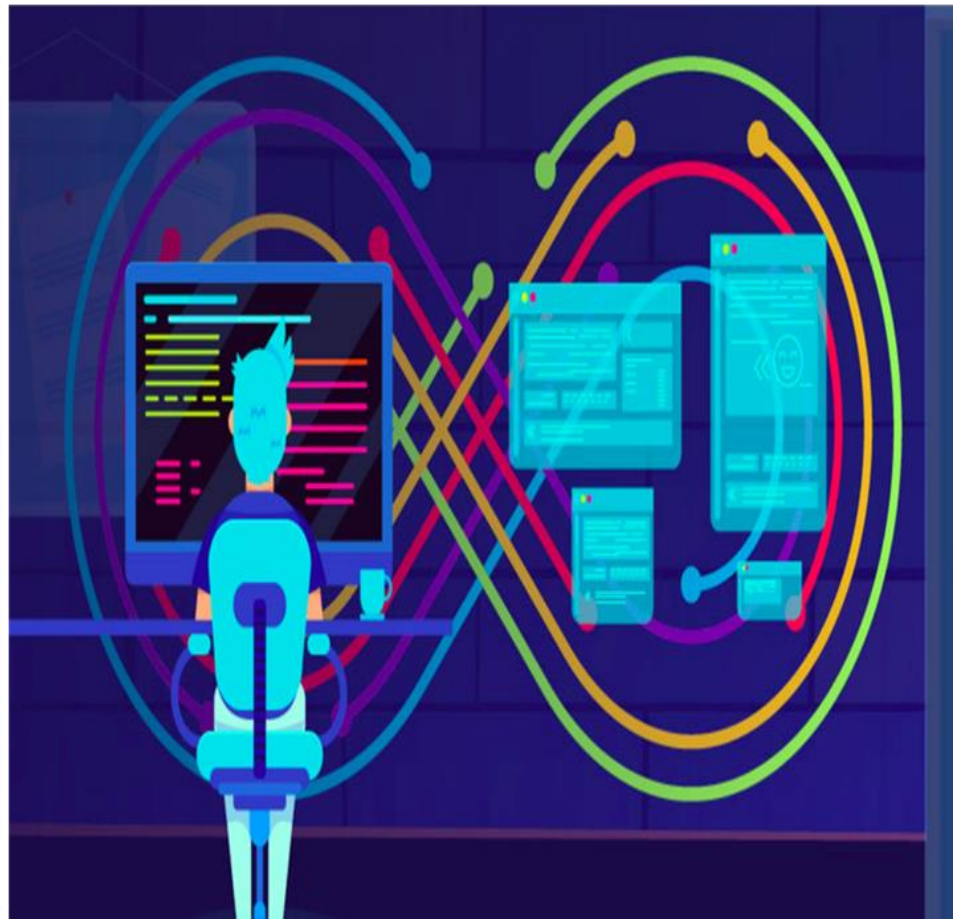
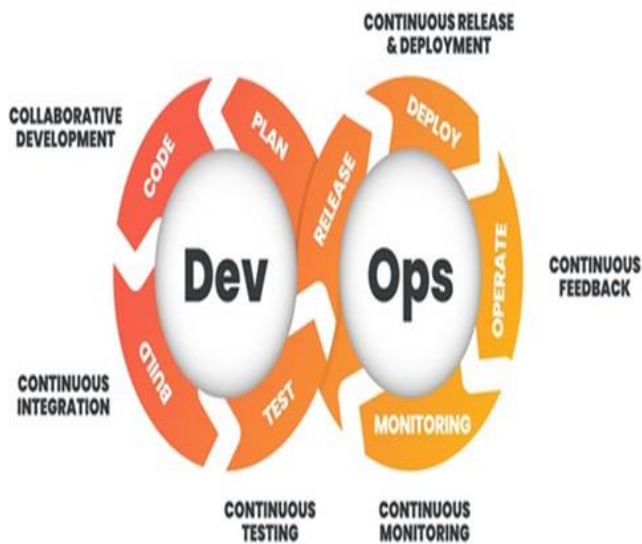


DEV OPS PROCESS

Development and Operations Integrated Process



REPORT:

SCIENTIFIC CALCULATOR USING DEVOPS

RICHA VARMA-MT2021107

What is DevOps?

DevOps can be best explained as people working together to conceive, build and deliver secure software at top speed. DevOps practices enable software developers (devs) and operations (ops) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement.

Stemming from an Agile approach to software development, a DevOps delivery process expands on the cross-functional approach of building and shipping applications in a faster and more iterative manner. In adopting a DevOps development process, you are making a decision to improve the flow and value delivery of your application by encouraging a more collaborative environment at all stages of the development cycle.

DevOps represents a change in mindset for IT culture. In building on top of Agile, lean practices, and systems theory, 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.

Reasons to choose DevOps Methodology

1. Quicker mitigation of software defects

With better communication and collaboration between operations and software development, you can identify and mitigate defects at any stage of the development cycle. The same culture can be applied to Application development, where defects prove costlier.

2. Better resource management

During the application and software development stage, developers and testers are constantly waiting for resources to arrive causing delays in delivery. Agile with DevOps ensures that the app development arrives in testing phase much quicker than existing operations.

3. Reduced human errors

DevOps reduces the chances of human errors during development and operations process by deploying frequent iterations. Lower the application failure rate with multiple deployments in the process in a defined timeline.

4. Enhanced version control

Emphasizing on the individuals and interactions, DevOps allows the developers to leverage on programmable dynamic infrastructure at all stages of software application development cycle. It allows version control and automated coding options.

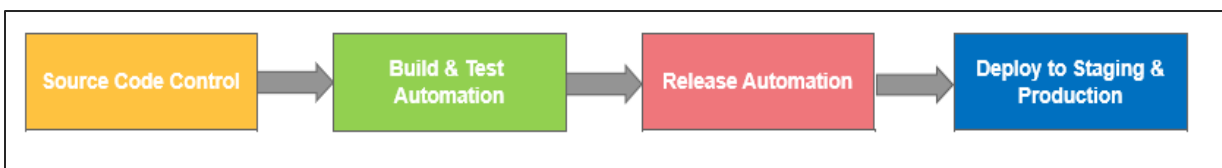
5. Stable operating environment

Stability is the key to any business platform, and DevOps is established to bring stability with reliability. Organizations with DevOps get their deployment 30 times faster than their rivals with 50% lesser chances of failure.

“The primary DevOps goal is to optimize the flow of value from idea to end user. There’s a cultural change that must happen for a company to be successful with DevOps, but the DevOps goal is to make the delivery of value more efficient and effective.”

DevOps Pipeline

In the software development process, a DevOps pipeline consists of different stages to move the code to production. The various stages are as follows:



Stage-1: Source Code Control

In this stage, developers get to work writing applications or updates within an editor. This can be as simple as a text editor or as detailed as an integrated development environment. All code is checked into a centralized source code repository. The pipeline is triggered once the developer commits the code to the Source Code Control system, then the next stage is executed.

Stage-2: Build and Test

This stage involves compiling the code to create a deployable build. After a successful build, automated testing is performed to ascertain that the build meets the required functional, performance, design, and implementation requirements.

Stage-3: Release Automation

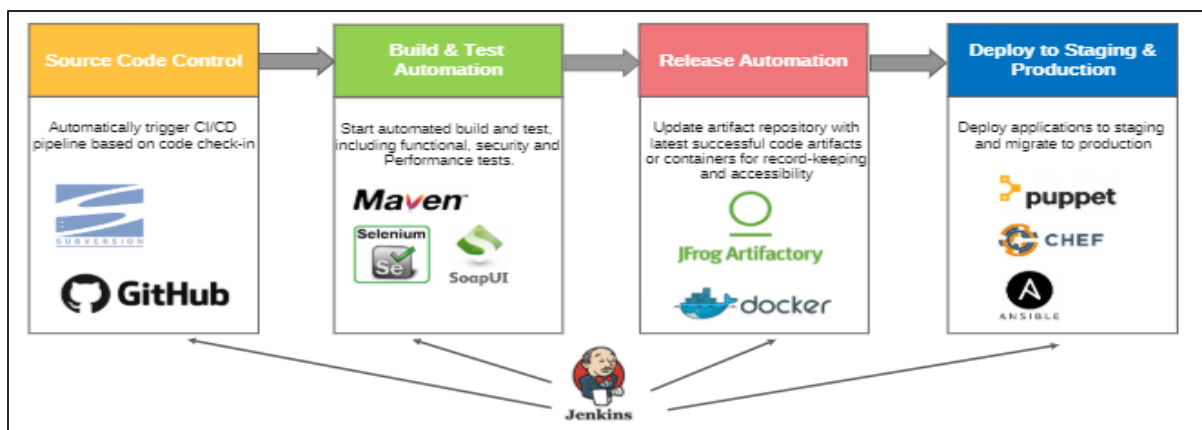
In this stage, the runtime artifact (which is the compiled code) is saved in a repository. The repository may also contain an older version of a runtime artifact which can be redeployed during a rollback process.

Stage-4: Deploy

The last stage is deploying the runtime artifact to the production environment. Once the build has successfully passed through all the required test scenarios, it is ready to deploy to live server.

A DevOps pipeline is focused on automating and connecting these stages of different tasks performed by several teams, such as continuous integration for developers, test automation for testers, code deployment by release managers.

Different Organizations use different tools to build this pipeline. However, the main challenge is connecting all these tools as part of a DevOps pipeline.



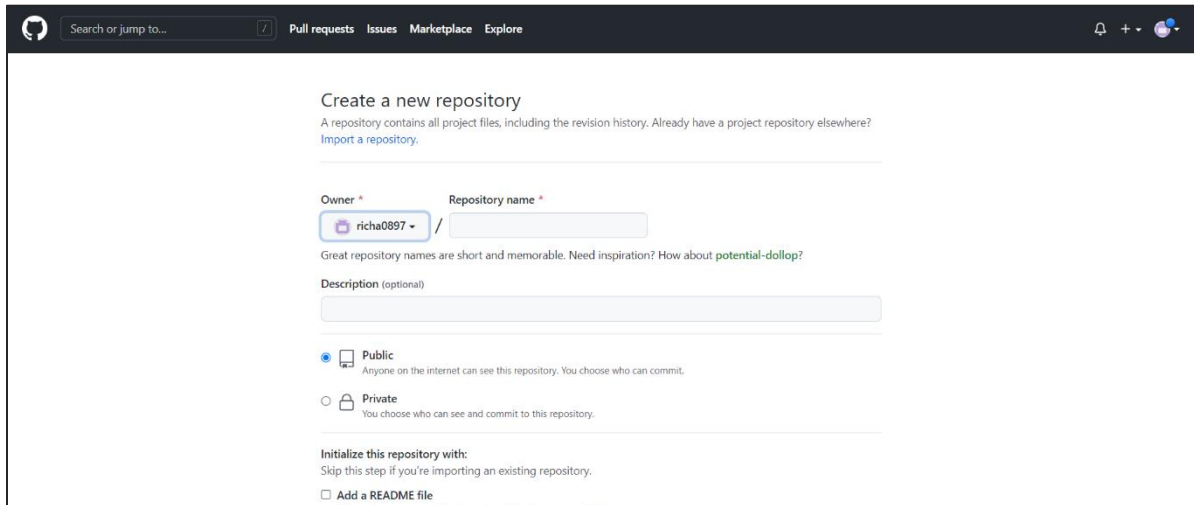
- ✚ For this project, GitHub is used for Source Control Management. A repository is created on GitHub and the latest version of the modified code is pushed to it.
- ✚ We are working on a Java project. Junit is used for writing the test cases and maven build is used for building the project.
- ✚ For our project we have used Jenkins to orchestrate the different stages in the pipeline. Jenkins provides a pipeline where you can utilize a combination of plugins to integrate the various tools to deliver a software application in a production environment.
- ✚ After that, docker is used for containerization where the docker image is pushed onto the docker hub.
- ✚ Ansible is used for deploying the image in Docker Hub to the production environment.
- ✚ We have also used elasticsearch-logstash-kibana for continuous monitoring

SOURCE CONTROL MANAGEMENT

Tool: GitHub

Source code management (SCM) is used to track modifications to our source code repository. We keep pushing the changes in our code to GitHub whenever we add or modify any features of the Calculator. With SCM we can track the different stages of development of our calculator and revert back to a version if required. We need to perform the following steps for this purpose-

Create a new Repository on GitHub: A new repository is created with a unique name and description on the GitHub account.

The screenshot shows the GitHub 'Create a new repository' interface. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the main heading is 'Create a new repository'. A subtext explains that a repository contains all project files, including revision history, and offers a link to 'Import a repository'. The form includes an 'Owner' dropdown menu currently showing 'richa0897', followed by a 'Repository name' text input field. Below these is a 'Description (optional)' text area. There are two radio button options: 'Public' (which is selected) and 'Private'. Underneath, there's a section 'Initialize this repository with:' which includes a checkbox for 'Add a README file'.

Install Git on local system: Next, we need to install git on our local system in order to start using it to track our source code. To install Git, we use the following commands-

```
$ sudo apt-get update
```

```
$ sudo apt-get install git
```

To verify if installation was successful, we use the following command-

```
$ git --version
```

Initialize project directory with git: To start tracking the files in our project directory, we need to initialize git in it. We can do so by using the following command: -

```
$ git init
```

Set up Remote Repository: The files in the local repository will be pushed into the newly created repository on GitHub. We can set up our remote repository using the command below-

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

Alternatively, we can clone our newly created repository on github using `git clone <git_repo_url>`. We can create our project in this cloned repository and it will automatically be tracked by git.

Add files to the Staging stage: Once we have added a feature of our Calculator, we can push the same to our remote repository i.e., the repository created on GitHub. To track the files, we want to push to the repository we use the following command-

`$ git add <filename> (to track a particular file)`

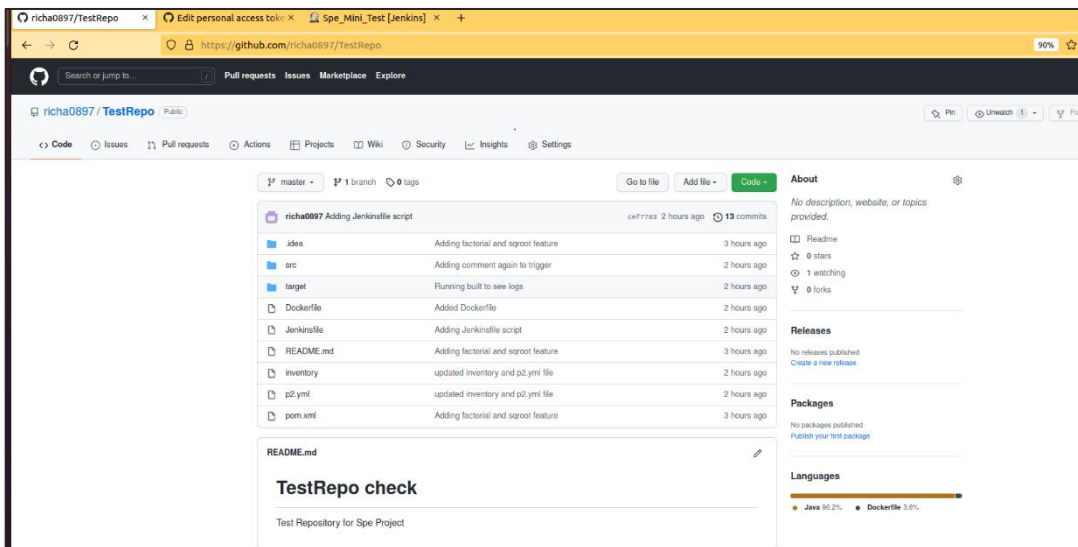
`$ git add . (to track all the files)`

Commit and Push: These commands add our file to the staging area. We can commit the changes using the following command and then push the change to our remote repository-

`$ git commit -m "Commit message name"`

`$ git push origin (when on master branch else merge branch with master and then push)`

The image below is a snap of the GitHub repository created for this project. The link to the GitHub repository is <https://github.com/richa0897/TestRepo>



CONTINUOUS BUILD

Tool: Maven

Maven is a project management tool. Most popular use of Maven is for build management and dependencies. Dependencies are external libraries which we use to integrate it with in your code. While building your Java project, we may need additional JAR files, for example Spring, Hibernate, Logging, JSON etc. Maven will go out and download the JAR files required for our project and make them available during compile/run.

Maven works according to these steps:

1. Read config file (pom.xml)
2. Maven will check the local repository that resides on your machine
3. If not found it will check the central repository(remote) for those dependencies
4. Save the file in the local repo
5. Use those dependencies to build and run your application

Installing Maven: To use maven to build our project locally we need to install maven. We need to perform the following steps for this purpose-

```
$ sudo apt update
```

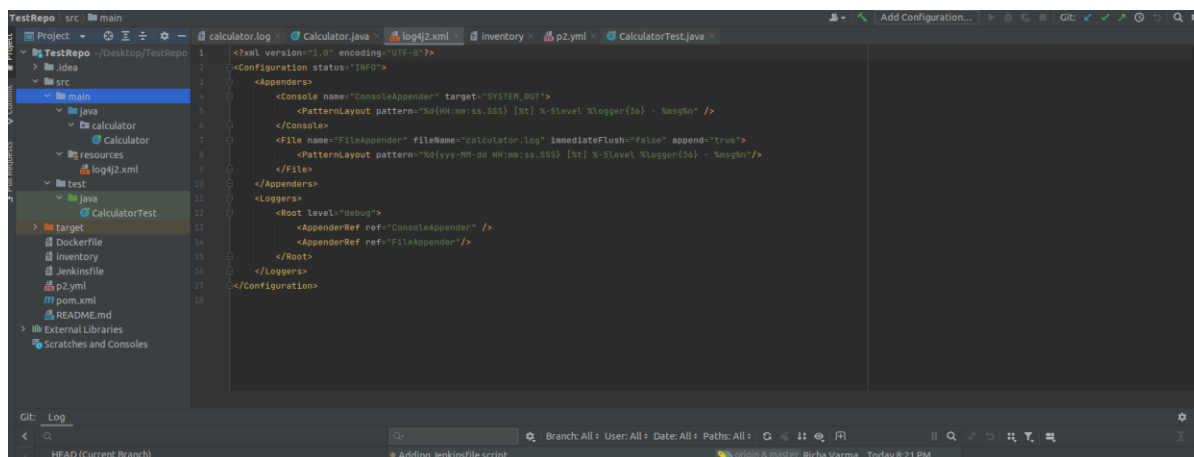
```
$ sudo apt install openjdk-8-jdk
```

```
$ sudo apt update
```

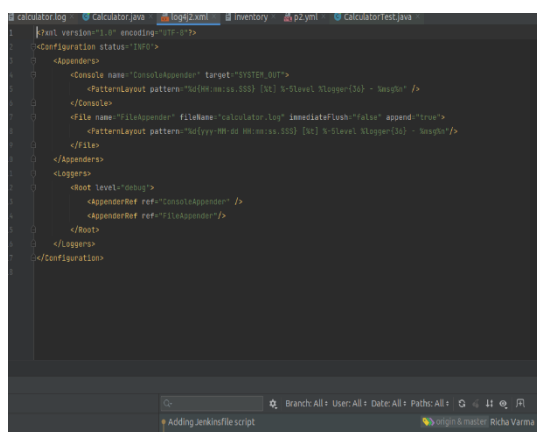
```
$ sudo apt install maven
```

We begin our project by creating a maven project using IntelliJ. The project is created in the directory that is being tracked by git. After creating the maven project, we add the functionalities of the calculator one by one. After adding each functionality, we push the code to our GitHub repository. The necessary thing to do is to add manifest within pom.xml which tells java to where to look for main class in project for an executable jar and since we are using dependencies here, we also made it a jar with dependencies.

The image below shows us the structure of our maven project-



To store the log, log4j Apache jar is used and hence the log4j.xml file was also added. The log4j.xml handles projects log functionalities, be its structure or where it should be stored or append or create a new file based on date and time. We can see the log4j.xml file in the image given below-



These images below show us the functions used to build our scientific calculator.


```

1 usage
public double factorial(double number1) {
    logger.info("[FACTORIAL] - " + number1);
    double result = fact(number1);
    logger.info("[RESULT - FACTORIAL] - " + result);
    return result;
}

5 usages
public double sqroot(double number1) {
    logger.info("[SQ ROOT] - " + number1);
    double result = Math.sqrt(number1);
    logger.info("[RESULT - SQ ROOT] - " + result);
    return result;
}

```

```

public double power(double number1, double number2) {
    logger.info("[POWER] - " + number1 + " * RAISED TO " + number2);
    double result = Math.pow(number1, number2);
    logger.info("[RESULT - POWER] - " + result);
    return result;
}

3 usages
public double naturalLog(double number1) {
    logger.info("[NATURAL LOG] - " + number1);
    double result = 0;
    try {
        if (number1 < 0) {
            result = Double.NaN;
            throw new ArithmeticException("Case of NaN 0.0/0.0");
        }
        else {
            result = Math.log(number1);
        }
    } catch (ArithmeticException error) {
        System.out.println("[EXCEPTION - LOG] - Cannot find log of negative numbers " + error.getMessage());
    }
    logger.info("[RESULT - NATURAL LOG] - " + result);
    return result;
}

```

CONTINUOUS TEST

Tool: JUNIT

JUnit is an open-source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. It is used for Unit Testing of a small chunk of code. Once we are done with our code, we need to execute all tests, and it should pass. Every time any code is added, we need to re-execute all test cases and makes sure nothing is broken.

JUnit provides static methods to test for certain conditions via the Assert class. They allow us to specify the error message, the expected and the actual result. For unit testing we have included junit dependency and written true positive and false positive cases to test while we build our project.

Following are few examples of the test cases that we have used in our project-

Adding Test Classes:

We created a Test class CalculatorTest under the test/java/ directory. Few of the tests can be seen in the image given below: -

```

@Test
public void sqrootTruePositive(){
    assertEquals( message: "Finding square root for True Positive", expected: 2, calculator.sqroot( number1: 4), DELTA);
    assertEquals( message: "Finding square root for True Positive", expected: 1, calculator.sqroot( number1: 1), DELTA);
}

@Test
public void sqrootFalsePositive(){
    assertNotEquals( message: "Finding square root for False Positive", unexpected: 1, calculator.sqroot( number1: 3), DELTA);
    assertEquals( message: "Finding square root for False Positive", unexpected: 0, calculator.sqroot( number1: 4), DELTA);
}
}

```



```

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

    @Test
    public void factorialTruePositive(){
        assertEquals( message: "Finding factorial of a number for True Positive", expected: 120, calculator.fact( num: 5), DELTA);
        assertEquals( message: "Finding factorial of a number for True Positive", expected: 24, calculator.fact( num: 4), DELTA);
    }

    @Test
    public void factorialFalsePositive(){
        assertNotEquals( message: "Finding factorial of a number for False Positive", unexpected: 120, calculator.fact( num: 6), DELTA);
        assertNotEquals( message: "Finding factorial of a number for False Positive", unexpected: 24, calculator.fact( num: 3), DELTA);
    }
}

```

Running Tests: In order to run the tests, we can use the following command-

\$mvn test -Dtest=TestClassName (Runs a single test on the console for the specified test class)

\$mvn test -Dtest=TestClassName1,TestClassName2 (Runs multiple tests on the console for the specified test classes)

\$ mvn clean test (Runs all the test cases)

After successfully building our project, we can run the project using the artifact that is generated in the target folder. On running the project, a log file calculator.log is generated that holds all the logs for our project. Following is the image of the log file-

```

2022-04-16 19:26:43.691 [main] INFO    calculator.Calculator - [FACTORIAL] - 5.0
2022-04-16 19:26:43.702 [main] INFO    calculator.Calculator - [RESULT - FACTORIAL] - 120.0
|
2022-04-16 19:57:53.541 [main] INFO    calculator.Calculator - [POWER - 4.0 RAISED TO] 2.0
2022-04-16 19:57:53.559 [main] INFO    calculator.Calculator - [RESULT - POWER] - 16.0
2022-04-16 19:58:04.318 [main] INFO    calculator.Calculator - [NATURAL LOG] - 6.0
2022-04-16 19:58:04.319 [main] INFO    calculator.Calculator - [RESULT - NATURAL LOG] - 1.791759469228055

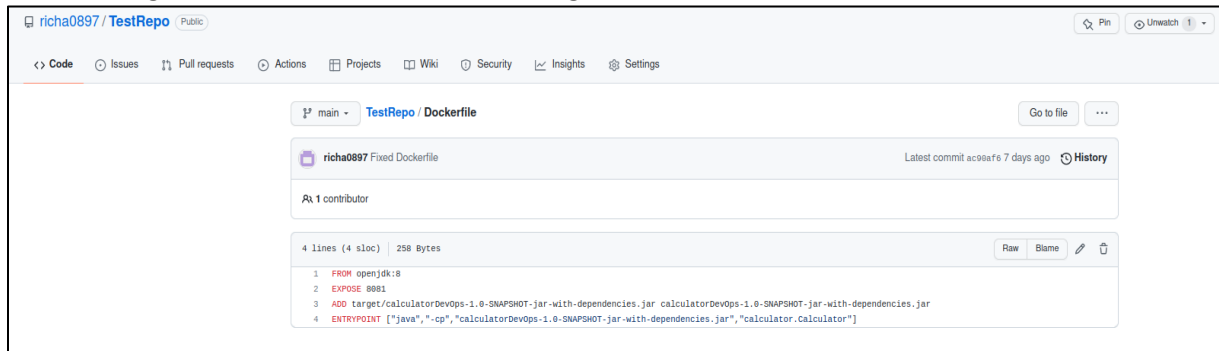
```

Containerization

Tool: Docker

Docker is platform which provides OS level virtualization to deliver software as packages. So the plan is to create a docker image of jar file created after build process and push the latest image on to docker hub. The pushed procedure would be done after every build which would include removing previously pushed docker image and replace with latest built one. The pushed images can be found at https://hub.docker.com/repository/docker/richavarma/my_docker_repo

We are using a Dockerfile to build a docker image. The contents of the Dockerfile can be seen below-



In order to run the jar file in the production environment open-jdk-8 image would be required. Therefore we are pulling this image from dockerhub in the first line of our dockerfile. We are exposing our port 8081. Then we are adding the jar file generated in the target directory to the specified destination in the image. We also need to specify the main class of our build as entrypoint.

When the image is deployed to the production environment we can create a container from it and run the jar file. For this docker needs to be installed on the client system. After installation we need to start the docker service via the command-

```
$sudo systemctl start docker
```

To create a container from the docker image we use the following command-

```
$docker run -i -t --create <container_name> <base_image_name> /bin/bash
```

Continuous Deployment

Tool: Ansible

Ansible is an open source automation platform. It is an automation engine that runs ansible playbooks. Playbooks are defined tasks, where we define environments and workflows. There are two types of machines in the Ansible architecture: control nodes and managed hosts. Ansible is installed and run from a control node, and this machine also has copies of your Ansible project files.

Managed hosts are listed in an inventory, which also organizes those systems into groups for easier collective management. The inventory can be defined in a static text file, or dynamically determined by scripts that get information from external sources.

Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. YAML stands for Yet Another Markup Language. Playbooks are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks.

Working with ansible:

For Ansible to work make sure you have python and ssh server installed on your controller and managed nodes. Follow the commands below:

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

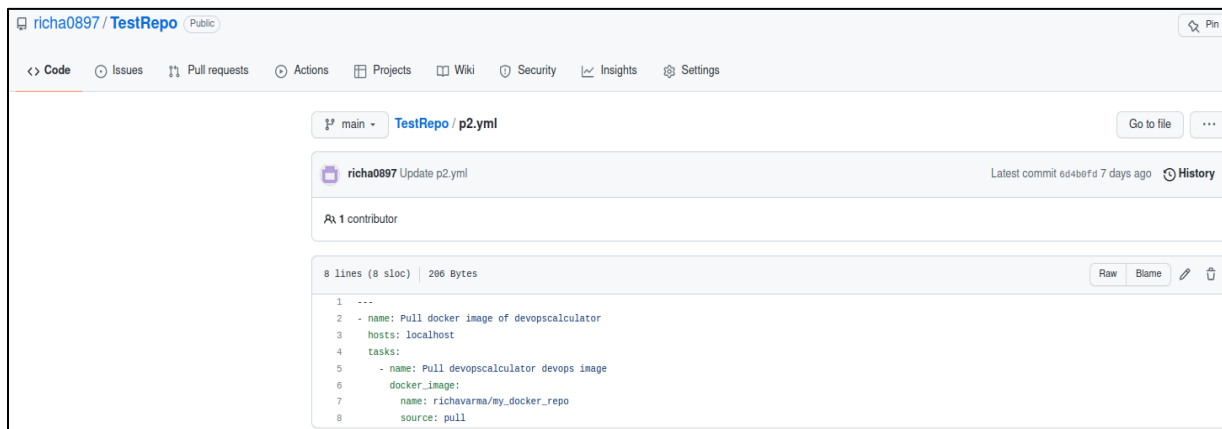
```
$ ssh-keygen -t rsa
```

Ansible is installed only on the controller node using the following commands:

```
$ sudo apt update
```

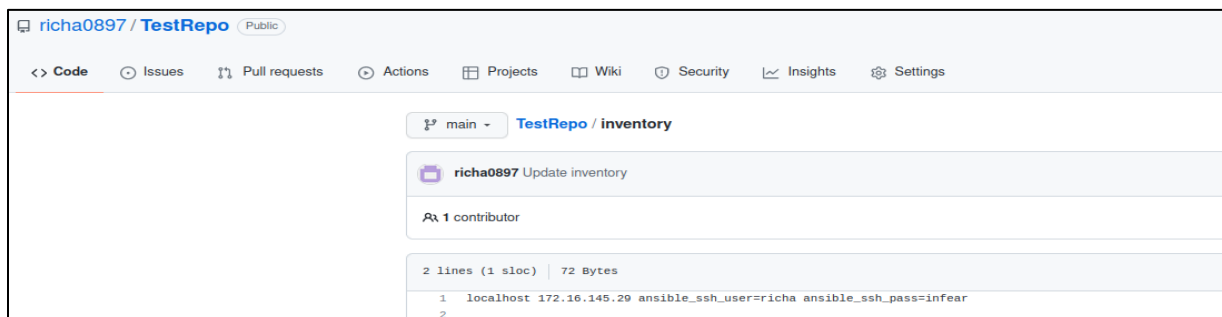
```
$ sudo apt install ansible
```

The contents of our inventory and playbook file can be seen in the images below-



The screenshot shows a GitHub repository named 'TestRepo' by user 'richa0897'. The file 'p2.yml' is selected, showing 8 lines of YAML code. The code defines a task to pull a Docker image of 'devopscalculator' from a repository named 'richavarma/my_docker_repo'.

```
1 ---
2 - name: Pull docker image of devopscalculator
3   hosts: localhost
4   tasks:
5     - name: Pull devopscalculator devops image
6       docker_image:
7         name: richavarma/my_docker_repo
8       source: pull
```



The screenshot shows the same GitHub repository 'TestRepo' by 'richa0897', but now the file 'inventory' is selected. It shows 2 lines of text defining the localhost host with its IP and SSH credentials.

```
1 localhost 172.16.145.29 ansible_ssh_user=richa ansible_ssh_pass=infear
2
```

CONTINUOUS INTEGRATION

Tool: Jenkins

Jenkins is used as an orchestrator to integrate different stages of DevOps Pipeline. The ultimate goal of DevOps is to automate all the stages of a software development life cycle and this can easily be done with Jenkins. Using a Jenkins pipeline project, we integrate SCM, Build, Test, Release and Deploy stage. For integrating these stages, we need to install a set of plugins and make certain configurations.

We need to perform the following steps to use Jenkins for our project-

1. Start the Jenkins server
2. Integrate Jenkins with Git and GitHub. Use webhooks to automatically trigger build on every push to the remote repository
3. Integrate with Maven to automatically build and test
4. Integrate with Docker to push the image to Docker Hub
5. Integrate with Ansible to pull the image from Docker Hub and deploy to production environment.
6. Create a pipeline project and connect all the stages

Starting Jenkins:

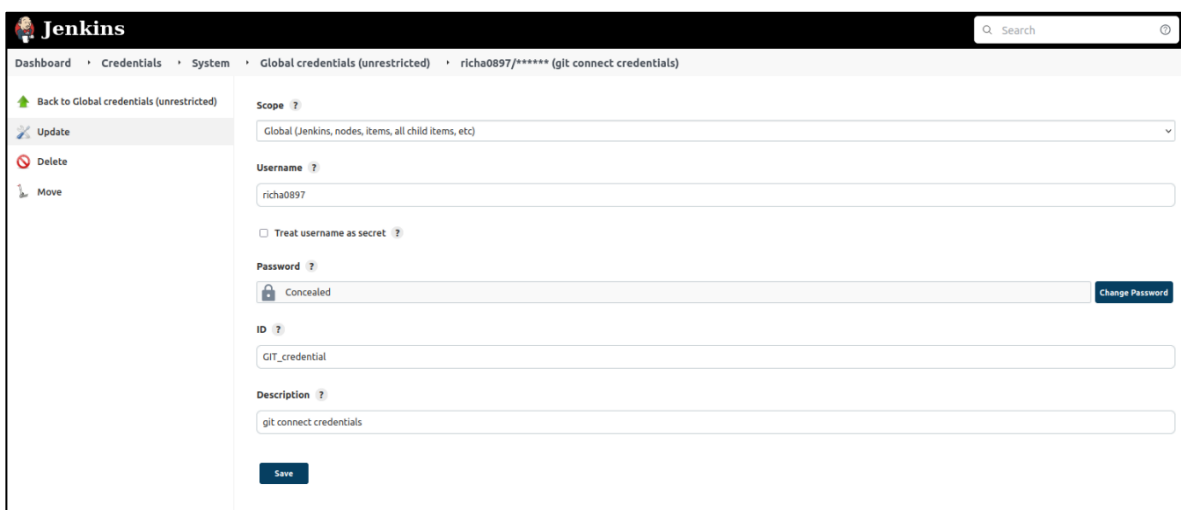
After installation we can start Jenkins using the command `sudo systemctl start Jenkins`. Jenkins starts at port number 8080. We can login to Jenkins using `http://localhost:8080` on our browser.

Integrating Jenkins with GitHub:

We would want a build to be triggered automatically on every push of a new feature or of any modifications to our source code to our github repository. To facilitate this the following steps need to be performed-

We need to install the Git and the GitHub plugins. To do so we go to Manage Jenkins on the Jenkins Dashboard → Manage plugins → Install Git Plugin and Install GitHub Plugin

We also need to set credentials for our GitHub Repository on Jenkins otherwise we won't be able to connect to it. For that we need to go to Manage Jenkins → Manage Credentials → Global Credentials → Add Credentials → Under Kind we select Username and Password and under Scope we select Global option → Enter your Github username and password → Save



The screenshot shows the Jenkins web interface for adding a new credential. The breadcrumb trail at the top reads: Dashboard > Credentials > System > Global credentials (unrestricted) > richa0897/***** (git connect credentials). On the left sidebar, there are links for 'Back to Global credentials (unrestricted)', 'Update', 'Delete', and 'Move'. The main form area is titled 'Add Credentials' and contains the following fields: 'Scope' (a dropdown menu set to 'Global (Jenkins, nodes, items, all child items, etc)'), 'Username' (a text input field containing 'richa0897'), a checkbox for 'Treat username as secret' (which is unchecked), 'Password' (a text input field with a lock icon and the text 'Concealed', and a 'Change Password' button), 'ID' (a text input field containing 'GIT_credential'), and 'Description' (a text input field containing 'git connect credentials'). A 'Save' button is located at the bottom left of the form.

We also need to add path to the git executable file. To find out this path we can use the command 'which git'. We go to Manage Jenkins → Global Tool Configuration → Under Git we add the path

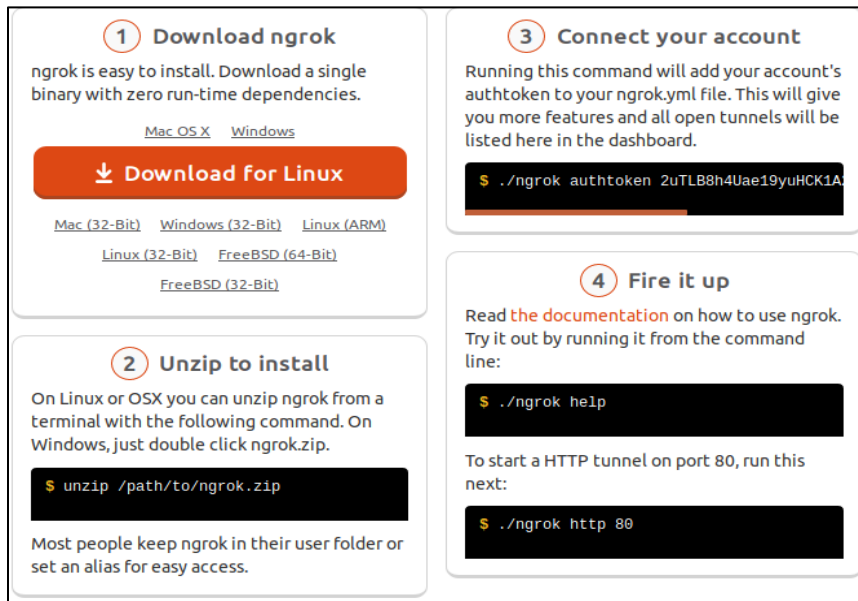


Git	
Git installations	
Name	Path to Git executable ?
Default	git
<input type="checkbox"/> Install automatically ?	
<button>Delete Git</button>	

To trigger build on every push to the repository we need to use GitHub hook trigger for GITScm Polling. A webhook is an outbound HTTP request that helps you create a relationship between your GitHub repository and a particular URL, in this case a pipeline. After you configure the webhook, changes you make in your GitHub repository will trigger the pipeline and its associated tasks.

It's common to use github webhook when jenkins is hosted on any cloud. But if Jenkins is running on localhost, then the webhook doesn't work. So, for such cases ngrok comes into play. Ngrok is a great tool for creating tunnel to a person's IP. If we want to securely expose our local web server to the internet and capture all traffic for detailed inspection and replay then we use this.

To start using ngrok we perform the following steps. Here we use port 8080 instead of port 80.



1 Download ngrok
ngrok is easy to install. Download a single binary with zero run-time dependencies.
[Mac OS X](#) [Windows](#)
Download for Linux
[Mac \(32-Bit\)](#) [Windows \(32-Bit\)](#) [Linux \(ARM\)](#)
[Linux \(32-Bit\)](#) [FreeBSD \(64-Bit\)](#)
[FreeBSD \(32-Bit\)](#)

2 Unzip to install
On Linux or OSX you can unzip ngrok from a terminal with the following command. On Windows, just double click ngrok.zip.

```
$ unzip /path/to/ngrok.zip
```


Most people keep ngrok in their user folder or set an alias for easy access.

3 Connect your account
Running this command will add your account's authtoken to your ngrok.yml file. This will give you more features and all open tunnels will be listed here in the dashboard.

```
$ ./ngrok authtoken 2uTLB8h4Uae19yuHCK1A
```

4 Fire it up
Read [the documentation](#) on how to use ngrok. Try it out by running it from the command line:

```
$ ./ngrok help
```


To start a HTTP tunnel on port 80, run this next:

```
$ ./ngrok http 80
```



```
ngrok by @inconshreveable

Session Status               online
Account                      varma.richavarma@gmail.com (Plan: Free)
Version                      2.3.40
Region                       United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://0e45-103-156-19-229.ngrok.io -> http://localhost:8080
                              https://0e45-103-156-19-229.ngrok.io -> http://localhost:8080

Connections                  ttl    opn    rt1    rt5    p50    p90
                              4      0      0.00   0.01   6.31   9.66

HTTP Requests
-----
GET /favicon.ico              200 OK
GET /static/de61580b/css/simple-page-variables.css 200 OK
GET /static/de61580b/images/svgs/Logo.svg         200 OK
GET /static/de61580b/css/simple-page.theme.css     200 OK
GET /static/de61580b/css/simple-page-forms.css     200 OK
```

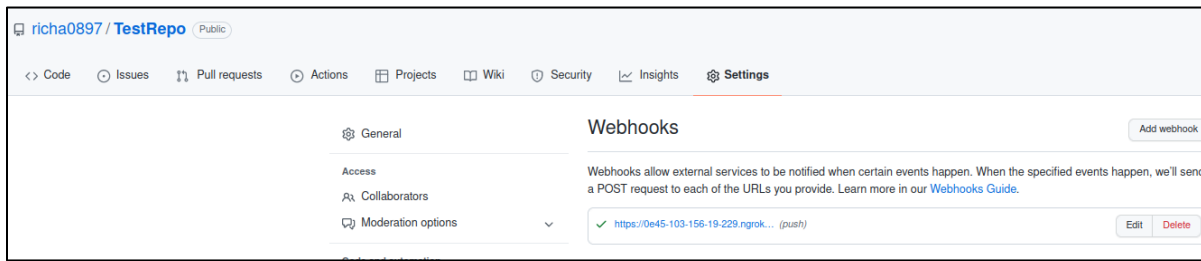
In the unpaid version of ngrok the IP that is generated is temporary and so one will need to change the IP details at the required locations from time to time.

To enable webhook we perform the following steps-

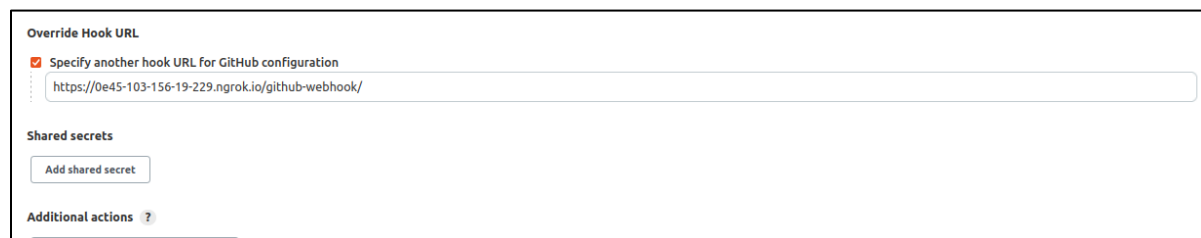
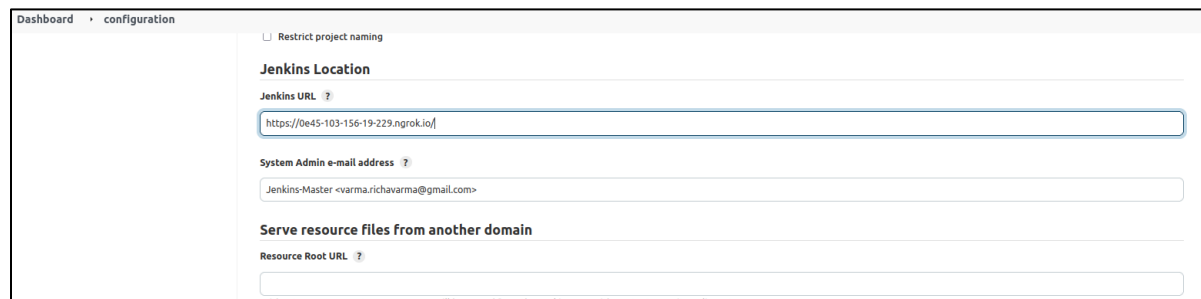
1. Go to the GitHub repository page in the web browser.
2. Click the Settings tab.
3. In the navigation pane, click WebHooks.
4. Click Add Webhook.
5. In the Payload URL field, paste the URL generated by ngrok followed by /github-webhook/
6. In the Content type field, select application/json.
7. Below Which events would you like to trigger this webhook select Push to trigger build on push to the repository.
8. Ensure that the Active check box is selected. This option keeps the webhook enabled and sends notifications whenever an event is triggered.
9. Click Add webhook to complete the configuration of the webhook in GitHub Enterprise.

The screenshot shows the GitHub repository settings page for 'richa0897 / TestRepo'. The 'Settings' tab is selected in the top navigation bar. On the left sidebar, the 'Webhooks' option is highlighted under the 'Code and automation' section. The main content area is titled 'Webhooks / Manage webhook' and contains a form for configuring a new webhook. The form includes the following fields and options:

- Payload URL:** A text input field containing the URL 'https://0e45-103-156-19-229.ngrok.io/github-webhook/'.
- Content type:** A dropdown menu with 'application/json' selected.
- Secret:** A text input field for a secret key.
- SSL verification:** A section with a note 'By default, we verify SSL certificates when delivering payloads.' and two radio buttons: 'Enable SSL verification' (selected) and 'Disable (not recommended)'.
- Which events would you like to trigger this webhook?:** A section with three radio buttons: 'Just the push event.' (selected), 'Send me everything.', and 'Let me select individual events.'.
- Active:** A checkbox labeled 'Active' which is checked, with a sub-note 'We will deliver event details when this hook is triggered.'

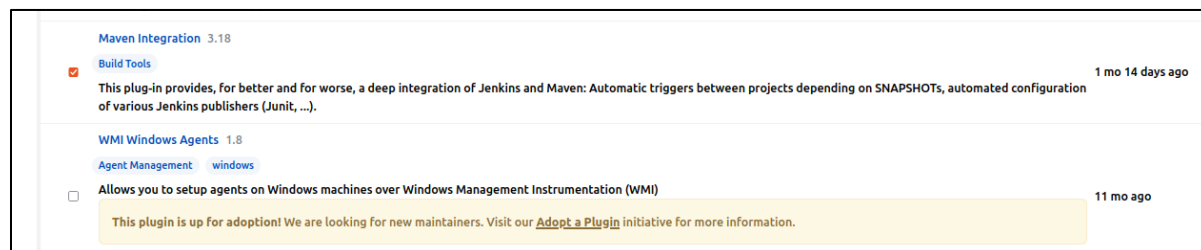


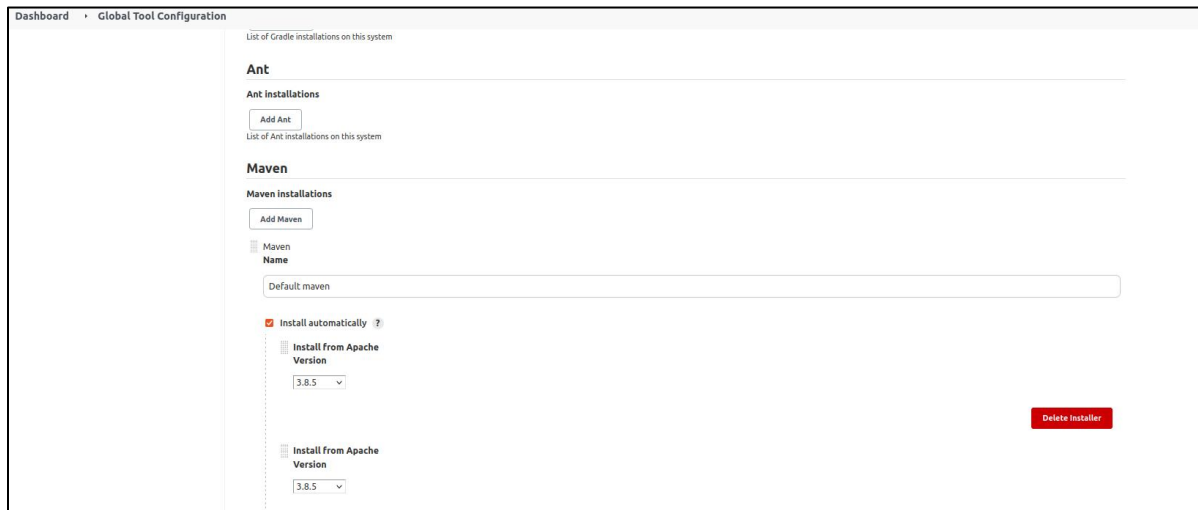
We need to make changes to the Jenkins URL to receive the payload. We go to Manage Jenkins→Configure Systems→Under Jenkins URL we specify the IP generated using ngrok. Under Github Server→We select mange hooks and under advanced option we override hook trigger with the payload URL which is exactly same as what was specified on github.



Integrating Jenkins with Maven:

To facilitate automatic build and test of our maven project in jenkins we need to first install Maven Integration Plugin. To tell Jenkins where Maven is installed we need to go to Manage Jenkins→Global Tool Configuration→Under Maven we need to specify Maven name and version.





Integrating Jenkins with Docker:

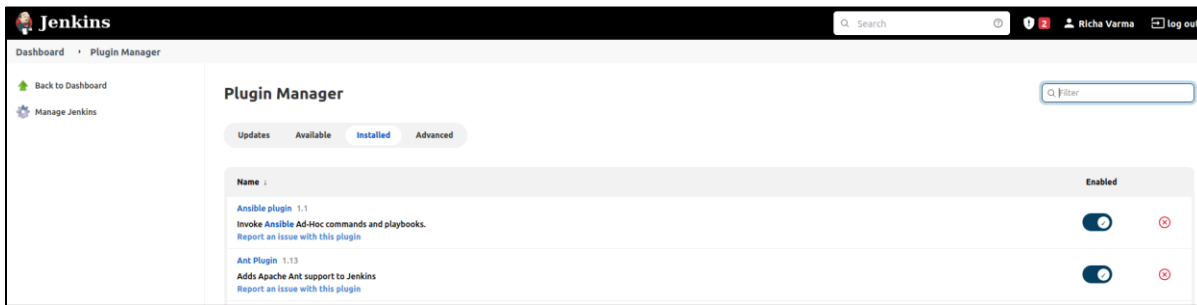
We need to containerize our project so that it can be deployed to the production environment easily. To run our project on the production environment open-jdk-8 would be required. So we build an image containing the open-jdk-8 package and the built artifact and release this image on the docker hub. To integrate docker with Jenkins we need to add docker hub credentials to Jenkins. For this we go to Manage Jenkins→Manage Credentials→Global→Add Credentials→Fill in the required details as given in the image below-



To build the image we would need one Dockerfile which will be at the same level as our source folder.

Integrating Jenkins with Ansible:

We also need to integrate Jenkins with Ansible in order to deploy the image pushed to docker hub to the production environment. To integrate Jenkins with Ansible we need to install the Ansible Plugin and the Ant Plugin. We also need to set path to the location where ansible is installed on our server.



For using Ansible to deploy our image we would need two files inventory and playbook file with yml extension. Inventory file contains the address of the hosts on which we want to deploy and the playbook file is used to get the image from docker hub and deploy it to the hosts mentioned on the inventory file.

Creating a pipeline project to connect all the stages:

Now we need to create a pipeline project and connect the different stages of DevOps pipeline. To do so we go to the Jenkins Dashboard → New Item → Give a name to the item and select pipeline project → Add item.

We select GitHub Hook trigger for GitScm Polling under build triggers.

Under Pipeline Script we choose Pipeline Script from SCM and do the following settings-

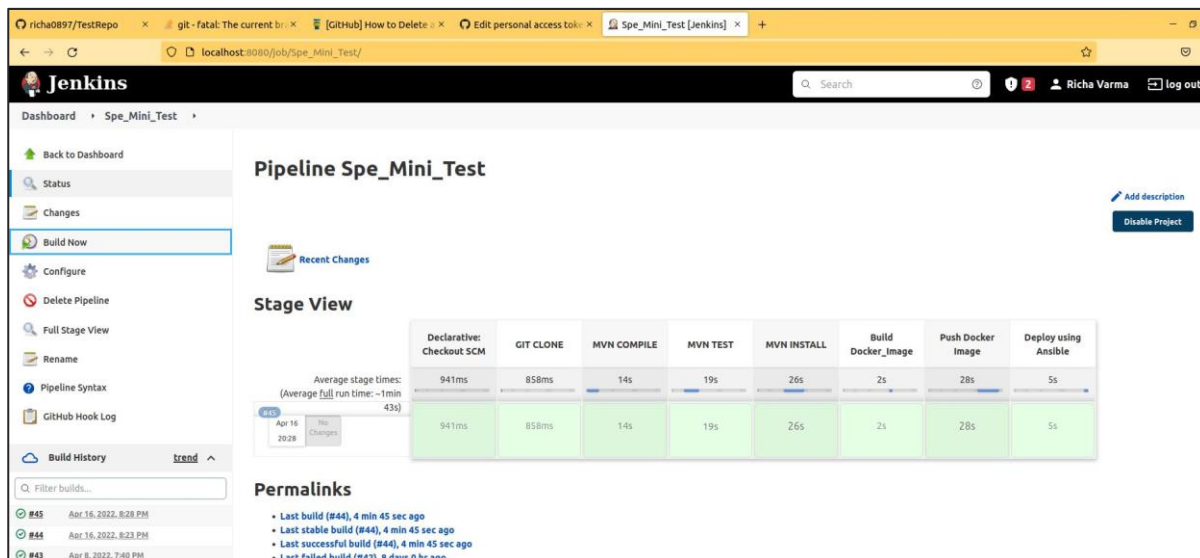
We are creating Jenkinsfile that has our pipeline syntax. The Jenkinsfile has to be pushed into our repository and the path to the file needs to be specified in the script path. For our project the Pipeline script can be seen below-

```
1 pipeline {
2   agent any
3   environment {
4     imageName = ""
5   }
6   stages {
7     stage('GIT CLONE') {
8       steps {
9         git url: 'https://github.com/richa0897/TestRepo.git', branch: 'main'
10      }
11    }
12    stage('MVN COMPILE') {
13      steps {
14        echo "Compiling the source Java classes of the project"
15        sh "mvn compile"
16      }
17    }
18    stage('MVN TEST') {
19      steps {
20        echo "Running the test cases of the project"
21        sh "mvn test"
22      }
23    }
24    stage('MVN INSTALL') {
25      steps {
26        echo "building the project and installs the project files(JAR) to the local repository"
27        sh "mvn install"
28      }
29    }
30    stage('Build Docker Image'){
31      steps {
32        script {
33          imageName = docker.build "richavarma/my_docker_repo:latest"
34        }
35      }
36    }
37    stage('Push Docker Image')
38    {
39      steps {
40        script{
41          docker.withRegistry('', 'credentials-Id') {
42            imageName.push()
43          }
44        }
45      }
46    }
47    stage('Deploy using Ansible'){
48      steps{
49        ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inventory', playbook: 'p2.yml'
50      }
51    }
52  }
53 }
54 }
55 }
```

Under pipeline syntax we need to give path to our inventory file and playbook file in repository for deployment using ansible.

We apply and save all the changes. This will get our pipeline up and ready. On any feature push the build gets triggered automatically and all the pipeline stages are executed one after the other. The image of our build is pushed to our dockerhub repository. Ansible pulls the latest image from the dockerhub and deploys it to the production environment.

The different build stages of our pipeline project can be seen in the image below-



After deployment to the production environment (here localhost) we can see the image pulled from docker hub. We can see it in the image given below-

```
richa@richa-VirtualBox:~/Desktop/TestRepo$ sudo docker images
[sudo] password for richa:
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
richavarma/my_docker_repo  latest      d86acd732119     3 minutes ago   528MB
richavarma/my_docker_repo  <none>      f2c54ef233f3     8 minutes ago   528MB
richavarma/my_docker_repo  <none>      d0332f6bbc07     8 days ago      528MB
<none>                <none>      0f57ecda2cfd     8 days ago      528MB
richavarma/my_docker_repo  <none>      fada4b3f116c     8 days ago      528MB
<none>                <none>      c309f1608b27     8 days ago      528MB
richavarma/my_docker_repo  <none>      51d06778ba9a     8 days ago      528MB
<none>                <none>      f184da8e3a3c     8 days ago      528MB
```

We create a container from the image . Then we can run the scientific calculator program easily-

```
richa@richa-VirtualBox:~/Desktop/TestRepo$ sudo docker run -it --name newcontspe richavarma/my_docker_repo /bin/bash
Calculator-DevOps, Choose to perform operation
Press 1 to find factorial
Press 2 to find Square root
Press 3 to find power
Press 4 to find natural logarithm
Press 5 to exit
Enter your choice: 1
```

Continuous Monitoring:

Tool: ELK - Elastic Search, Logstash, Kibana

ELK stack makes the monitoring tool for any deployed software, it analyzes the logs and the same analysis can then be viewed on kibana dashboard.

To start with download elastic search, logstash and kibana from <https://www.elastic.co>

Run them side by side and feed the logstash your log set after configuring its config file. In the config file we provide the details for logs. Here the config file logstash.conf.

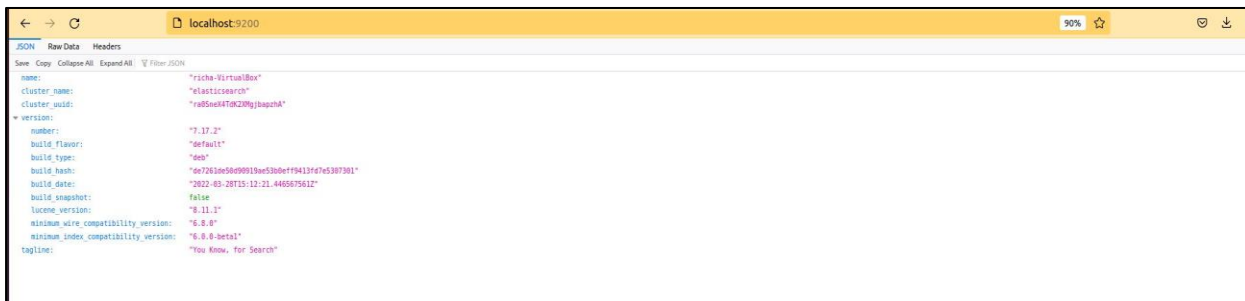
Elastic search starts at localhost:9200, and kibana starts at localhost:5601

Following images show the installation commands and commands to start the elk services-

First we install elasticsearch from command line-

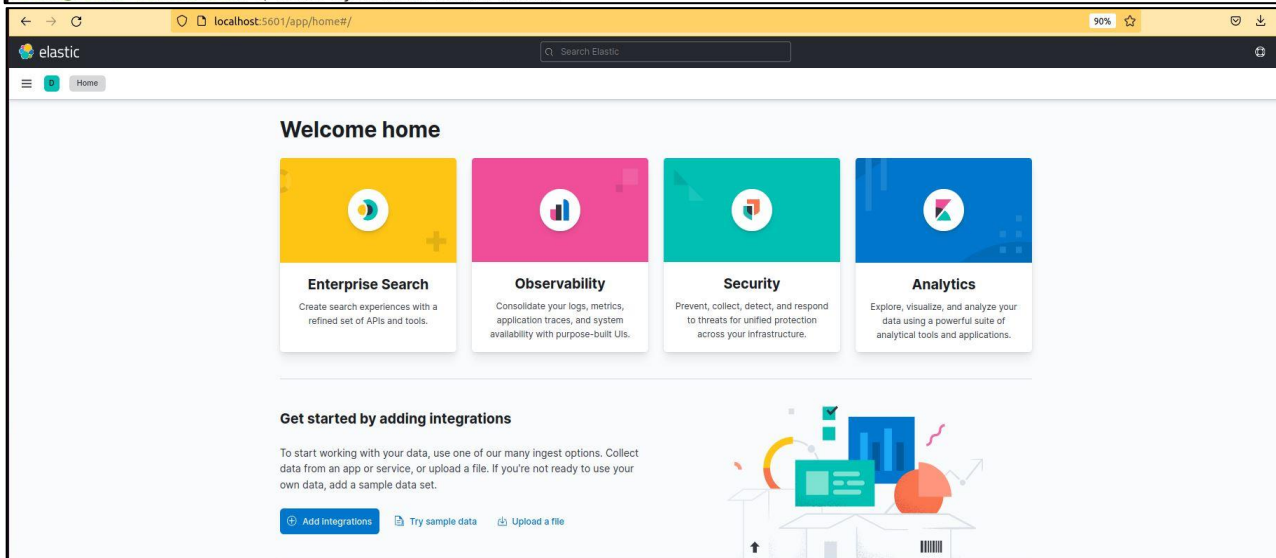
```
richa@richa-VirtualBox:~$ sudo apt install elasticsearch
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  daemon libfwupdplugin1 linux-headers-5.13.0-30-generic linux-hwe-5.13-headers-5.13.0-30 linux-image-5.13.0-30-generic
  linux-modules-5.13.0-30-generic linux-modules-extra-5.13.0-30-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  elasticsearch
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 312 MB of archives.
After this operation, 517 MB of additional disk space will be used.
Get:1 https://artifacts.elastic.co/packages/7.x/apt/stable/main amd64 elasticsearch amd64 7.17.2 [312 MB]
Fetched 312 MB in 2min 54s (1,791 kB/s)
Selecting previously unselected package elasticsearch.
(Reading database ... 233000 files and directories currently installed.)
Preparing to unpack .../elasticsearch_7.17.2_amd64.deb ...
Creating elasticsearch group... OK
Creating elasticsearch user... OK
Unpacking elasticsearch (7.17.2) ...
Setting up elasticsearch (7.17.2) ...
### NOT starting on installation, please execute the following statements to configure elasticsearch service to start automatically using systemd
  sudo systemctl daemon-reload
  sudo systemctl enable elasticsearch.service
### You can start elasticsearch service by executing
  sudo systemctl start elasticsearch.service
Created elasticsearch keystore in /etc/elasticsearch/elasticsearch.keystore
Processing triggers for systemd (245.4-4ubuntu3.15) ...
richa@richa-VirtualBox:~$ sudo systemctl enable elasticsearch
Synchronizing state of elasticsearch.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable elasticsearch
Created symlink /etc/systemd/system/multi-user.target.wants/elasticsearch.service → /lib/systemd/system/elasticsearch.service.
richa@richa-VirtualBox:~$ sudo systemctl start elasticsearch
```

The elasticsearch starts on port 9200



We then install and start kibana service. Kibana service can be opened on port 5601.

```
richa@richa-VirtualBox:~$ sudo apt install kibana
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  daemon libfwupdplugin1 linux-headers-5.13.0-30-generic linux-hwe-5.13-headers-5.13.0-30 linux-image-5.13.0-30-generic
  linux-modules-5.13.0-30-generic linux-modules-extra-5.13.0-30-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  kibana
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 286 MB of archives.
After this operation, 771 MB of additional disk space will be used.
Get:1 https://artifacts.elastic.co/packages/7.x/apt stable/main amd64 kibana amd64 7.17.2 [286 MB]
Fetched 286 MB in 1min 23s (3,466 kB/s)
Selecting previously unselected package kibana.
(Reading database ... 234125 files and directories currently installed.)
Preparing to unpack .../kibana_7.17.2_amd64.deb ...
Unpacking kibana (7.17.2) ...
Setting up kibana (7.17.2) ...
Creating kibana group... OK
Creating kibana user... OK
Created Kibana keystore in /etc/kibana/kibana.keystore
Processing triggers for systemd (245.4-4ubuntu3.15) ...
richa@richa-VirtualBox:~$ sudo systemctl enable kibana
Synchronizing state of kibana.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable kibana
Created symlink /etc/systemd/system/multi-user.target.wants/kibana.service → /etc/systemd/system/kibana.service.
richa@richa-VirtualBox:~$ sudo systemctl start kibana
```



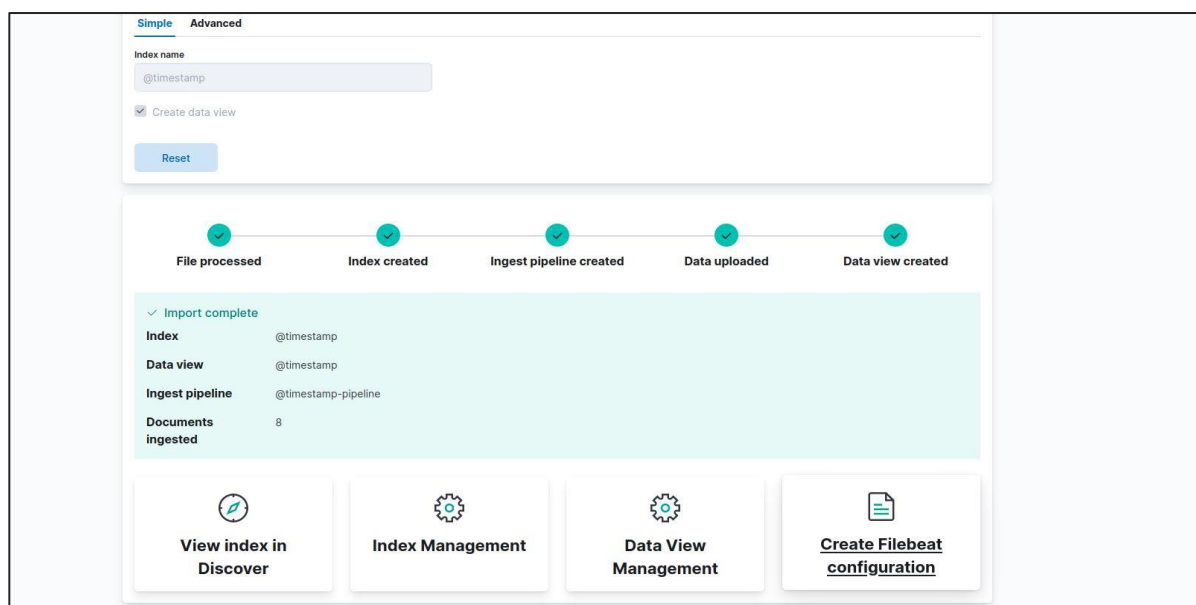
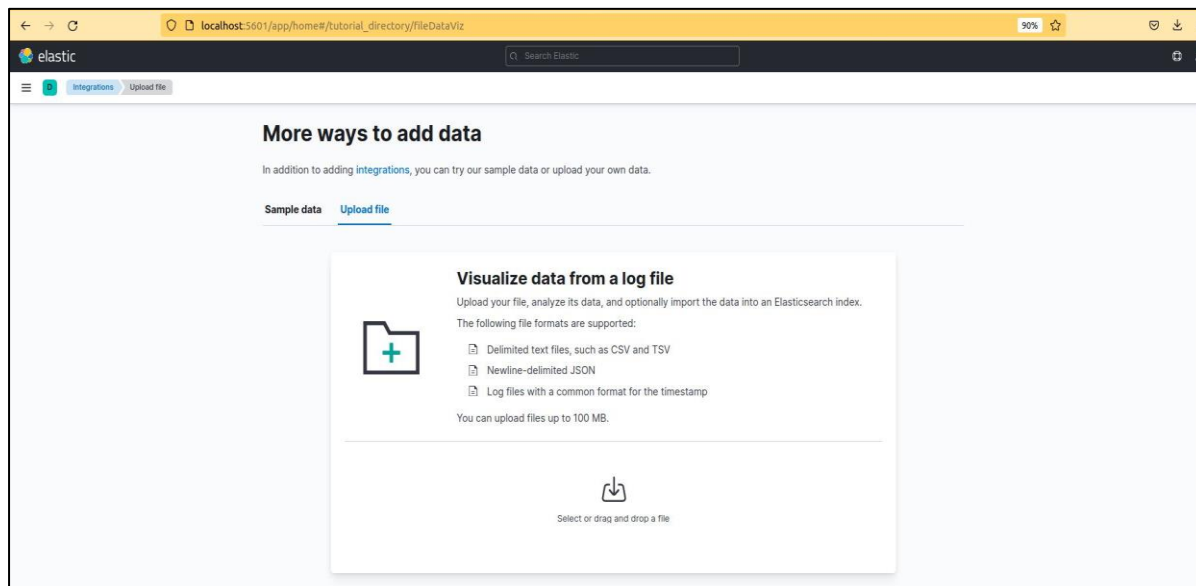
At last, we install logstash and enable the service.

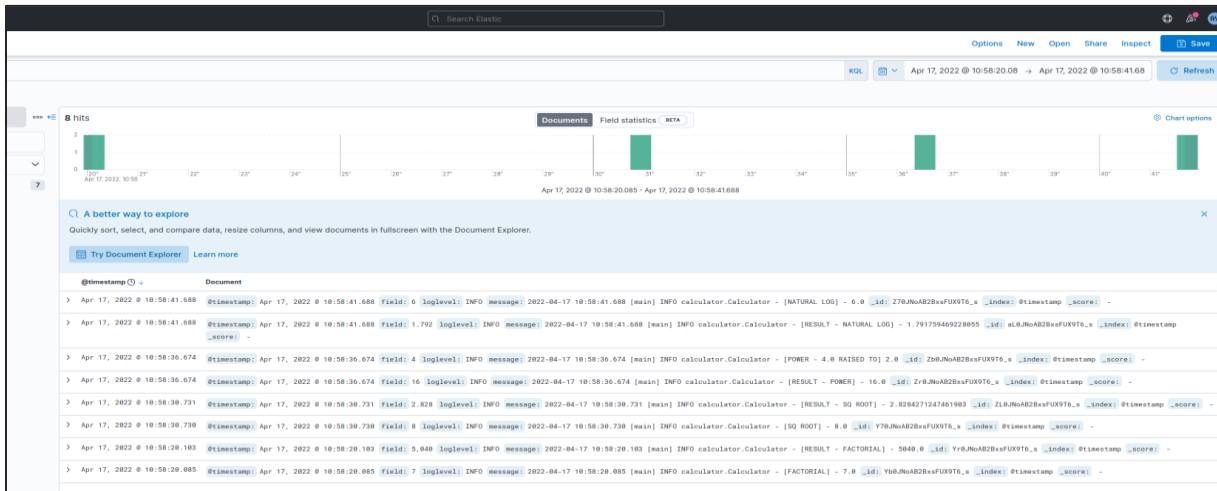
```
richa@richa-VirtualBox:~$ sudo apt install logstash
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  daemon libfwupdplugin1 linux-headers-5.13.0-30-generic linux-hwe-5.13-headers-5.13.0-30 linux-image-5.13.0-30-generic
  linux-modules-5.13.0-30-generic linux-modules-extra-5.13.0-30-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  logstash
0 upgraded, 1 newly installed, 0 to remove and 35 not upgraded.
Need to get 365 MB of archives.
After this operation, 626 MB of additional disk space will be used.
Get:1 https://artifacts.elastic.co/packages/7.x/apt stable/main amd64 logstash amd64 1:7.17.2-1 [365 MB]
Fetched 365 MB in 1min 59s (3,064 kB/s)
Selecting previously unselected package logstash.
(Reading database ... 294533 files and directories currently installed.)
Preparing to unpack .../logstash_1%3a7.17.2-1_amd64.deb ...
```



```
richa@richa-VirtualBox:~$ sudo systemctl start logstash
richa@richa-VirtualBox:~$ sudo systemctl enable logstash
richa@richa-VirtualBox:~$ sudo service logstash status
● logstash.service - logstash
   Loaded: loaded (/etc/systemd/system/logstash.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2022-04-08 23:22:17 IST; 36s ago
     Main PID: 53894 (java)
       Tasks: 14 (limit: 7040)
      Memory: 299.1M
    CGroup: /system.slice/logstash.service
            └─53894 /usr/share/logstash/jdk/bin/java -Xms1g -Xmx1g -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiating
Apr 08 23:22:17 richa-VirtualBox systemd[1]: logstash.service: Scheduled restart job, restart counter is at 3.
Apr 08 23:22:17 richa-VirtualBox systemd[1]: Stopped logstash.
Apr 08 23:22:17 richa-VirtualBox systemd[1]: Started logstash.
Apr 08 23:22:17 richa-VirtualBox logstash[53894]: Using bundled JDK: /usr/share/logstash/jdk
Apr 08 23:22:17 richa-VirtualBox logstash[53894]: OpenJDK 64-Bit Server VM warning: Option UseConcMarkSweepGC was deprecated in version 9.0 and will li
```

We then upload the generated log file by going to Integration on Kibana dashboard. We can view your logs and visualize them based on the chosen index pattern. We click on the discover or visualize to see through the logs and monitor them.





Challenges faced during project-

1. For building project automatically on push to GitHub repository we needed a public IP. For that ngrok was used. The unpaid version of ngrok gives an IP that expires after few hours. So, we need to new IP and make a new webhook for that IP in case the previous one expires
2. During deployment through ansible, python3 module was disutils. spawn modules were needed to be installed. Ansible needs the host machine to have these modules installed for successful deployment.
3. To make sure the container does not exit after its creation we use -t flag while executing the docker run command to attach it to the terminal. Also, we need to use -i flag to make the session interactive.
4. There was a connect issue when we were pushing the image to the docker-hub. However, it was resolved after adding the credentials id for docker-hub under global credentials.
5. The elk services can cause problems in low RAM environment. So, it can also be run on browser. One only needs to create his/her user account