Calculator Mini Project
MT2022170

Github Link : https://github.com/Prathviraj-B-N/calculatorJenkins

We will be creating a calculator application using ReactJS framework for the frontend, Vite for the build process, npm for the package management and running our scripts, Jenkins and Ansible for continuous integration and deployment, ngrok for secure tunnels and Docker for containerization.

Devops

DevOps is a set of practices that aims to bring together the development (Dev) and operations (Ops) teams in an organization to improve collaboration, communication, and integration between them. It involves the use of automation tools, processes, and culture to achieve faster software delivery, improved quality, and greater reliability.

DevOps emphasizes the need for cross-functional teams to work together throughout the entire software development life cycle, including planning, coding, testing, deployment, and monitoring. By breaking down silos between development and operations, DevOps aims to enable continuous integration and delivery (CI/CD), which allows organizations to release software updates more frequently and reliably.

Some of the key principles of DevOps include automation, continuous testing and integration, monitoring and feedback, and collaboration and communication. The adoption of DevOps has become increasingly important for organizations looking to remain competitive in today's fast-paced digital landscape.

Why Devops?

1. **Improved Collaboration**: DevOps brings together cross-functional teams, such as development, operations, and quality assurance, to work collaboratively and align their goals. This leads to better communication, collaboration, and a shared understanding of the software development process, which can result in higher-quality software products.

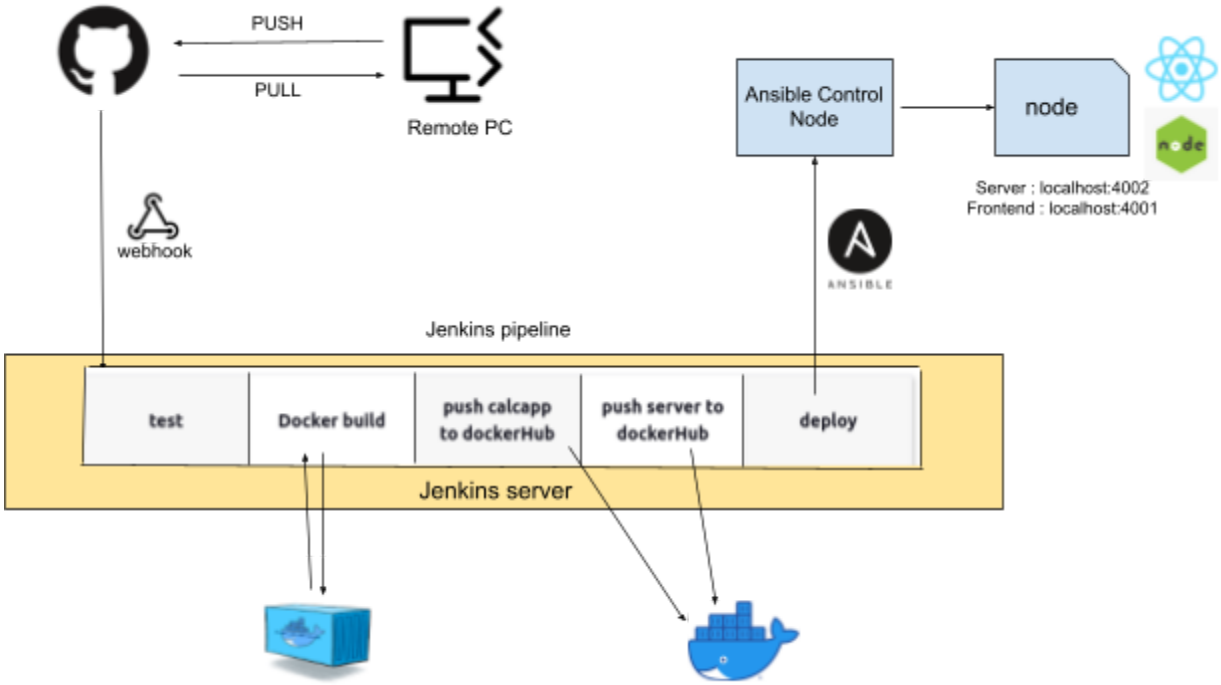
2. **Faster Time-to-Market**: With DevOps, software development teams can deliver new features and updates to users more quickly and efficiently. By using continuous integration and continuous delivery (CI/CD) practices, DevOps teams can automate the testing and deployment processes, allowing for faster and more frequent releases.
3. **Improved Quality**: DevOps emphasizes the importance of continuous testing and monitoring throughout the software development lifecycle. This ensures that any issues or bugs are caught early and fixed before they can cause major problems. This, in turn, leads to higher-quality software products and better user experiences.
4. **Increased Efficiency**: DevOps automates many of the repetitive and time-consuming tasks associated with software development, such as testing, building, and deploying. This frees up developers to focus on more important tasks, such as coding and innovation, which can lead to increased efficiency and productivity.
5. **Better Business Results**: By implementing DevOps practices, organizations can realize a range of business benefits, such as improved customer satisfaction, increased revenue, and reduced costs. DevOps can also help organizations respond more quickly to changing market conditions and customer needs, giving them a competitive advantage in their industry.

Tech Stack used:

Frontend	**React / HTML / CSS**
Backend	**JS / Express js / node js**
CI/CD	**Jenkins**
Containerization	**Docker**

Configuration Management	**Ansible**
Tunneling	**ngrok**
Logging	**Winston**
Testing	**Jest**
Deployment	**localhost**

Devops Architecture



Steps

1. Write code
2. Push it to github
3. GitHub webhook will send the changed payload to jenkins server
4. Jenkins will run the following stages
    a. **Test**

```
stage('test') {
      steps {
        dir("server") {
          sh "pwd"
          sh 'npm i'
          sh 'pm2 --name server start npm -- start'
          sh 'npm test'
          sh 'pm2 delete 0'
          echo 'All test passed!'
        }
      }
    }
```

We will test our calculator backend before we build our Docker image so that bad code doesn't get deployed.

i. Install node packages
ii. Start the service in background using pm2 package
iii. Run tests
iv. Stop the service

b. **Docker Build**

Build both the frontend and backend containers

```
stage('Docker build') {
      steps {
        sh 'docker build -f Dockerfile -t calcapp .'
        sh 'docker build -f server/Dockerfile -t server ./server'
      }
    }
```

c. Push images to dockerHub

We will push both the containers to DockerHub so that we can pull it later for deployment

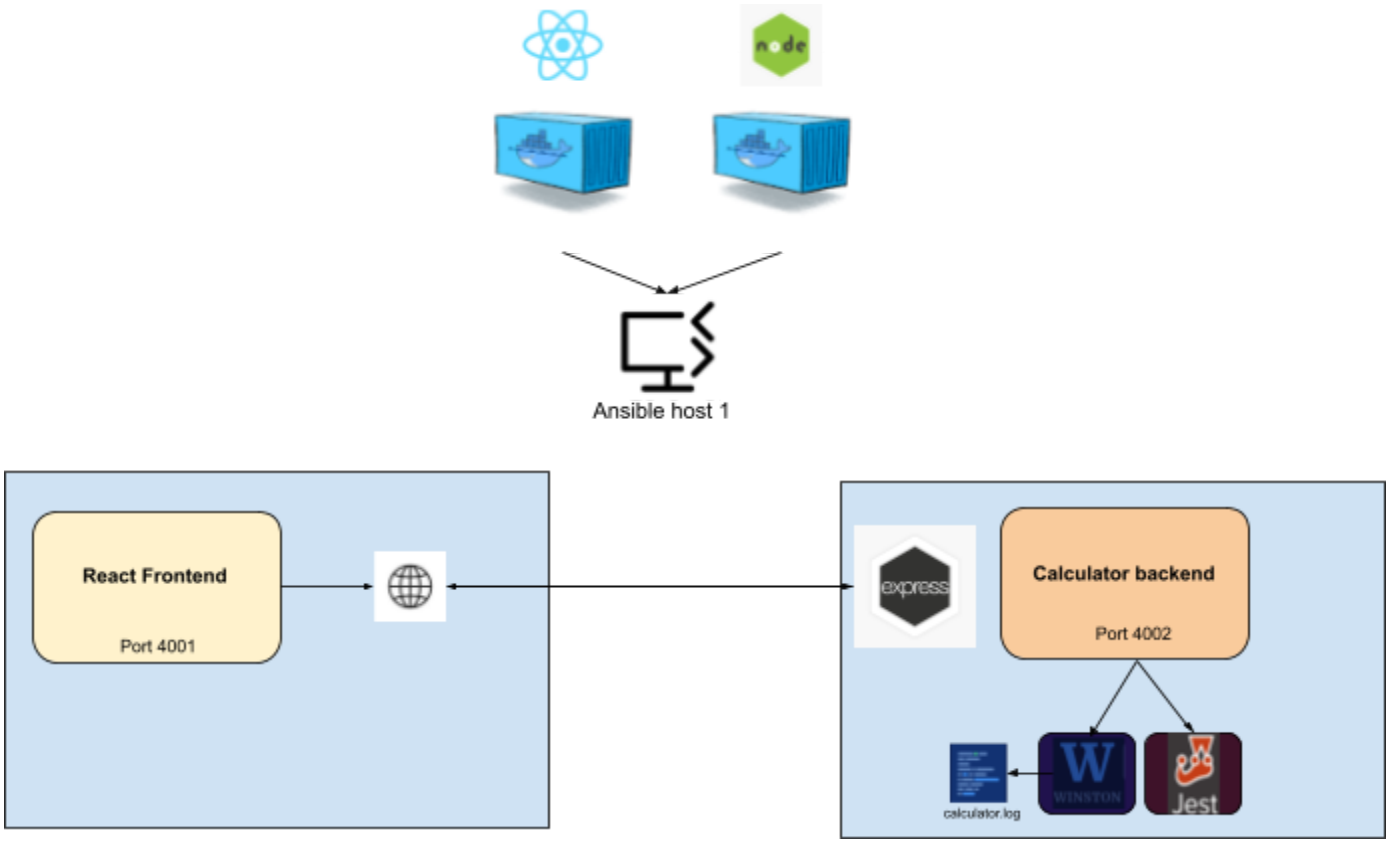
```
stage('push calcapp to dockerHub') {
      steps {
        sh 'docker tag calcapp prathvirajbn/calcapp'
        withDockerRegistry([ credentialsId: 'dockerHubCreds', url: '' ]) {
          sh 'docker push prathvirajbn/calcapp:latest'
        // docker run -p 4001:3000 calcapp
        }
      }
    }
    stage('push server to dockerHub') {
      steps {
        sh 'docker tag server prathvirajbn/server'
        withDockerRegistry([ credentialsId: 'dockerHubCreds', url: '' ]) {
          sh 'docker push prathvirajbn/server:latest'
        // docker run -p 4002:4002 server
        }
      }
    }
```

d. Deploy using ansible

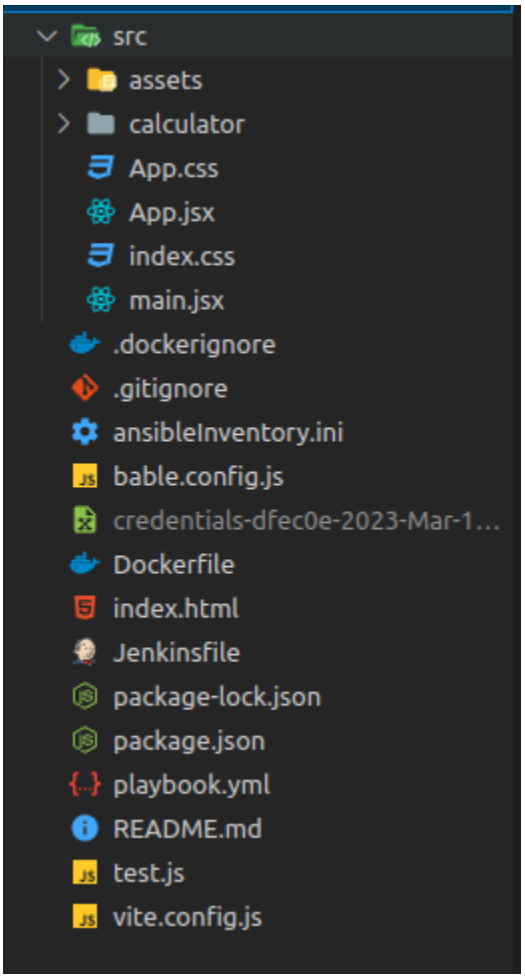
```
stage('deploy') {
      steps {
        sh
'ansible-playbook playbook.yml -i ansibleInventory.ini'
      }
    }
```

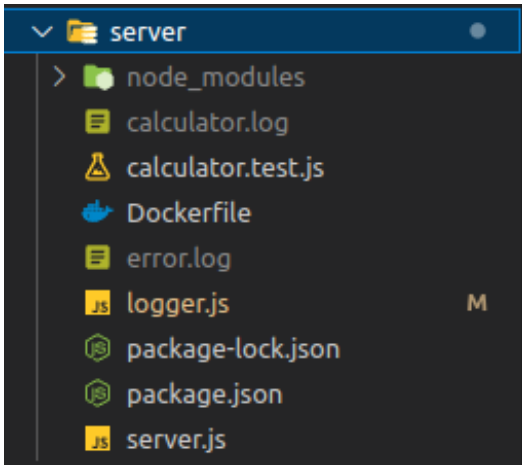
System Design

We will have 2 containers one for frontend and one for backend service.Both the containers will be deployed on the same server (localhost) and communication will happen through ports 4001 and 4002 for frontend and backend respectively.



Project Structure





1. Introduction

To build a calculator project, we will use React as our front-end library, HTML and CSS to create the user interface. On the back-end, we will use Express.js and Node.js to handle the server-side logic, Winston logger for logging and Jest for testing.

To automate the deployment process, we will use Jenkins as our Continuous Integration and Continuous Deployment (CI/CD) tool, which will help us in building, testing, and deploying the application.

To containerize the application, we will use Docker, which will help us in creating a container for our application and deploying it in any environment. We will use Ansible

as our configuration management tool to automate the deployment of our Docker container to multiple environments.

To establish webhooks, we will use ngrok, which will help us in creating a secure tunnel between our local machine and the remote repository.

To log information, errors and debug our application, we will use Winston, which is a logging library for Node.js that helps us in logging errors, warnings, and information about our application.

1. **React JS** is a popular JavaScript framework for building dynamic user interfaces. It allows for efficient rendering of components and enables building reusable UI components. We will use ReactJS to create the frontend of our calculator application.
2. **Jenkins** is an open-source automation server that helps in continuous integration and deployment of software applications. It allows for automating the build, testing, and deployment processes. We will use Jenkins to automate the building and testing of our application.
3. **Ansible** is an open-source automation tool that helps in automating infrastructure management. It allows for automating repetitive tasks and ensures consistency in the deployment process. We will use Ansible to automate the deployment of our application.
4. **Ngrok** is a secure tunneling service that allows for exposing local web servers to the internet securely. We will use ngrok to securely expose jenkins to the internet to use github webhook.
5. **Docker** is a popular containerization platform that allows for packaging an application and its dependencies into a container. It provides a consistent environment for running the application, making it easy to deploy across different platforms. We will use Docker to containerize our application and ensure consistent deployment across different environments.

2. Setup Project

1. Clone the Repository

This will clone the repository into local machine so that we can work on it.

```
$ git clone
https://github.com/Prathviraj-B-N/calculatorJenkins.git
```

2. Setup your own github repo

```
$ git remote set-url origin http://github.com/YOU/YOUR_REPO
```

3. Now the repository is in your local machine. Go to the project directory.

```
$ cd calculatorJenkins
```

4. Run the Server **(***In project directory " /server "* **)**

```
$ npm install
$ node server.js
```

5. Run Calculator Frontend **(***In project directory " / "* **)**
    1. Install node modules

    ```
    $ npm install
    ```

    2. Run the Server

    ```
    $ npm run dev
    ```

    ```
    asus@asus-TUF-Gaming-FX505DT-FX505DT:~/Documents/projects/calculatorJenkins$ npm run dev

    > calculatorjenkins@0.0.0 dev
    > vite


      VITE v4.1.4  ready in 419 ms

      ➜  Local:   http://localhost:5173/
      ➜  Network: use --host to expose
      ➜  press h to show help
    ```

3. Setup Jenkins

We will use Jenkins Pipeline for our CI/CD

1. Reusability and Modularity: With Jenkins pipeline, you can define your entire build, test, and deployment process as a single script, which can be easily shared and reused across multiple projects.
2. Version Control: You can store your pipeline script in a version control system like Git, enabling you to track changes to your pipeline over time and collaborate with other team members.
3. Visibility and Transparency: Jenkins pipeline provides a graphical representation of your pipeline, which gives you visibility into the status of each stage in the pipeline and enables you to troubleshoot issues quickly.
4. Flexibility: Jenkins pipeline offers a high degree of flexibility in terms of how you structure your pipeline. You can define sequential or parallel stages, add conditional logic, and trigger jobs based on events or schedules.
5. Integration with other tools: Jenkins pipeline integrates with a wide range of tools and services, including source code management systems, testing frameworks, and deployment platforms, enabling you to build a complete end-to-end CI/CD pipeline.

1. Configure Jenkins

1. Goto Jenkins Dashboard > New Item > Pipeline
2. Enter your Github URL in project url field

Dashboard > calculatorMiniProject > Configuration

Configure

- General
- Advanced Project Options
- Pipeline

General

Description

[Plain text] Preview

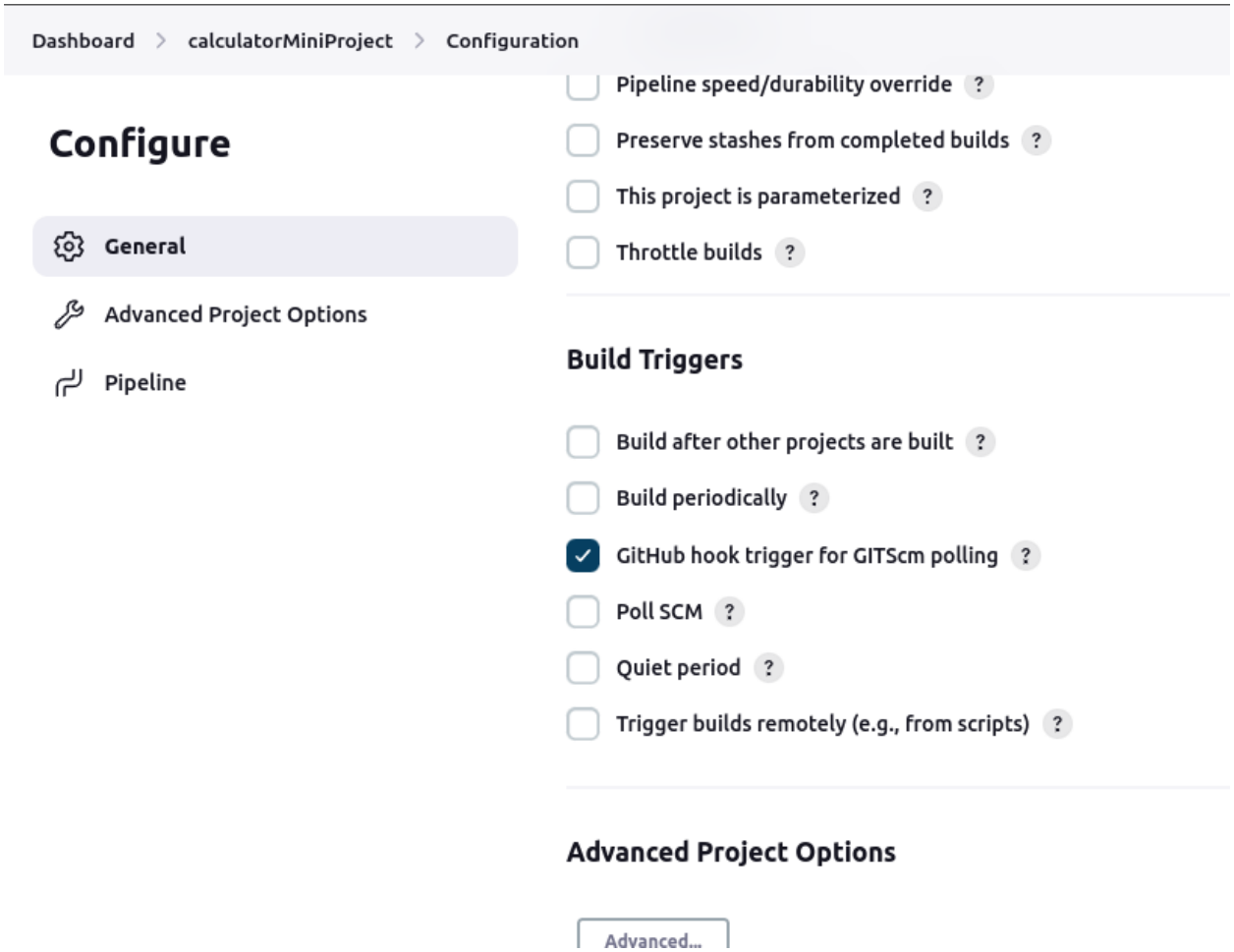
- [ ] Discard old builds ?
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url ?

https://github.com/Prathviraj-B-N/calculatorJenkins.git

Advanced...

3. Add Github hook Build triggers

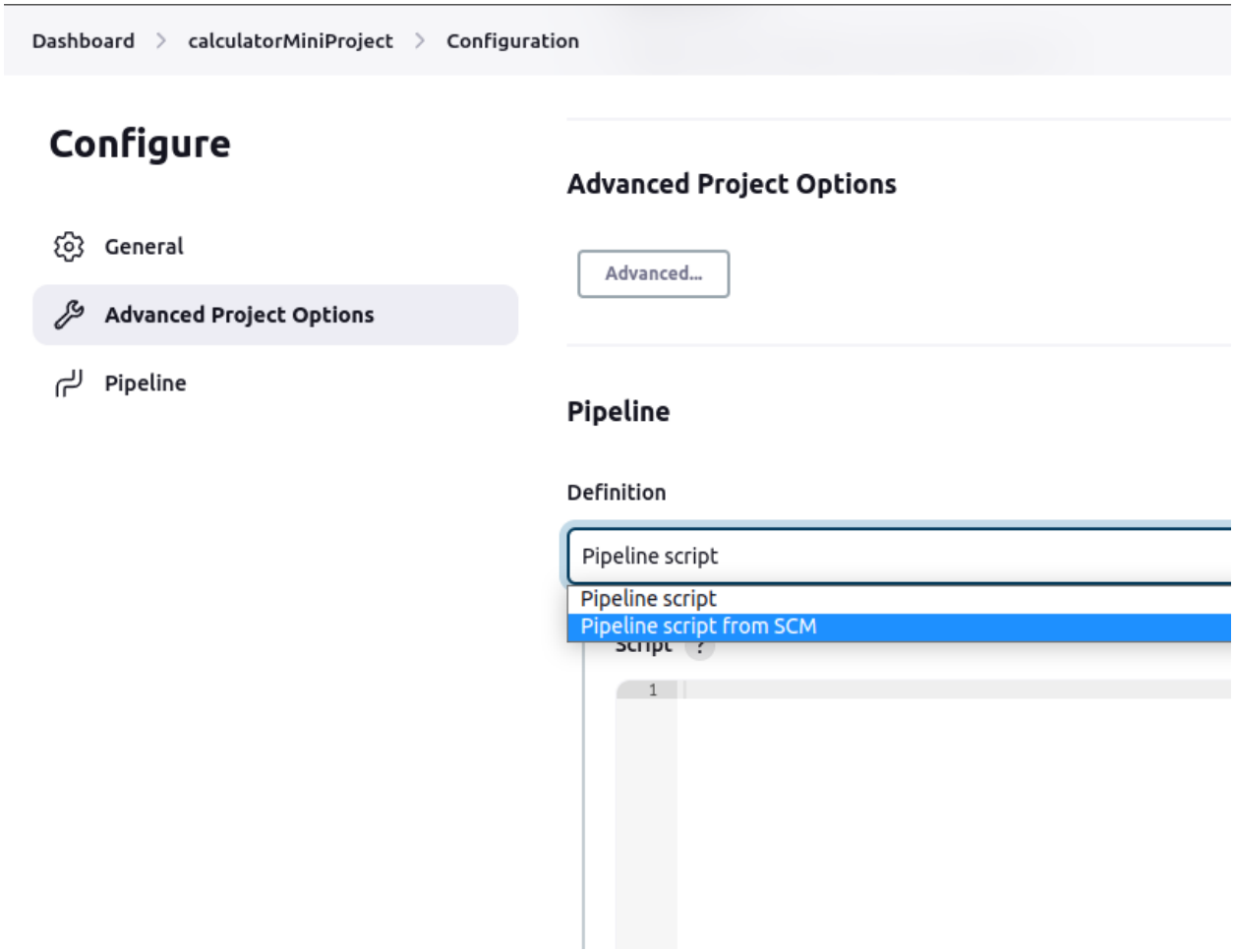


GitHub hook trigger is a webhook that can be used in Jenkins to trigger a build in response to a specific event that occurs in a GitHub repository. *We will use PUSH events only.*

GitHub hook trigger will help us in:

1. **Automated Builds**: By using GitHub hook trigger, you can set up Jenkins to automatically trigger a build whenever a specific event occurs in a GitHub repository, such as a new commit being pushed.
2. **Real-time updates**: With GitHub hook trigger, you can ensure that Jenkins receives real-time updates from your GitHub repository, so your builds are always up to date.
3. **Efficiency**: GitHub hook trigger eliminates the need for manual intervention to trigger builds in Jenkins, making the development process more efficient.

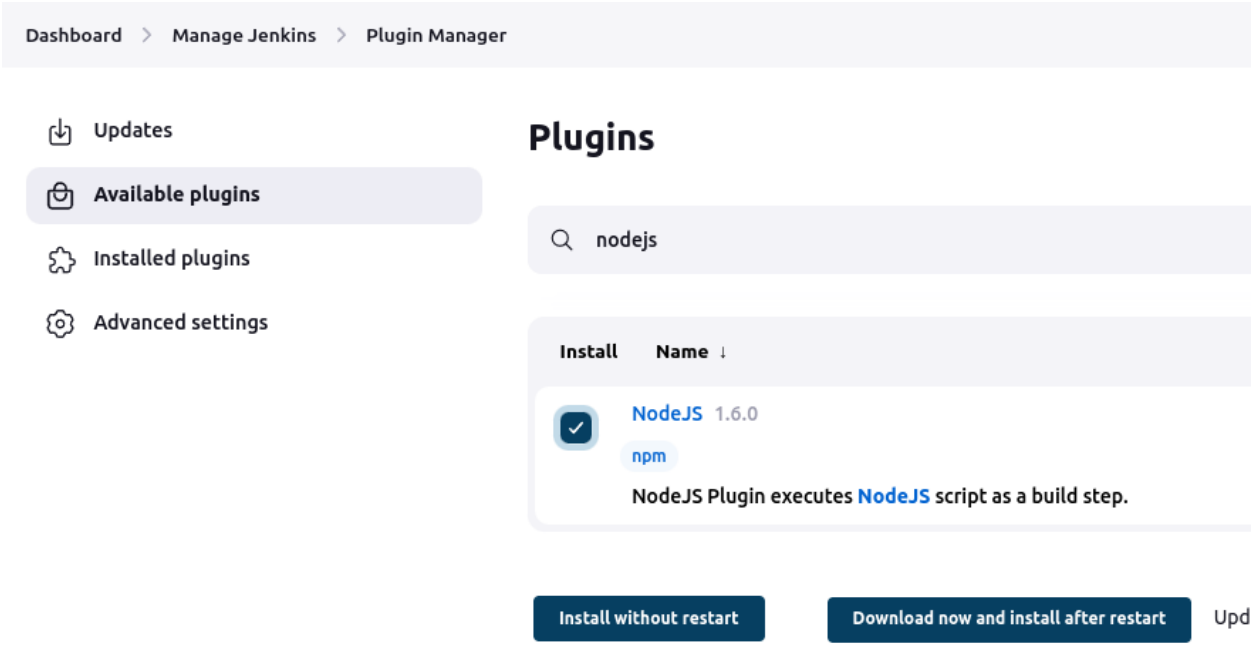
4. **Flexibility**: GitHub hook trigger offers a high degree of flexibility in terms of which events trigger a build. You can choose to trigger builds for specific branches, tags, pull requests, and more.
5. **Integration**: GitHub hook trigger integrates seamlessly with Jenkins, allowing you to build a complete CI/CD pipeline that automates the entire development process.



**We will be keeping our Jenkins pipeline file along with our code so that we can have a version control on it. So we need to select 'pipeline script from SCM (Source Code Manager).*

4. Install Plugins

Since we are using nodejs we need to install it to run npm and node scripts



4. Pipeline Design

Test -> Docker Build -> publish containers to DockerHub -> deploy to target machines

5. Setup Github webhook

1. Expose Jenkins to the internet using ngrok
    a. Go to https://ngrok.com/download ,login and download ngrok for your device.
    b. Extract the downloaded file
    c. Add authtoken to ngrok config using

```
$ ngrok config add-authtoken <token>
```

    d. Start a tunnel

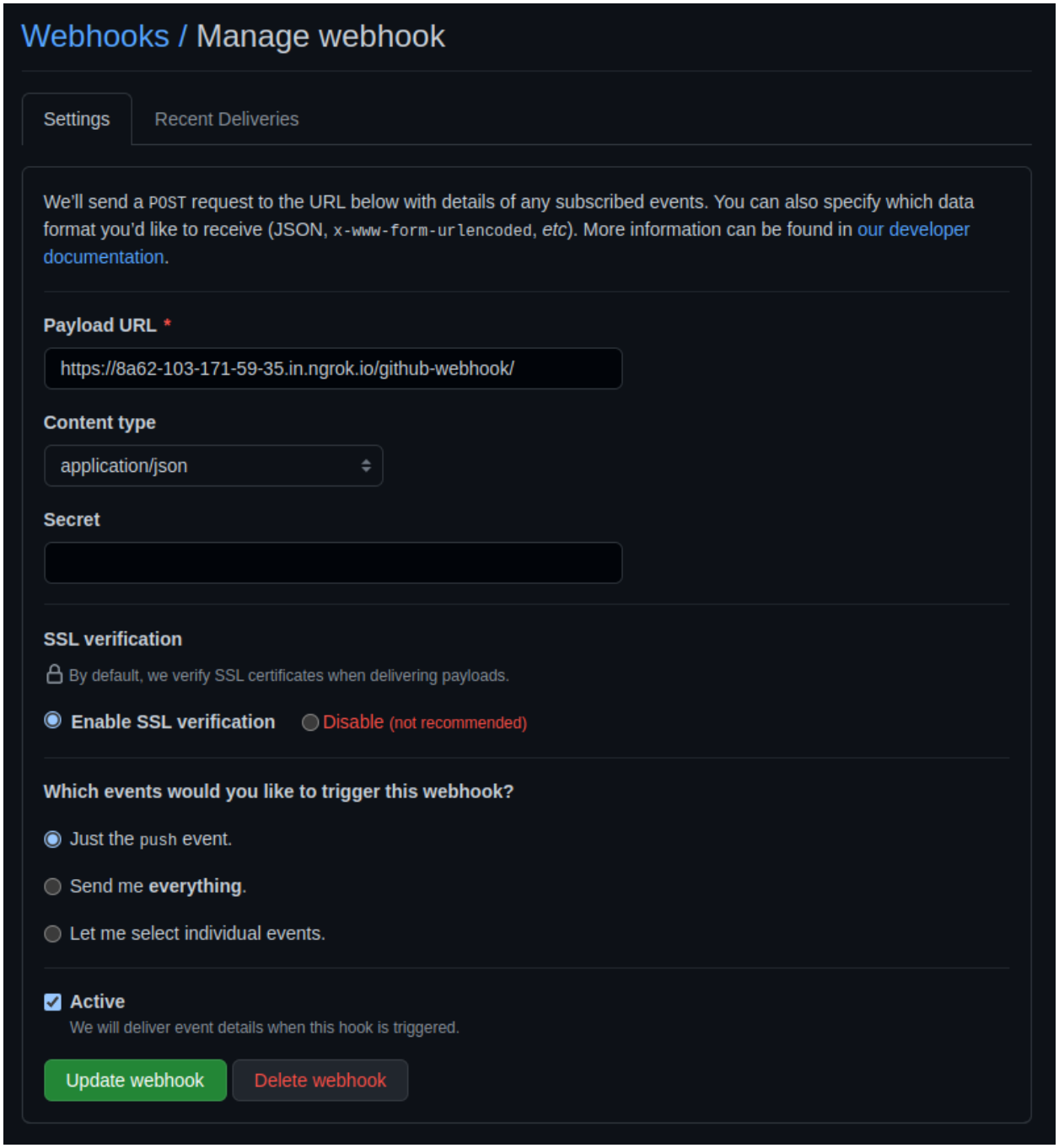
```
$ ngrok http 8080
```

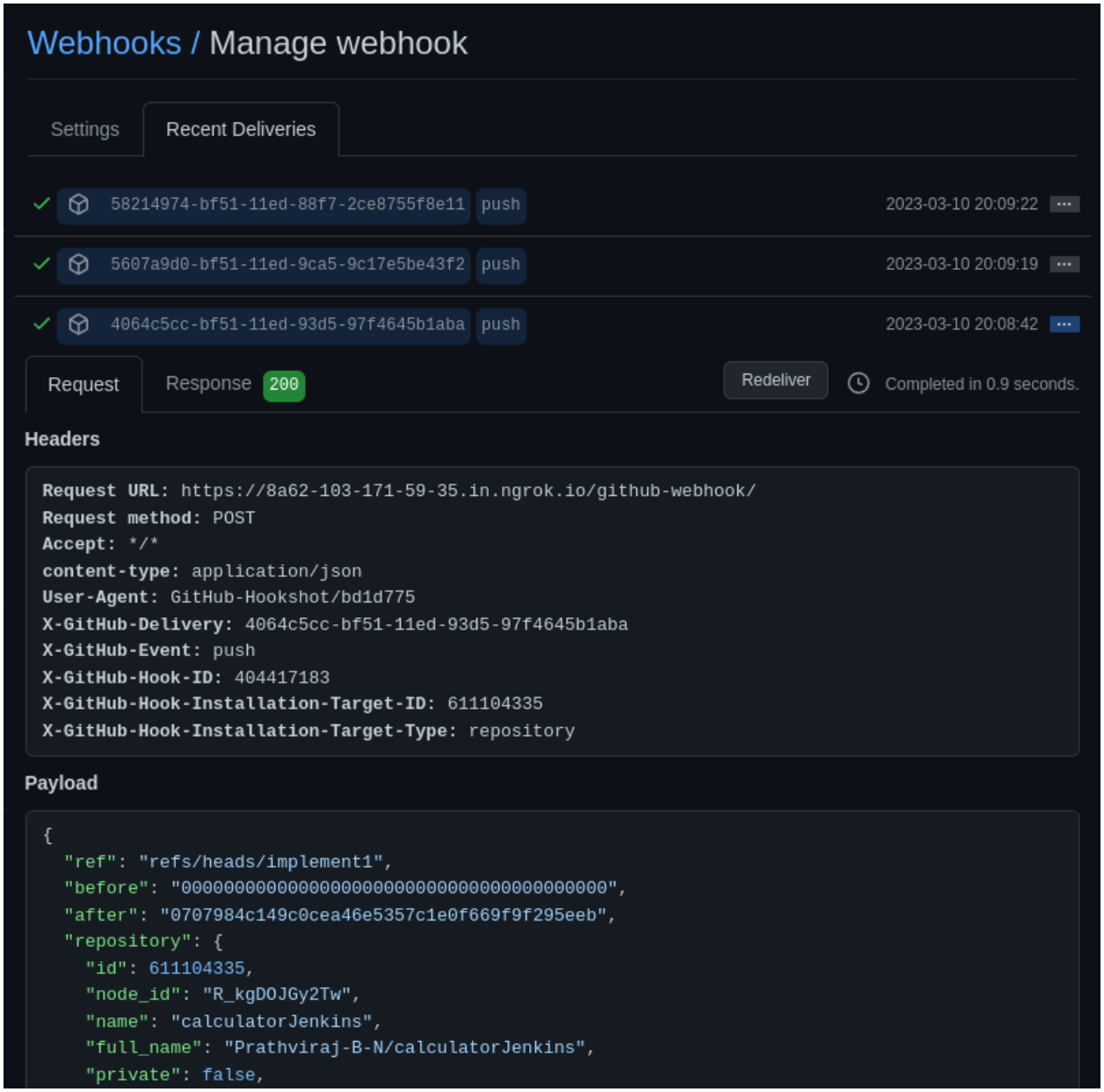
**We are exposing port 8080 in our localhost to a public ip using ngrok so that GitHub hooks can send some data to our jenkins*

```
ngrok

Announcing ngrok-go: The ngrok agent as a Go library: https://ngrok.com/go

Session Status                online
Account                       prathviraj b (Plan: Free)
Version                       3.1.1
Region                        India (in)
Latency                       48ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://8a62-103-171-59-35.in.ngrok.io -> http://localhost:8080

Connections                   ttl     opn     rt1     rt5     p50     p90
                              47      1       0.00    0.04    0.31    30.11

HTTP Requests
-------------

GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
GET /job/calculatorMiniProject/wfapi/runs 200 OK
```

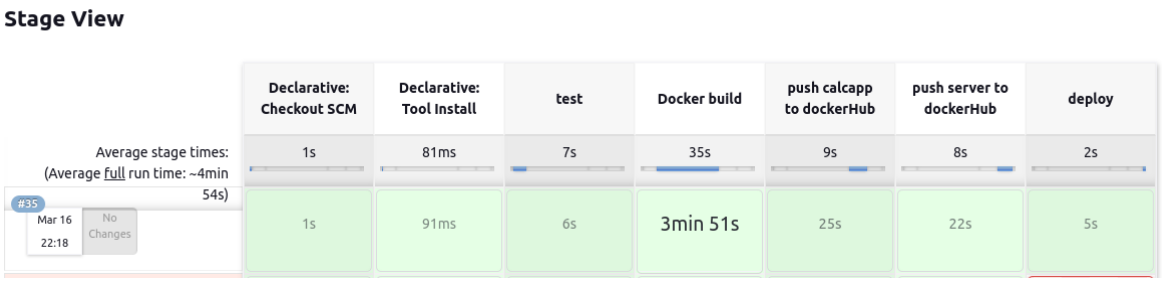
2. Go to GitHub repo you have forked
3. Settings > Webhooks > Add webhook
    a. Paste Public URL given by ngrok toPayload URL field
    b. Change Content type to application/json
    c. Create webhook

**Remember to append "/github-webhook/" to public URL*


Webhooks / Manage webhook
Settings
Recent Deliveries
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 *
https://8a62-103-171-59-35.in.ngrok.io/github-webhook/
Content type
application/json
Secret
SSL verification
By default, we verify SSL certificates when delivering payloads.
Enable SSL verification
Disable (not recommended)
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.
Update webhook
Delete webhook



Now whenever a push event happens, webhook will notify Jenkins. Jenkins will run the pipeline automatically.



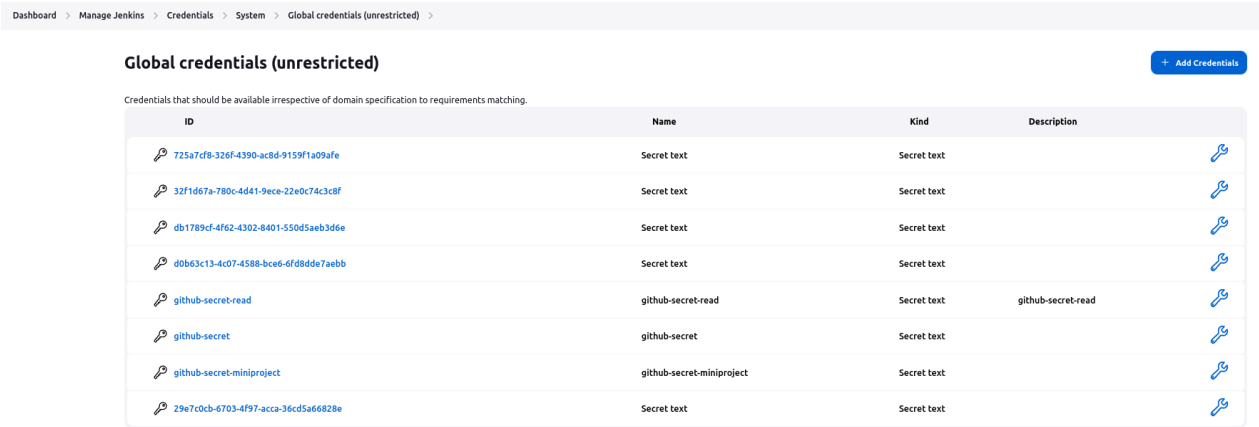
6. Containerization

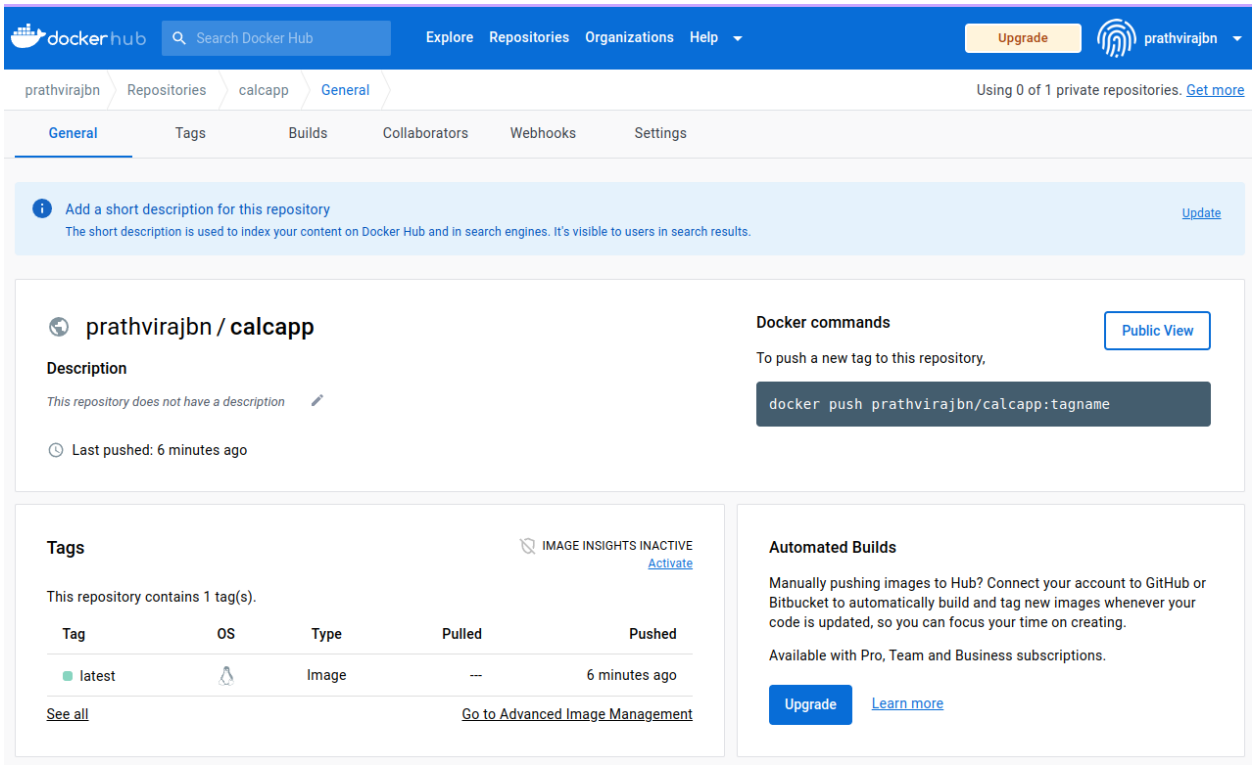
To give jenkins, permission to run docker

```
$ sudo usermod -a -G docker jenkins
$ sudo service jenkins restart
```

We will push docker images to docker hub, Pushing Docker images to Docker Hub allows for centralized storage, easy deployment, version control, and automated builds. It streamlines the development and deployment process and provides a centralized location for sharing images.

1. Add DockerHub credentials to Jenkins





7. Ansible

Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them. Ansible then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update
$ sudo apt-get install ansible
$ ansible –version
```

If you get error related to language format, do the following

```
$ sudo nano /etc/default/locale
```

Paste these 2 lines

```
LANG=en_US.utf8
LC_CTYPE=en_IN.utf8
```

```
$ sudo update-locale LANG=en_IN.utf8 LC_CTYPE=en_IN.utf8
```

Ansible uses an **inventory file** to keep track of which hosts are part of your infrastructure, and how to reach them for running commands and playbooks.

```
$ sudo adduser ansible
```

Ansible will be our controlNode so give this user sudo permissions

```
$ visudo
```

Add "ansible ALL=(ALL:ALL) ALL" below %sudo

If you use EC2:

```
$ vi /etc/ssh/sshd_config
```

Here set passwordAuth to yes

```
$ service ssh restart
```

*****Use same steps and create host1 but don't give to sudo privileges***

SSH into ansible and host:

1. SSH into ansible user

   ```
   $ ssh ansible@localhost
   ```

2. In ansible@localhost $ ssh-keygen

   ```
   $ ssh-copy-id ansible-host1@localhost
   ```

   **Public key will be copied to host and ssh ansible-host1@localhost will automatically connect without password*

```
$ ssh ansible@localhost
$ exit
```

3. Define your IP's in inventory file(ansible@loclhost)
    1. Create a file called **ansibleInventory.ini**
    2. Add your hosts to this file

```
ansible-host1@localhost ansible_user=ansible-host1

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

4. Check connection status:

```
$ ansible -m ping -i ansibleInventory.ini all
```

```
asus@asus-TUF-Gaming-FX505DT-FX505DT: ~/Do
asus@asus-TUF-Gaming-FX505DT-FX505DT:~/Docume
ansible -m ping -i ansibleInventory.ini all
ansible-host1@localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

5. Create a playbook

```yaml
---
- hosts: localhost
  tasks:
    - name: Pull Docker image
      docker_image:
        name: prathvirajbn/calcapp
        tag: latest
        source: pull

    - name: Pull Docker image server
      docker_image:
        name: prathvirajbn/server
        tag: latest
        source: pull

    - name: Create Docker containe
      shell: "docker run -d -p 4001:3000 prathvirajbn/calcapp"

    - name: Create Docker container server
      shell: "docker run -d -p 4002:4002 prathvirajbn/server"
```

6. Run Playbook

```
$ sudo ansible-playbook playbook.yml -i ansibleInventory.ini
```

```
asus@asus-TUF-Gaming-FX505DT-FX505DT: ~/Documents/projects/...
*******************************************************************************
ok: [localhost]

TASK [Pull Docker image] ******************************************************
*******************************************************************************
ok: [localhost]

TASK [Create Docker container] ************************************************
*******************************************************************************
changed: [localhost]

PLAY RECAP ********************************************************************
*******************************************************************************
localhost                  : ok=3    changed=1    unreachable=0    failed=0    skippe
d=0    rescued=0    ignored=0

asus@asus-TUF-Gaming-FX505DT-FX505DT:~/Documents/projects/calculatorJenkins$
```

```
stage("deploy") {
  steps {
    sh 'ansible-playbook playbook.yml -i ansibleInventory.ini'
  }
}
```

Now when you commit something, complete test build and deploy will happen

8. Testing *(in project directory ' /server ')*

1. Install required packages

```
$ npm i jest supertest cross-env
```

   I. **Jest** is a framework for testing JavaScript code. Unit testing is the main usage of it.
   II. **Supertest**: Using Supertest, we can test endpoints and routes on HTTP servers.
   III. **cross-env**: You can set environmental variables inline within a command using cross-env.

2. Open your package.json file and add the test script to the scripts.

```
"scripts": {
    "test": "cross-env NODE_ENV=test jest --testTimeout=5000",
    "start": "node server.js",
    "dev": "nodemon server.js"
}
```

   In this case, we're using cross-env to set environment variables, jest to execute test suites, and testTimeout is set to 5000 because certain requests might take a while to finish.

3. Run tests

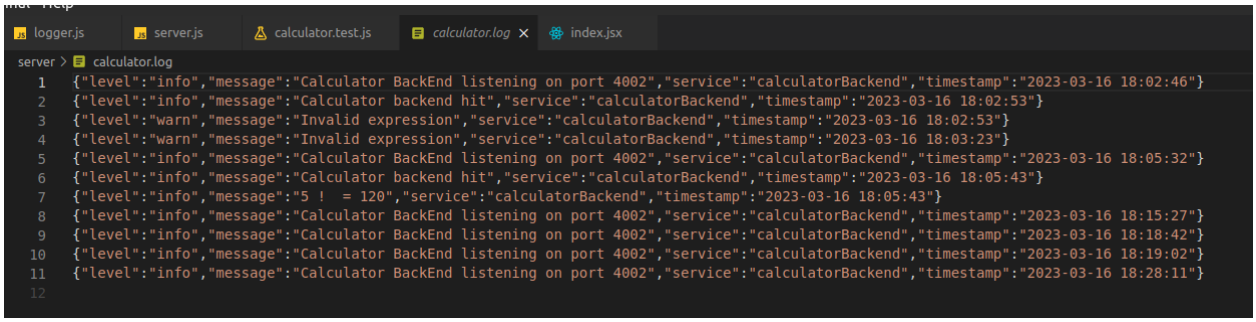
```
asus@asus-TUF-Gaming-FX505DT-FX505DT:~/Documents/projects/calculatorJenkins/server$ npm test

> server@1.0.0 test
> cross-env NODE_ENV=test jest --testTimeout=5000

 PASS  ./calculator.test.js
  GET /api/
    ✓ should return hello world (93 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.382 s, estimated 1 s
Ran all test suites.
```

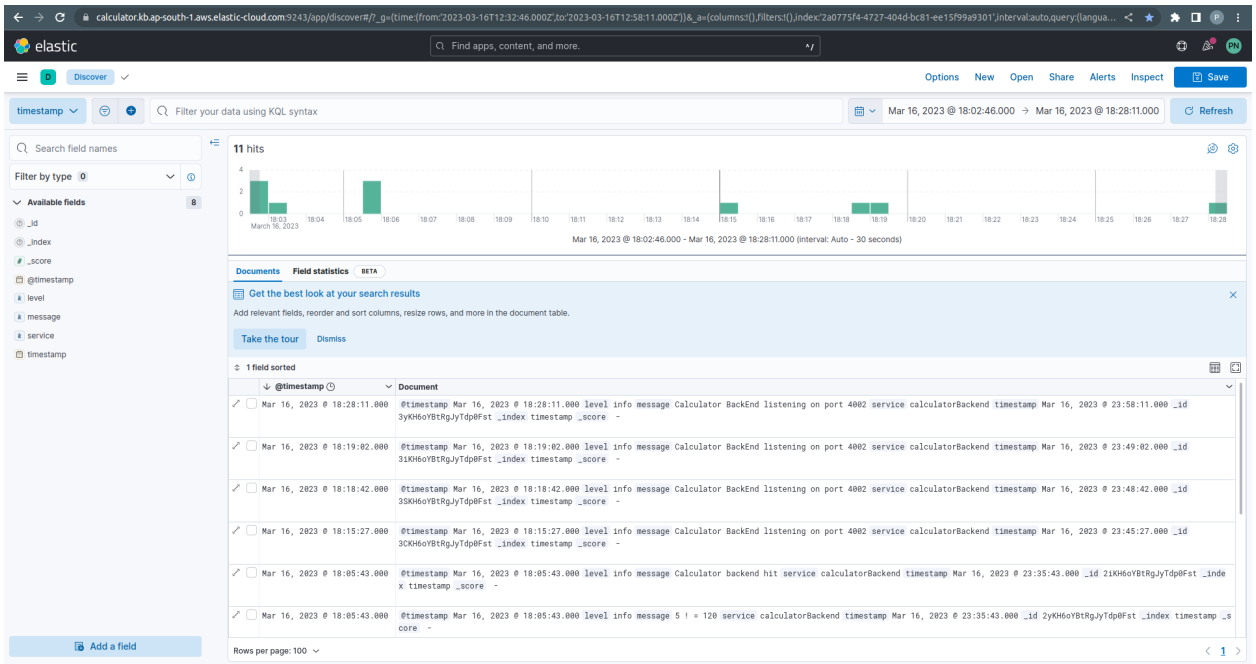
9. Continuous Monitoring

In software development, logging is a crucial process that allows developers to monitor and debug applications as they run. A logger is a tool that enables developers to record and store information about various events and actions that occur within an application. Typically, logging is done by using the console.log() function in JavaScript to write messages to the console. However, relying solely on console.log() can be limiting, especially when dealing with complex applications with multiple components and services.The logger.js file will contain code that defines the logger object, including any custom methods or properties that developers want to include. Instead of using the console.log() function to write messages to the console, developers can use the logger object's log() method.

Once the logger is configured, it can be used throughout the application to generate log messages at various levels of severity, such as info, warn, and error. These messages can then be written to the file specified in the logger configuration, along with any additional metadata that may be useful for debugging and analysis.

Overall, using Winston for logging into a file provides a powerful and customizable way to capture and store log messages generated by an application, which can be useful for troubleshooting issues and gaining insights into application behavior over time.

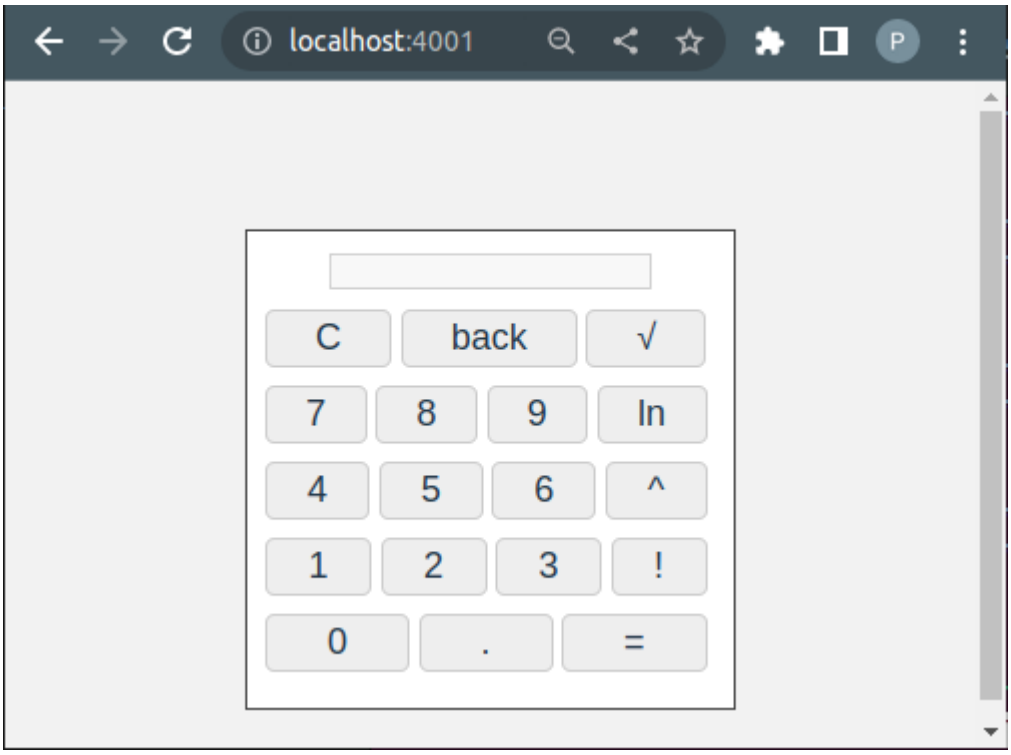




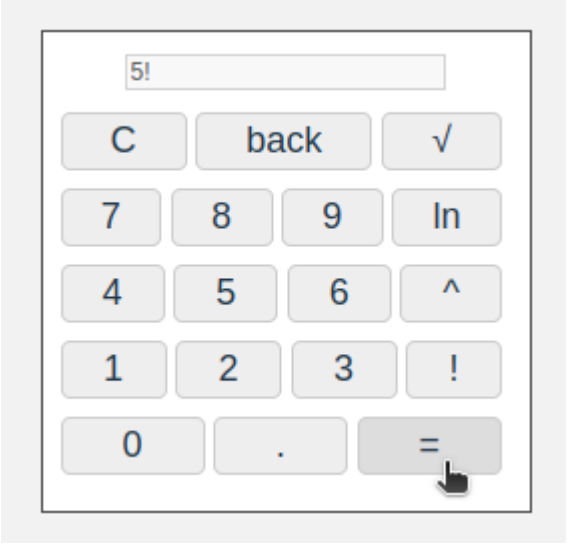
To get the log files from docker container

```
$ docker cp container_name:/app/calculator.log
/path/to/your/project/calculatorJenkins/server/calculator.log
```

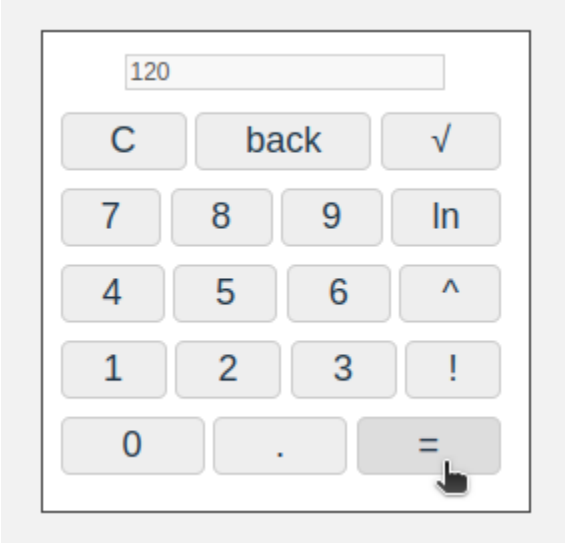
10. Result



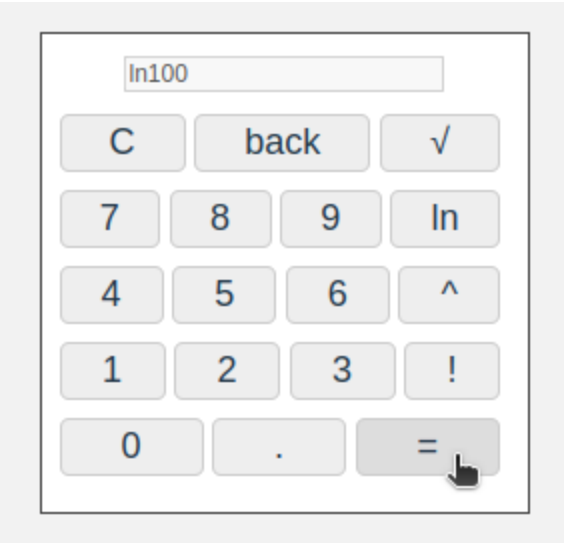
operations

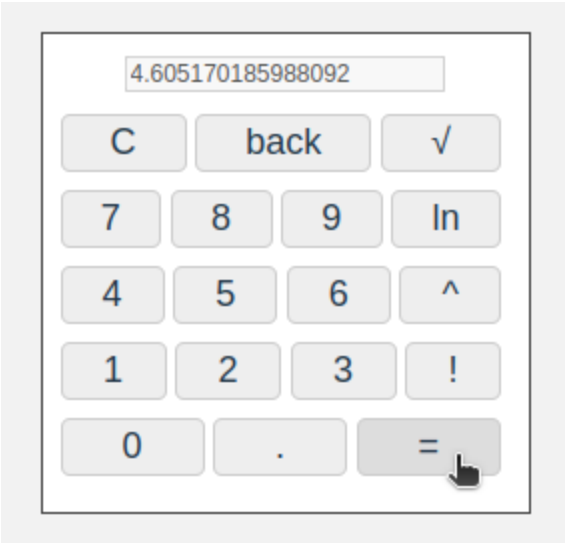


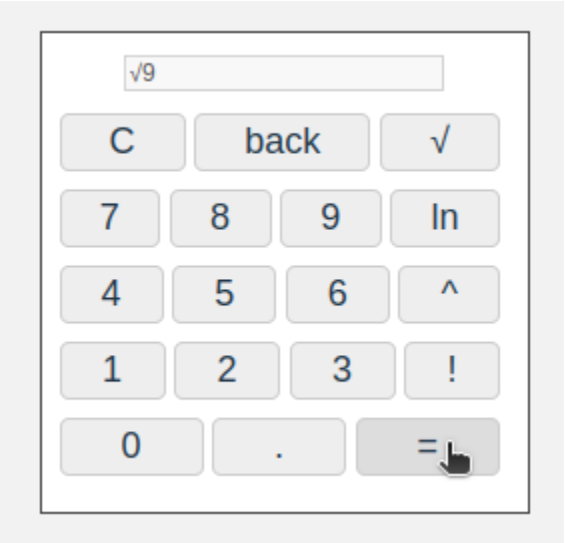
s

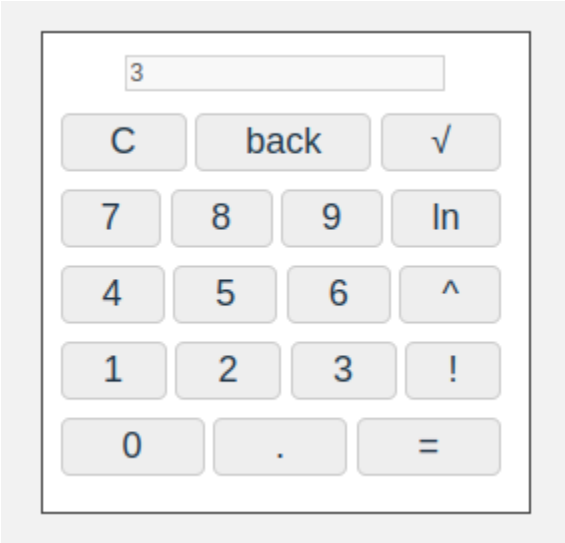


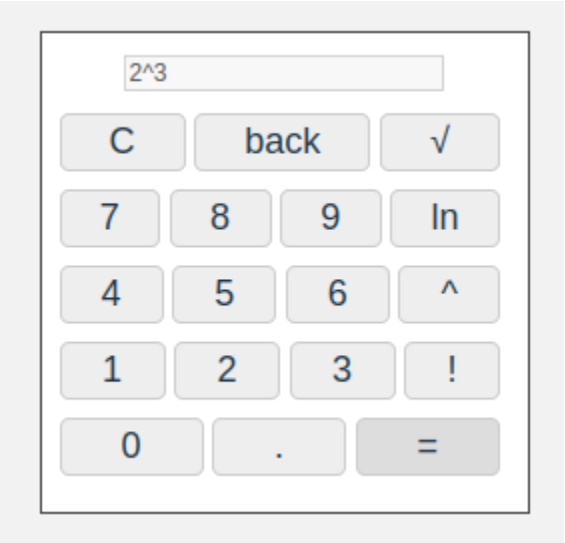
s

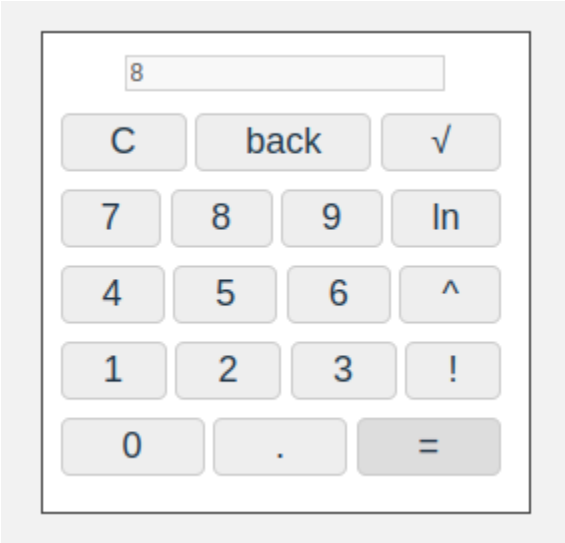

ln100
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=


4.605170185988092
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=


√9
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=


3
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=


2^3
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=


8
C
back
√
7
8
9
ln
4
5
6
^
1
2
3
!
0
.
=