

Mini Project - Scientific Calculator with DevOps

CS816 - Software Production Engineering

Battula Jashwanth Sai - MT2022029

March 2023

Contents

1	Problem Statement	2
2	Deliverables	2
3	What and Why DevOps?	2
3.1	Core Principles of DevOps	3
3.2	Goal of DevOps	3
3.3	Benefits	4
4	Tools Used	4
5	Development Steps	4
5.1	Setup Development Environment	4
5.2	Source Control Management Setup	5
5.3	Docker and Docker-Hub	6
5.4	CI/CD Pipeline Setup	6
5.4.1	Build and Test	6
5.4.2	Containerize and Publish to DockerHub	9
5.4.3	Deployment	9
5.4.4	Viewing the Pipeline	12
6	Monitoring	13
6.0.1	Configuration	13

1 Problem Statement

Create a scientific calculator program with the following operations:

- Square root function \sqrt{x}
- Factorial function $x!$
- Natural logarithm (base e) $\ln(x)$
- Power function x^b

2 Deliverables

- GitHub Repository - [spe-mini-project](#)
- Docker Hub Repository - [docker-repository](#)
- Calculator demo - [gdrive link](#)

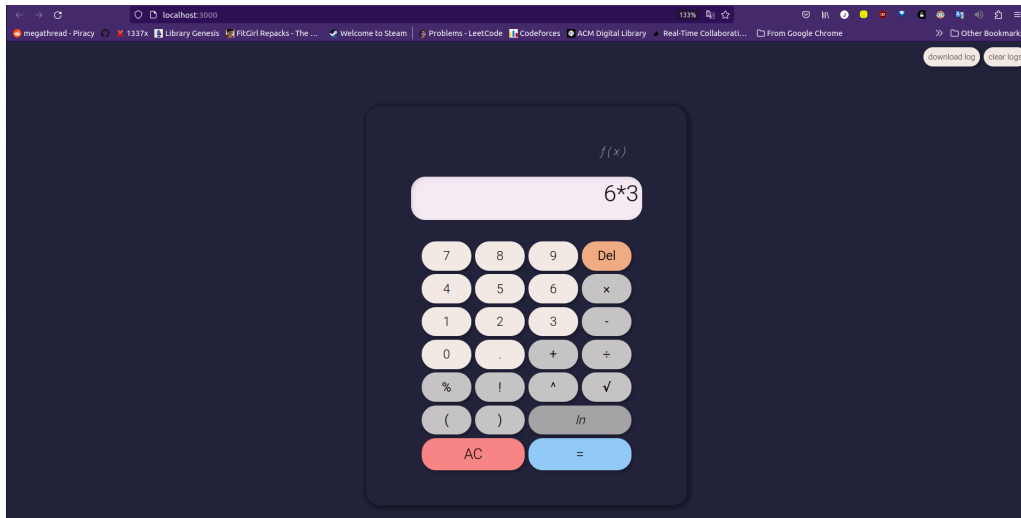


Figure 1: Scientific Calculator

3 What and Why DevOps?

DevOps is a combination of software development (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility.

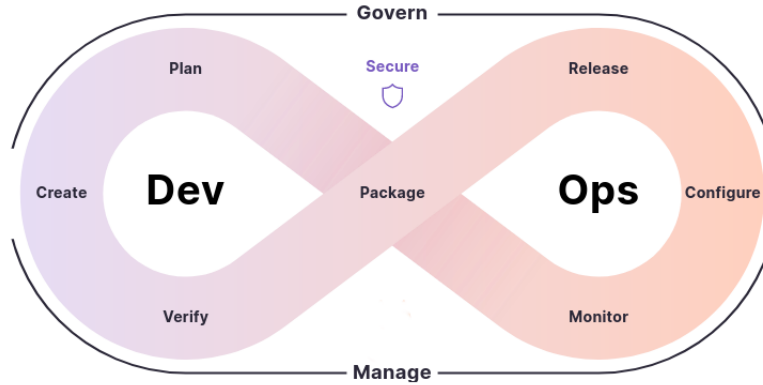


Figure 2: DevOps Life Cycle

3.1 Core Principles of DevOps

The DevOps methodology comprises four key principles that guide the effectiveness and efficiency of application development and deployment. These principles, listed below, center on the best aspects of modern software development.

- Automation of the software development lifecycle. This includes automating testing, builds, releases, the provisioning of development environments, and other manual tasks that can slow down or introduce human error into the software delivery process.
- Collaboration and communication. A good DevOps team has automation, but a great DevOps team also has effective collaboration and communication.
- Continuous improvement and minimization of waste. From automating repetitive tasks to watching performance metrics for ways to reduce release times or mean-time-to-recovery, high performing DevOps teams are regularly looking for areas that could be improved.
- Hyperfocus on user needs with short feedback loops. Through automation, improved communication and collaboration, and continuous improvement, DevOps teams can take a moment and focus on what real users really want, and how to give it to them.

By adopting these principles, organizations can improve code quality, achieve a faster time to market, and engage in better application planning.

3.2 Goal of DevOps

DevOps represents a change in mindset for IT culture. In building on top of Agile practices, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes.

Adopting a DevOps strategy enables businesses to increase operational efficiencies, deliver better products faster, and reduce security and compliance risk.

3.3 Benefits

Adopting DevOps breaks down barriers so that development and operations teams are no longer siloed and have a more efficient way to work across the entire development and application lifecycle. Without DevOps, organizations often experience handoff friction, which delays the delivery of software releases and negatively impacts business results.

- **Collaboration** - Adopting a DevOps model creates alignment between development and operations teams; handoff friction is reduced and everyone is all in on the same goals and objectives.
- **Fluid responsiveness** - More collaboration leads to real-time feedback and greater efficiency; changes and improvements can be implemented quicker and guesswork is removed.
- **Shorter cycle time** - Improved efficiency and frequent communication between teams shortens cycle time; new code can be released more rapidly while maintaining quality and security.

4 Tools Used

- **Git** - 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.
- **GitHub** - is an Internet hosting service for software development and version control using Git.
- **GitHub Actions** - GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.
- **Docker** - Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.
- **Node** - Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.
- **npm** - is a package manager for the JavaScript programming language - used to install dependencies, build, test and run JavaScript Projects.
- **React** - React is a free and open-source front-end JavaScript library - used to build calculator frontend.

5 Development Steps

5.1 Setup Development Environment

- Install nodejs

```
$ sudo apt install nodejs
```

The nodejs package contains both the node and npm binaries.

Verify Install using:

```
$ node -v
```

- Install create-react-app using npm

```
$ npm i create-react-app
```

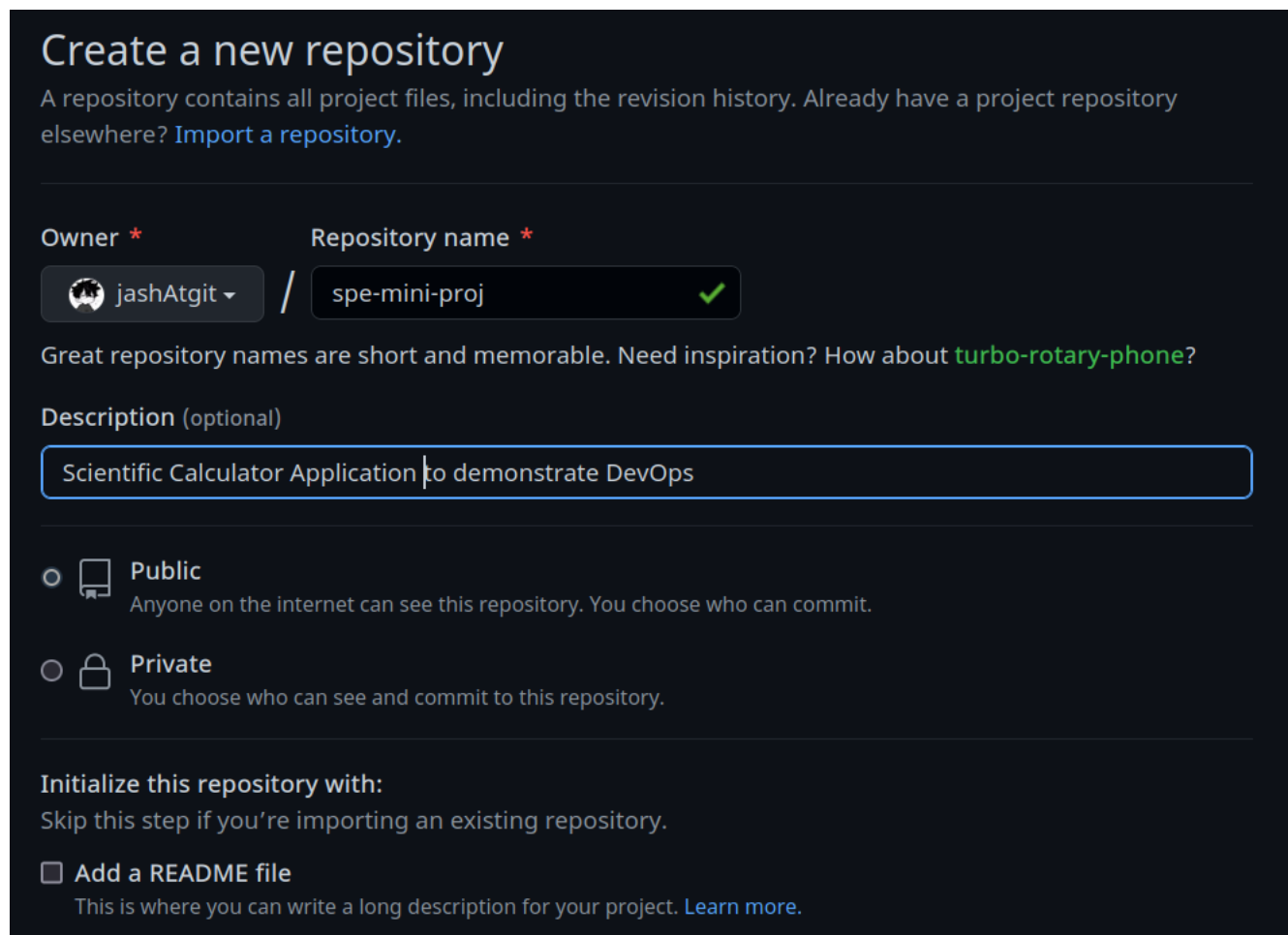
Run create-react-app to build a template React project to get started:

```
$ npx create-react-app spe-mini-project
$ cd spe-mini-project
$ npm start
```

5.2 Source Control Management Setup

The basic goal of SCM is to keep software in its current state (known as the “baseline”) while allowing developers to work on new versions for new features or repairs. This is accomplished with the help of GitHub.

Create a Remote Repository on GitHub :



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and explains that a repository contains all project files, including the revision history. It offers a link to 'Import a repository' if the user already has one elsewhere. Below this, there are two main input fields: 'Owner' and 'Repository name'. The 'Owner' field is set to 'jashAtgit' with a dropdown arrow. The 'Repository name' field is set to 'spe-mini-proj' and has a green checkmark next to it. Below these fields, there is a suggestion: 'Great repository names are short and memorable. Need inspiration? How about turbo-rotary-phone?'. The 'Description (optional)' field contains the text 'Scientific Calculator Application to demonstrate DevOps'. Below the description field, there are two radio button options for repository visibility: 'Public' (selected) and 'Private'. The 'Public' option is described as 'Anyone on the internet can see this repository. You choose who can commit.' The 'Private' option is described as 'You choose who can see and commit to this repository.' At the bottom, there is a section titled 'Initialize this repository with:' which includes a note to 'Skip this step if you're importing an existing repository.' and a checkbox labeled 'Add a README file' with a link to 'Learn more.'

Figure 3: Create Repository

Follow the steps mentioned to initialize a local repository on your machine and push your code. Run these in the working directory of your app created using create-react-app.

```
$ git remote add origin https://github.com/jashAtgit/spe-mini-project.git
$ git branch -M master
$ git push -u origin master
```

Use GitHub **PAT(Pesonal Access Token)** as password to prove authentication when prompted for password and your github userid as username.

5.3 Docker and Docker-Hub

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

Create an Account on Docker Hub - This account will be used to create docker repository and then push docker images to it.

Create a Public Repository on Docker Hub -

The screenshot shows the Docker Hub 'Create repository' interface. The repository name is 'jashsai/spe-mini-project' and the description is 'Docker Repo for Scientific Calculator - SPE'. The visibility is set to 'Public'. A 'Pro tip' box shows CLI commands for tagging and pushing an image. At the bottom are 'Cancel' and 'Create' buttons.

Figure 4: Docker Hub Repository Creation

5.4 CI/CD Pipeline Setup

A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.

In your working directory, create a *ci.yml* file with the following directory structure to get started with automating workflows using GitHub Actions.

5.4.1 Build and Test

Building stage involves building the software and creating the executable files that will be deployed to the production environment. This includes compiling the code, packaging the files, and creating the deployment

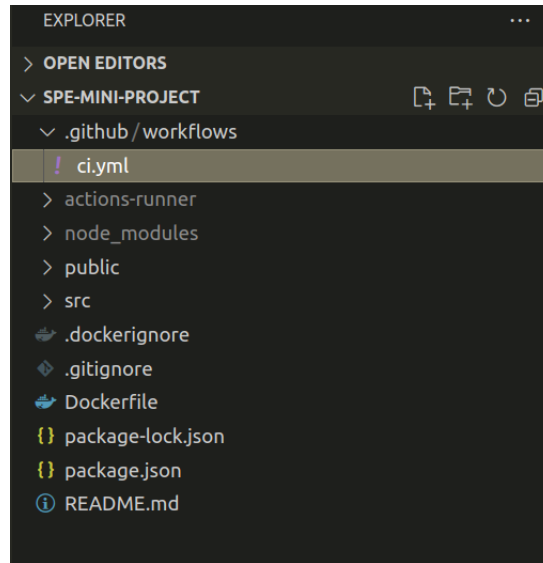


Figure 5: Directory Structure for workflow files

artifacts.

Testing stage involves testing the software to ensure it meets the specified requirements and is ready for deployment. This includes unit testing, integration testing, performance testing, and acceptance testing. The `@testing-library/react` library is used for testing in this project.

View full testing code `App.test.js` [here](#).

Now we setup jobs for build and testing the project automatically when changes are pushed to the *master* branch.

This is done using the `ci.yml` file, which uses the workflow syntax for github actions.

```
1  name: CI/CD Pipeline
2
3  on:
4    push:
5      branches: [ "master" ]
6    pull_request:
7      branches: [ "master" ]
8
9  jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     strategy:
15       matrix:
16         node-version: [16.x]
17
18
```

```

v ✓ Run npm test
3679
3680 PASS src/App.test.js
3681   Calculator
3682     ✓ calculator dev element is rendered (55 ms)
3683     ✓ displays input on screen when button is clicked (31 ms)
3684     ✓ updates display on button and operator click (18 ms)
3685     ✓ addition operation (124 ms)
3686     ✓ subtraction operation (40 ms)
3687     ✓ double subtraction without brackets (30 ms)
3688     ✓ double subtraction with brackets (33 ms)
3689     ✓ clears display when AC is pressed (10 ms)
3690     ✓ deletes last digit when Del is pressed (34 ms)
3691     ✓ factorial normal functionality (14 ms)
3692     ✓ factorial negative number : edge case 1 (14 ms)
3693     ✓ factorial input = 0 : edge case 2 (16 ms)
3694     ✓ factorial : large number 50! : edge case 3 (13 ms)
3695     ✓ log 10 base e (49 ms)
3696     ✓ log 0 base e (27 ms)
3697     ✓ log 1 base e (28 ms)
3698     ✓ log base e of a negative number (52 ms)
3699     ✓ sqrt functionality (53 ms)
3700     ✓ sqrt edge case : negative number (58 ms)
3701     ✓ pow functionality (27 ms)
3702     ✓ pow edge case: 5**0 (42 ms)
3703     ✓ pow edge case: 5**-2 (negative exponent) (36 ms)
3704     ✓ pow edge case: -5**2, without brackets (27 ms)
3705     ✓ pow (-5)**2 test (30 ms)
3706
3707   Test Suites: 1 passed, 1 total
3708   Tests:       24 passed, 24 total
3709   Snapshots:   0 total
3710   Time:        3.433 s
3711   Ran all test suites.
```

Figure 6: Testing Results

```

19  steps:
20    - uses: actions/checkout@v3
21    - name: Use Node.js ${{ matrix.node-version }}
22      uses: actions/setup-node@v3
23      with:
24        node-version: ${{ matrix.node-version }}
25        cache: 'npm'
26    - run: npm install
27    - run: npm run build
28    - run: npm test
```

The *name* attribute is self-explanatory the *on* attribute is used to specify the action on which to run the job. The *jobs* attribute is used to list one or more jobs, and the sequence of commands to run are specified under the *steps* attribute. The *yml* file uses indentation to separate blocks and hence needs to be properly indented.

5.4.2 Containerize and Publish to DockerHub

Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. **Docker** is a containerization solution that allows containers to be quickly and easily created.

We have built the project and ran tests on it, in the previous section. Now for the next part, the *publish* job will build a docker image of our project and push it to the docker hub [public repository](#).

```
1 publish:
2   needs: build
3   runs-on: ubuntu-latest
4
5   steps:
6   - uses: actions/checkout@v3
7     name: Check out code
8
9   - uses: mr-smithers-excellent/docker-build-push@v6
10     name: Build & push Docker image
11     with:
12       image: jashsai/spe-mini-project
13       tags: latest
14       registry: docker.io
15       dockerfile: Dockerfile
16       username: ${ secrets.DOCKER_USERNAME }
17       password: ${ secrets.DOCKER_PASSWORD }
```

The *publish* job uses the [Docker Build and Push Action v6.0](#) from github marketplace, which simplifies building and pushing a docker image.

Fill in the required parameters, like *image*, *tags*, *dockerfile*, and *username*, *password* - of your docker account.

The *username* and *password* fields reference the corresponding secrets from github actions. You can add repository specific secrets on GitHub by going to: **Settings under your repository** → **Secrets and Variables** → **Actions** → **New Repository Secret**

5.4.3 Deployment

Deployment stage involves deploying the software to the production environment. This includes setting up the necessary infrastructure, configuring the environment, and deploying the artifacts.

The *deploy* job is responsible for pulling the docker image from docker hub and spinning up a new docker container on a local machine. For github to push jobs onto a local machine we have to install and configure what is called a self-hosted runner on our local machine. On running this self-hosted runner application on our local machine, the runner listens for jobs to run from github. We will use this runner to deploy our docker image.

The *deploy* job is as follows:

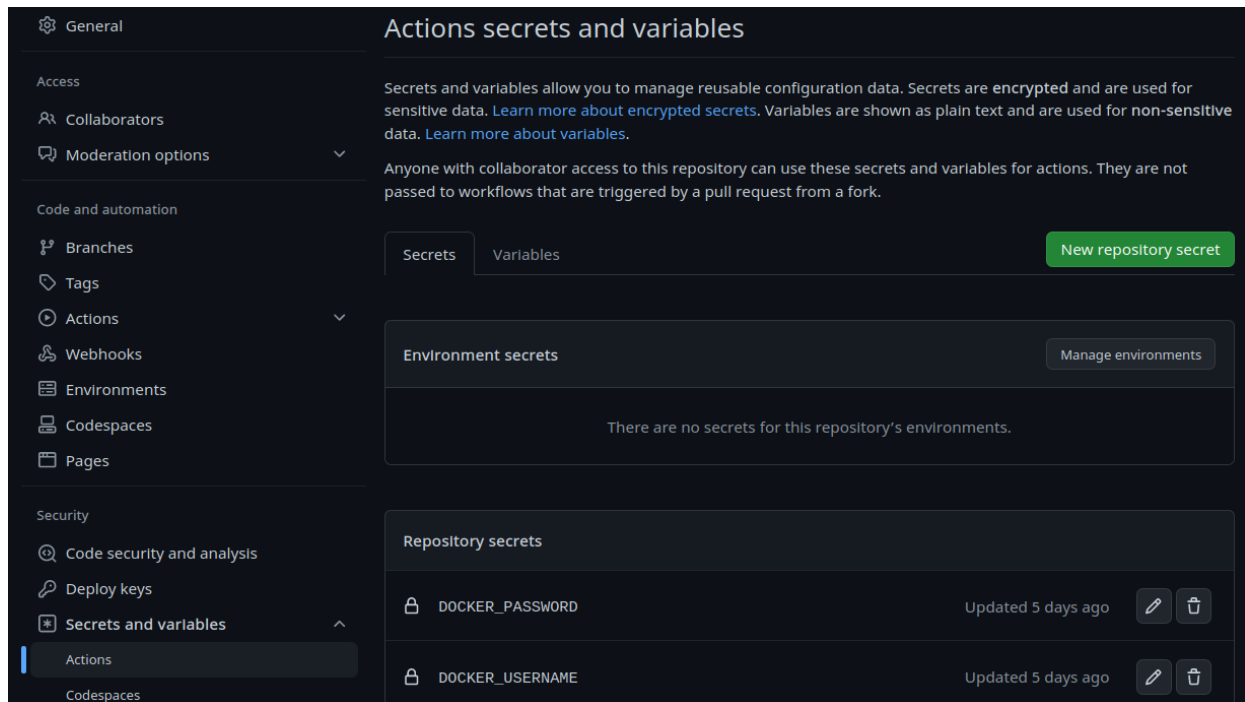


Figure 7: Adding Secrets

```

1  deploy:
2      needs: publish
3      runs-on: self-hosted
4
5      steps:
6      - name: Pull Docker image
7        run: docker pull jashsai/spe-mini-project:latest
8      - name: Stop running calculator-spe
9        run: docker stop calculator-spe || true
10     - name: Start new container
11       run: docker run --name calculator-spe -d -p 3000:3000
12         jashsai/spe-mini-project:latest
13     - name: Clean up unused containers
14       run: docker system prune -af

```

This pull the image from the repository `jashsai/spe-mini-project:latest` and starts a new container named `calculator-spe` if another container with the same name does not exist.

The **self-hosted** keyword under `runs-on` field tells github actions to run this particular job on a self-hosted runner.

Configuring a local machine as a self-hosted runner is done by: **Settings under your repository** → **Actions** → **Runners** → **New Self-Hosted Runner**. Follow the steps mentioned to configure the self-hosted runner.

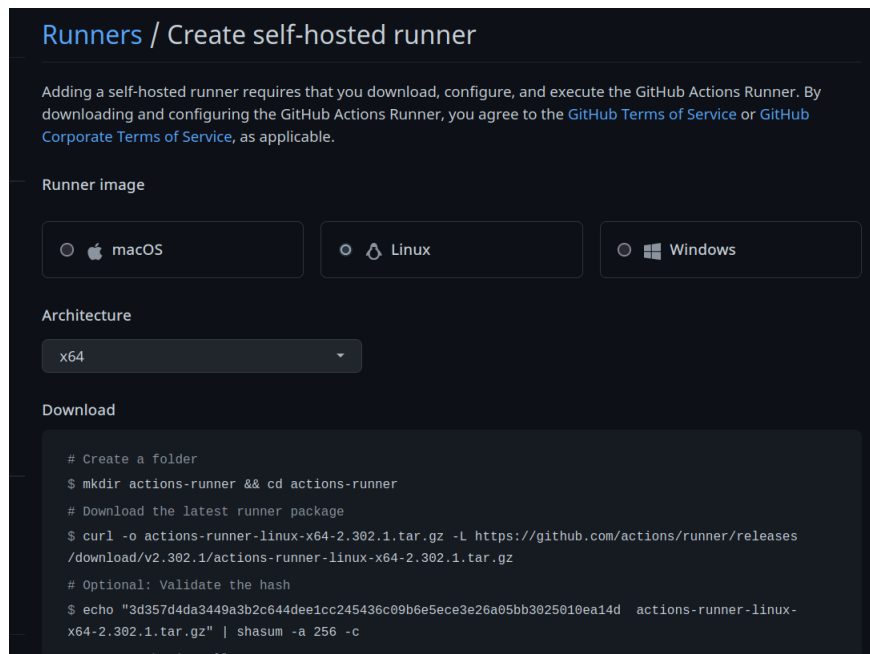


Figure 8: Configuring a self-hosted runner - part 1

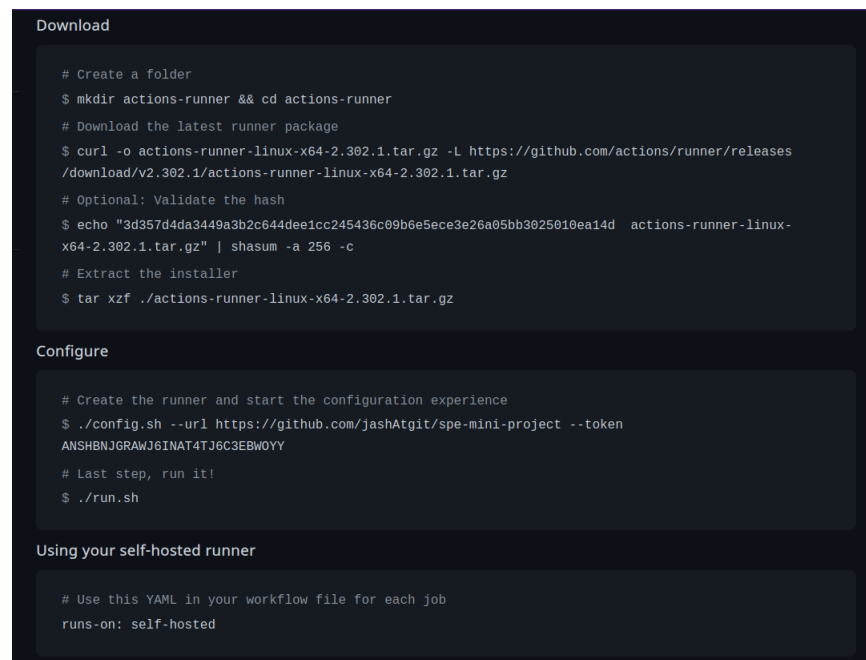


Figure 9: Configuring a self-hosted runner - part 2



Figure 10: Pipeline View - 1

5.4.4 Viewing the Pipeline

Pushing new changes to master branch will trigger the pipeline that we have setup Build and Test → Publish → Deploy.

We can view this under the *Actions* tab of our repository.

Click on a job for a detailed view.



Figure 11: Pipeline View(Detailed) - 2

6 Monitoring

This stage involves monitoring the software and the environment to ensure it is running smoothly and meeting the required performance and availability metrics. This includes *logging*, *metrics collection*, and *alerting*.

We will be using the **Elastic Stack**, which is a group of open source products from Elastic designed to help users take data from any type of source and in any format, and search, analyze and visualize that data in real time. The Elastic Stack three tools - *Elasticsearch*, *Kibana*, *Beats*, and *Logstash* (also known as the **ELK Stack**).

The following structured format is used for logging data to be used with Elastic Stack, for monitoring and analysis.

```
1 {  
2   "timestamp": "2023-03-16T17:34:20.024Z",  
3   "code": "sucess",  
4   "expr": "6*3",  
5   "res": "18",  
6   "id": 1  
7 }
```

6.0.1 Configuration

- Download ELK stack tools - Elasticsearch, Kibana and Logstash on your local machine, alternatively these can also be used on the cloud.
- Login using your Elastic Stack account.
- Upload your logfile in your project homepage.

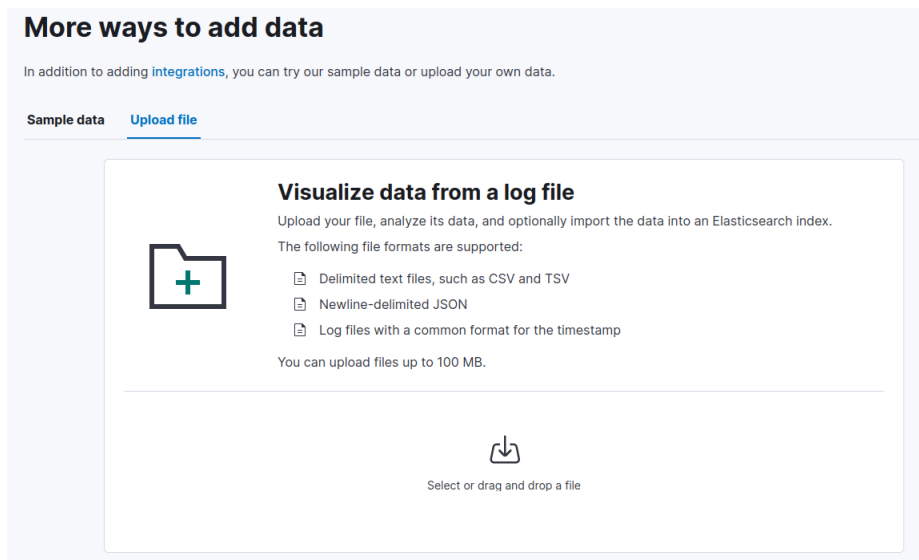


Figure 12: Upload log file

- Click on *OverrideSettings* on the next page and specify a grok pattern to filter the log file and turn it into structured data. Then click on *Import*.

Summary	
Number of lines analyzed	51
Format	semi_structured_text
Grok pattern	%{QUOTEDSTRING:field}: "%(TIMESTAMP_ISO8601:timestamp)"\n %{QUOTEDSTRING:field2}: %{QUOTEDSTRING:status},\n %{QUOTEDSTRING:field4}: %{QUOTEDSTRING:expression},\n %{QUOTEDSTRING:field6}: %{QUOTEDSTRING:result},\n %{QUOTEDSTRING:field8}: %{INT:id}\n \},\n {\n
Time field	timestamp
Time format	ISO8601
<div>Override settings</div> <div>Analysis explanation</div>	

Figure 13: Grok pattern

- Specify a name for the index and proceed by clicking on **Import**.

log(2).json

Import data

[Simple](#)[Advanced](#)

Index name

calculator

☒ Create data view

Import

Figure 14: Index Naming

- Now Create a dashboard using Kibana using the fields of your choice, to visualize and monitor the application.

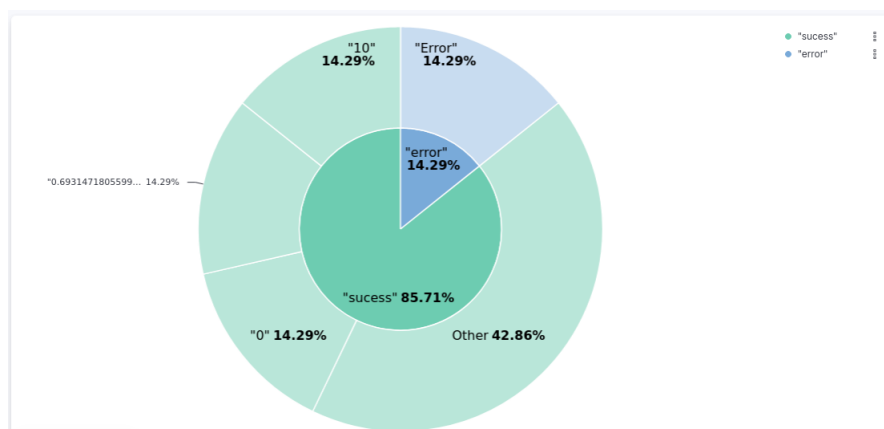


Figure 15: Sample Dashboard