKERHUB_USERNAME" --password-stdin
35        docker tag $IMAGE_NAME $DOCKERHUB_USERNAME/$IMAGE_NAME:latest

```

config.yml is valid and ready to commit show in JSON

Fig 22 : Contains Config.yml code

In the above sections we ignored the docker part. Let's have a look at it now.

- `setup_remote_docker` - It tells CircleCI to allocate a new Docker Engine, separate from the environment that is running our job, specifically to execute the Docker commands.
- `Build and Push Docker Image` - This is the crucial part. After successful testing, we want to build our docker image and push it to Docker Hub. Note that I have used an environment variable called `IMAGE_NAME` which specifies the name of my docker image. Firstly, we build the docker image with the same command we used in our local machine `docker build -t $IMAGE_NAME:latest .`

Now, remember we logged into Docker Hub from our terminal when building image locally?

We need to do this via CircleCI now. But it is not safe and a secure practice to store passwords and sensitive keys in a text format simply in the config.yml file. So, we add these credentials to the environment variables in our CircleCI project.

The screenshot shows the CircleCI web interface for the 'Dev-Ops-MiniProject' pipeline. The pipeline has run 17 times, with the most recent being 15 hours ago. All builds have been successful, with durations ranging from 1m 2s to 1m 21s. The pipeline configuration is named 'workflow'. The commits for each build are listed, along with their commit hash and message. The interface includes filters for pipelines and branches, and a 'Project Settings' button in the top right corner.

Fig : Go to Project Settings

The screenshot shows the 'Project Settings' page for the 'Dev-Ops-MiniProject'. The left sidebar lists various settings sections: Overview, Advanced, Environment Variables (which is currently selected and highlighted in grey), SSH Keys, API Permissions, Jira Integration, and Slack Integration. The main content area is currently empty, indicating no environment variables have been defined yet.

Fig : Select Environment Variables

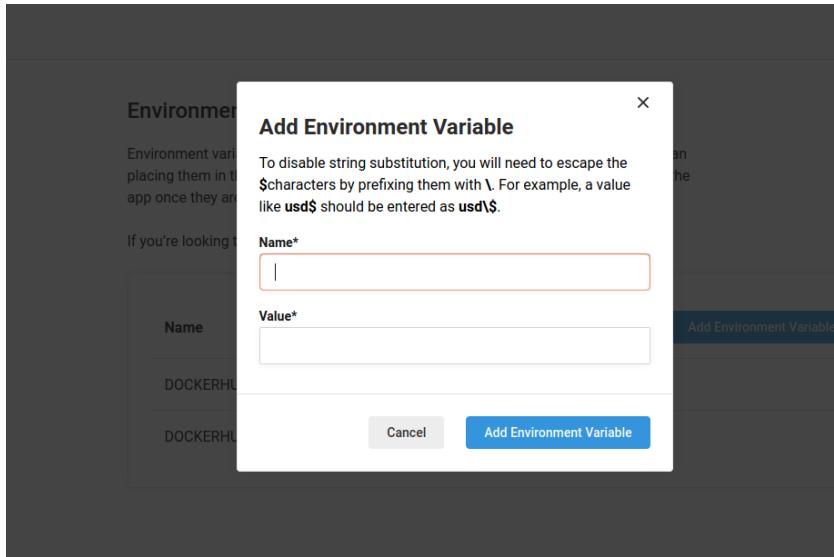


Fig : Then click on Add Environment Variable, and simply add the variable and its value.

Now, we are ready to use our Docker Hub credentials safely in our config.yml file.

```
echo "$DOCKERHUB_PASSWORD" | docker login -u "$DOCKERHUB_USERNAME" --password-stdin  
docker tag $IMAGE_NAME $DOCKERHUB_USERNAME/$IMAGE_NAME:latest  
docker push $DOCKERHUB_USERNAME/$IMAGE_NAME:latest
```

We simply login into Docker Hub by using the environment variables ( starts with '\$') and follow the same procedure of tagging our image and then pushing it.

Let us now push the updated config.yml file and our Dockerfile to our GitHub repo. And then, this happens on CircleCI:

The screenshot shows a CircleCI pipeline build page. The URL is https://app.circleci.com/pipelines/github/asadiiitb/Dev-Ops-MiniProject/17/workflows/4f0f94f9-781a-4b86-a3b6-ac01f735c144/jobs/17. The pipeline is named 'Dev-Ops-MiniProject' and the branch is 'main'. The job is labeled 'build (17)' and is marked as 'Success'. The build duration was 1m 1s / 15h ago, and it was queued for 0s. The executor used was Docker Medium. The branch is main, commit is e466582, and the author & message is 'Update README.md'. The pipeline has 17 parallel runs, with the first run showing steps: Spin up environment (2s), Preparing environment variables (0s), Checkout code (0s), Setup a remote Docker engine (4s), install dependencies (7s), run tests (0s), and Build and Push Docker Image (42s). All steps are marked as successful.

Let us checkout the Docker part. As we can see, it successfully ran our Dockerfile from Github which we pushed before. Also, the login and pushing to Docker Hub was Successful.

```
Login Succeeded
The push refers to a repository [docker.io/******/devops_calculator]

151f21f3: Preparing
533f8141: Preparing
2ae0cd39: Preparing
9c875b6c: Preparing
5feaafc21: Preparing
0c0b2d66: Preparing
639a150e: Preparing
a179c158: Preparing
717d0e01: Preparing
447ffe30: Preparing
01416dd2: Preparing
6d8c72c2: Preparing
48d6e877: Preparing
latest: digest: sha256:90124bce624e2de5db865790f818e3c4ba9a25f5e03de915fbb0e7b50193b1a4 size: 3258
CircleCI received exit code 0
```

By doing this, we have successfully automated our calculator project. Let's just try to run this image we pushed from CircleCI on our local machine which will verify that our image will work on any linux machine:

```
9076231..658d54b  Math => Math
redbot@redbot-G3-3500:~/PythonProjects/Calculator$ sudo docker run -i imasadansari/devops_calculator
[sudo] password for redbot:
Sorry, try again.
[sudo] password for redbot:
Unable to find image 'imasadansari/devops_calculator:latest' locally
latest: Pulling from imasadansari/devops_calculator
e22122b926a1: Already exists
f29e09ae8373: Already exists
e319e3daef68: Already exists
e499244fe254: Already exists
5a6ebcd20e89: Already exists
56b703a5a371: Already exists
dec810778c22: Already exists
505ac614a24: Already exists
36deae172a15: Already exists
6042dfdeaf0f: Pull complete
b62408ca8804: Pull complete
682b113b4afd: Pull complete
096e526720b5: Pull complete
1ca281bf0b35: Pull complete
Digest: sha256:7d177ccdec73d0137d055dc4485ae49f8e84a9723fb947e834d8a30478aab037
Status: Downloaded newer image for imasadansari/devops_calculator:latest
TERM environment variable not set.
Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 3
Enter the number: 2
Answer : 0.6931471805599453

Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 4
Enter the number: 2
```

Fig : Pulling latest image pushed via CircleCI

```

096e526720b5: Pull complete
1ca281bf0b35: Pull complete
Digest: sha256:7d177ccdec73d0137d055dc4485ae49f8e84a9723fb947e834d8a30478aab037
Status: Downloaded newer image for imasadansari/devops_calculator:latest
TERM environment variable not set.
Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 3
Enter the number: 2
Answer : 0.6931471805599453

Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : 4
Enter the number: 2
Enter the power : 10
Answer : 1024.0

Select Operation.
1: Sqaure Root - √x
2: Factorial - x!
3: Natural Log - ln(x)
4: Power - x^y
e: Exit
Enter your choice : []

```

Fig : Updated image running

## Logging:

Logs provide us with an extra set of eyes that are constantly looking at the flow that an application is going through. They can store information, like which user or IP accessed the application. If an error occurs, then they can provide more insights than a stack trace by telling you what the state of the program was before it arrived at the line of code where the error occurred.

Python provides a logging system as a part of its standard library, so we can quickly add logging to our application.

In Python, by default, there are 5 standard levels indicating the severity of events. Each has a corresponding method that can be used to log events at that level of severity. The defined levels, in order of increasing severity, are the following:

1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

```
logging.basicConfig(filename='calcy.log', level=logging.DEBUG,
                    format='%(asctime)s:%(levelname)s:%(message)s')
```

For this project, we have changed the level to DEBUG, and the logs are being stored in a log file called Calculator.log in the specified format.

Remember the Scientific\_Calculator.py module I created at the beginning which contained the main functions? I have added logging in those functions so that whenever these functions are called, the application will store the values entered by the user in these logs.

\*\* Logging image of code

Now, whenever any function is called, the value entered by the user will be logged. For example:

\*\* Log file image

After Running the application:

\*\* Image of running code and log file

As we can see the new inputs are being recorded in log file

## **Monitoring:**

For monitoring, we are going to use the ELK stack, mainly Kibana. Kibana is an open source browser based visualization tool mainly used to analyze large volumes of logs in the form of line graph, bar graph, pie charts, heat maps, region maps, coordinate maps, etc. The visualization makes it easy to predict or to see the changes in trends of errors or other significant events of the input source.

First, we setup our 14 day free trial account on [www.elastic.co](http://www.elastic.co)

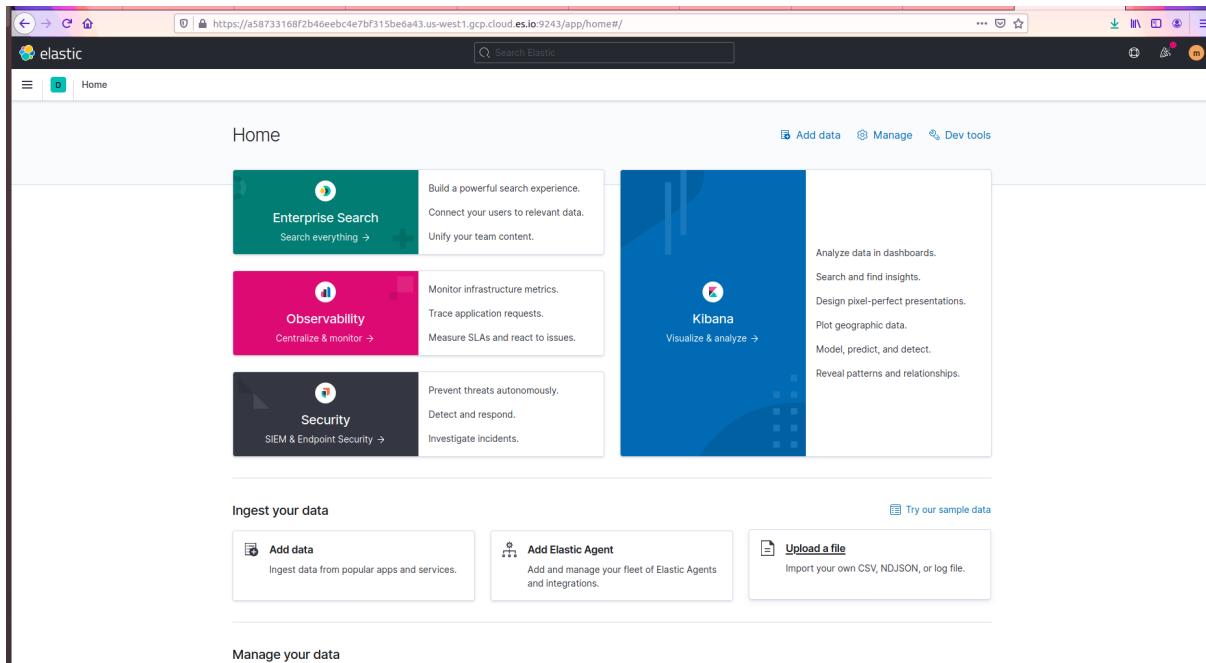
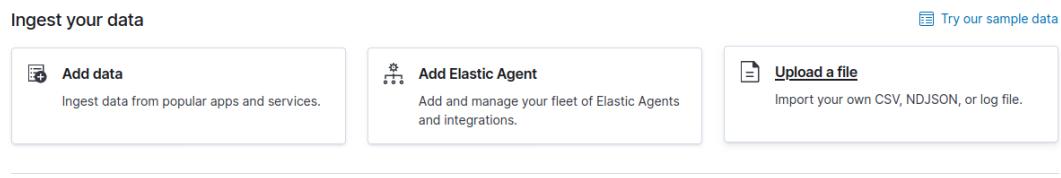


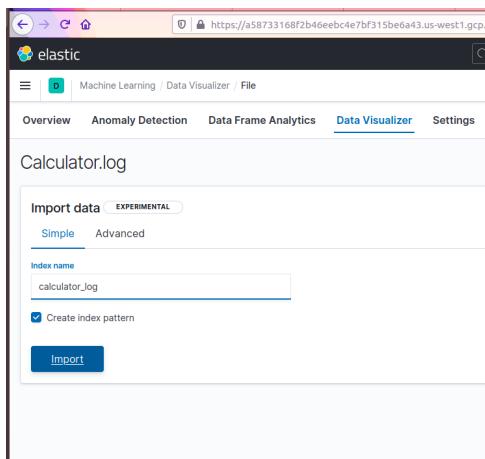
Fig: ELK Dashboard

Next, we upload the log file generated by our calculator python application.



#### Manage your data

Click on Import and add an index name.



The screenshot shows the Elasticsearch Data Visualizer interface. At the top, there's a navigation bar with icons for back, forward, and home, followed by the URL <https://a58733168f2b46eebc4e7bf315be6a43.us-west1.gcp.cloud.es.io>. Below the URL is the elastic logo and a search bar.

The main menu includes 'Machine Learning / Data Visualizer / File', 'Overview', 'Anomaly Detection', 'Data Frame Analytics', 'Data Visualizer' (which is underlined in blue), and 'Settings'.

The page title is 'Calculator.log'. Under 'File contents', it shows the first 54 lines of the log file:

```

1 2021-03-14 17:09:29,402:DEBUG:Square Root of 225 = 15.0
2 2021-03-14 17:09:29,402:DEBUG:Factorial of 10 = 3628800
3 2021-03-14 17:09:29,402:DEBUG:Natural Log of 225 = 5.41610040220442
4 2021-03-14 17:09:29,402:DEBUG:3 Raised to Power -4 = 15.0
5 2021-03-14 17:09:29,402:DEBUG:Square Root of 225 = 15.0
6 2021-03-14 14:49:13,651:DEBUG:Factorial of 10 = 3628800
7 2021-03-14 14:49:13,651:DEBUG:Natural Log of 225 = 5.41610040220442
8 2021-03-14 14:49:13,651:DEBUG:3 Raised to Power -4 = 15.0
9 2021-03-14 14:49:13,651:DEBUG:Square Root of 225 = 15.0
10 2021-03-14 15:03:17,199:DEBUG:Factorial of 10 = 3628800
11 2021-03-14 15:03:17,199:DEBUG:Natural Log of 225 = 5.41610040220442
12 2021-03-16 15:03:17,199:DEBUG:3 Raised to Power -4 = 15.0
13 2021-03-16 15:03:31,515:DEBUG:Square Root of 225 = 15.0
14 2021-03-16 15:04:31,515:DEBUG:Factorial of 10 = 3628800
15 2021-03-16 15:04:31,515:DEBUG:Natural Log of 225 = 5.41610040220442
...

```

Below the file contents is a 'Summary' section with the following details:

- Number of lines analyzed:** 54
- Format:** delimited
- Delimiter:**,
- Has header row:** false
- Time field:** column1
- Time format:** yyyy-MM-dd HH:mm:ss

At the bottom of the summary section are two buttons: 'Override settings' and 'Analysis explanation'.

At the very bottom of the page is a dark bar with 'Import' and 'Cancel' buttons.

If the visualization build is successful, we will see something like this

The screenshot shows the Elasticsearch pipeline status page. At the top, there's a timeline with five green checkmark icons and labels: 'File processed', 'Index created', 'Ingest pipeline created', 'Data uploaded', and 'Index pattern created'.

Below the timeline is a summary table:

<input checked="" type="checkbox"/> Import complete	
Index	calculator_log
Index pattern	calculator_log
Ingest pipeline	calculator_log-pipeline
Documents ingested	54

Below the table are five action buttons:

- View index in Discover** (with a magnifying glass icon)
- Open in Data Visualizer** (with a bar chart icon)
- Index Management** (with a gear icon)
- Index Pattern Management** (with a gear icon)
- Create Filebeat configuration** (with a document icon)

At the bottom of the page is a dark bar with 'Back' and 'Cancel' buttons, and the URL [https://a58733168f2b46eebc4e7bf315be6a43.us-west1.gcp.cloud.es.io:9243/app/ml/jobs/new\\_job/datavisualizer?index=6fed76e0-877c-11eb-920b-c1f16996ebca](https://a58733168f2b46eebc4e7bf315be6a43.us-west1.gcp.cloud.es.io:9243/app/ml/jobs/new_job/datavisualizer?index=6fed76e0-877c-11eb-920b-c1f16996ebca)

Let's open our log file in data visualizer.

Note that we can change have multiple indexes and we can refine the graph by the time from which we want to see the logs.

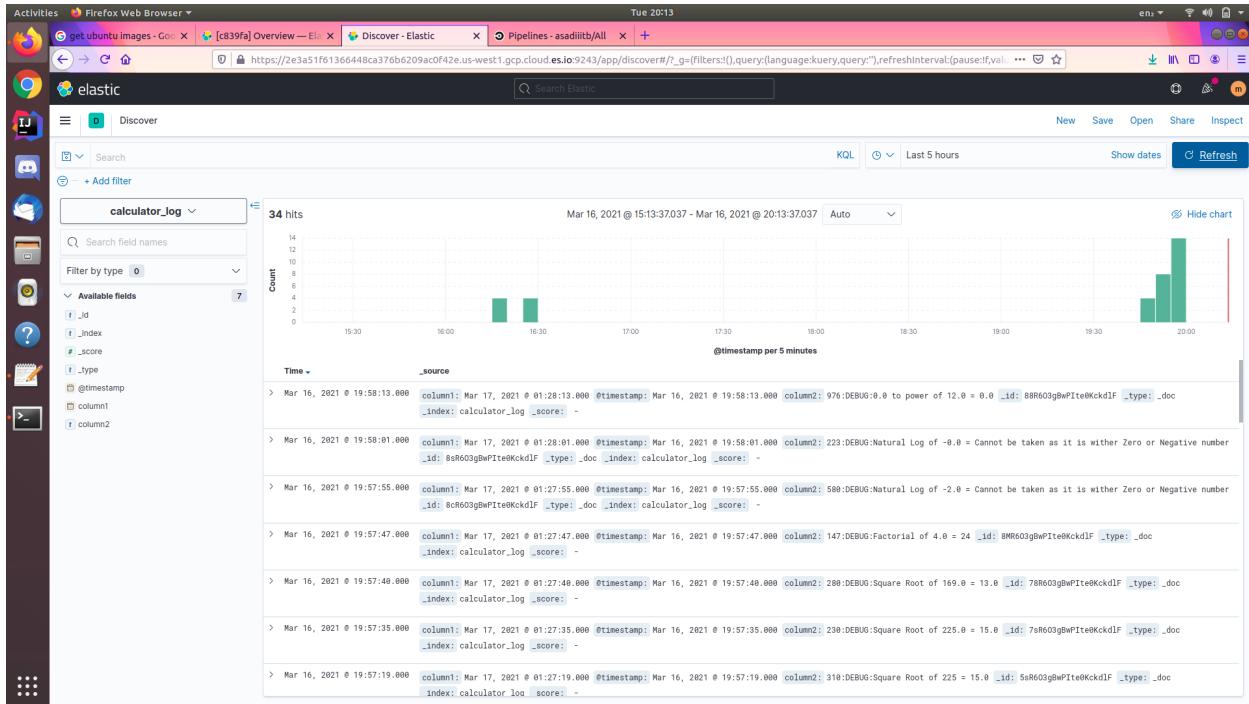


Fig: Visualizing logs of last 5 hours

We can refine this for certain other duration like below.

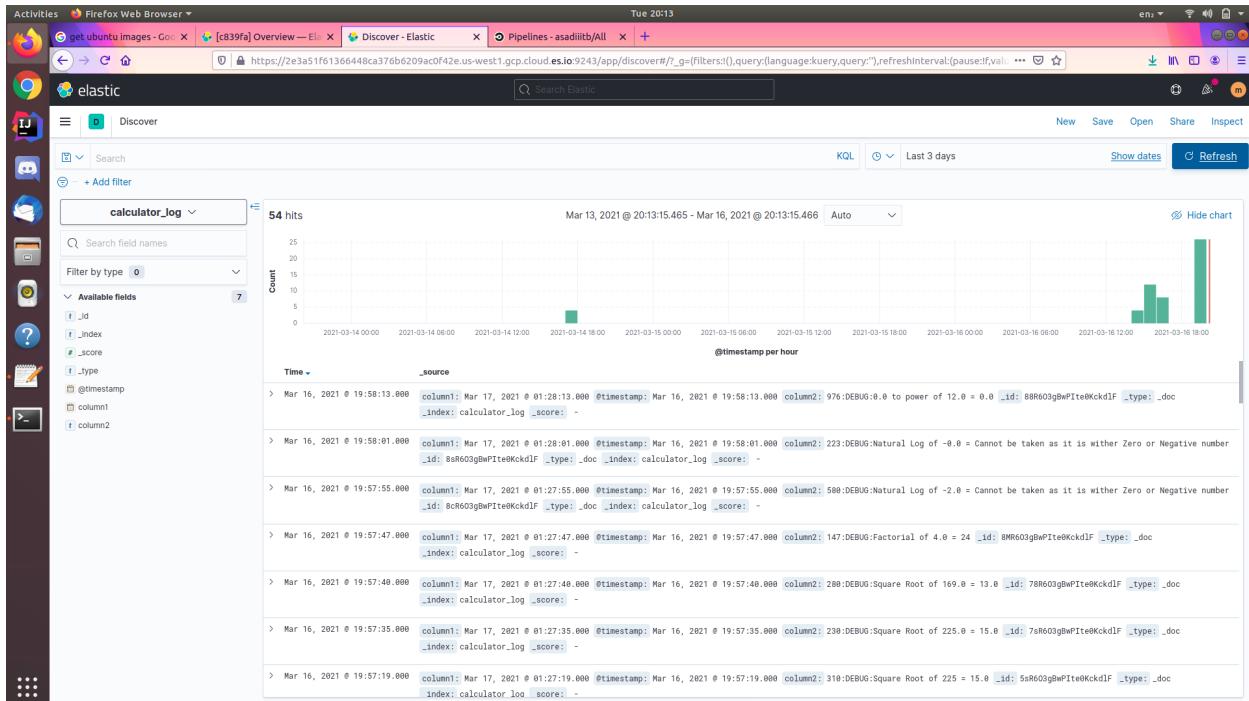


Fig: Visualizing logs of last 3 days

## Future Scope:

- We can use a convert this terminal based calculator to a GUI based Calculator, Glimse of which can be seen below. You can view its source code at <https://github.com/asadiiitb/ScientificCalculator>. Due to Display Environment error we were unable to connect it to CircleCI and Docker Image.

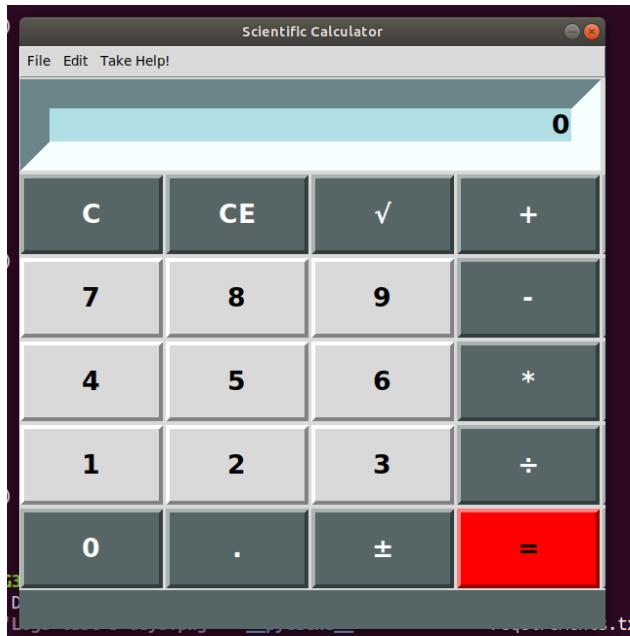


Fig : Scientific Calculator in Standard Mode

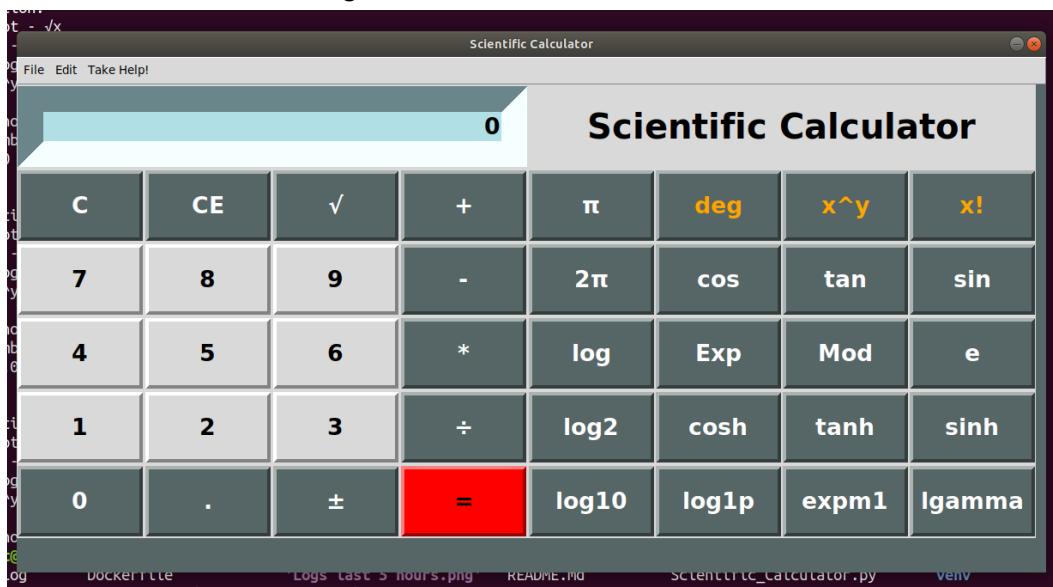


Fig : Calculator in Scientific Mode

## **References**

1. <https://circleci.com/blog/using-circleci-workflows-to-replicate-docker-hub-automated-builds/>
2. <https://circleci.com/blog/how-to-build-a-docker-image-on-circleci-2-0/>
3. <https://realpython.com/python-logging/>
4. <https://circleci.com/blog/build-cicd-pipelines-using-docker/>
5. <https://realpython.com/docker-in-action-fitter-happier-more-productive/>
6. <https://realpython.com/python-continuous-integration/>

**GitHub** : <https://github.com/asadiiitb/Dev-Ops-MiniProject>

**Docker Hub** : [https://hub.docker.com/repository/docker/imasadansari/devops\\_calculator](https://hub.docker.com/repository/docker/imasadansari/devops_calculator)