

CS 816 - Software Production Engineering

Mini Project - Scientific Calculator with DevOps

NAME - SAMIR AHMED GHOURI(MT2022100)

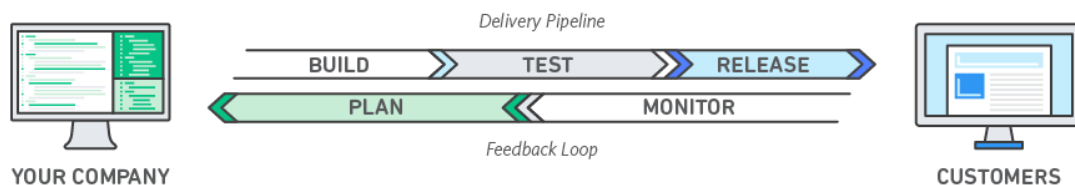
Problem Statement

Create a scientific calculator program with user menu driven operations

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

What is DevOps?

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



Why is DevOps Used?

- **Speed**

Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example,

microservices and continuous delivery let teams take ownership of services and then release updates to them quicker.

- **Rapid Delivery**

Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers' needs and build competitive advantage. Continuous integration and continuous delivery are practices that automate the software release process, from build to deploy.

- **Reliability**

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like continuous integration and continuous delivery to test that each change is functional and safe. Monitoring and logging practices help you stay informed of performance in real-time.

- **Scale**

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.

- **Improved Collaboration**

Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams collaborate closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).

- **Security**

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale.

Tools Used



• : It's a Java-based application development tool that lets us add dependencies and build a jar file (a snapshot of our project) that can be run on any machine.



• : Helps code hosting platform for version control and collaboration.



• : To convert the private IP address of the local machine to a public IP address to perform a webhook.



• **Jenkins** : It is used for DevOps(for Continuous Integration and Continuous Deployment)



• **docker** : It is used for developing, shipping, and running applications.



• : It is used for configuration management, application deployment, intraservice orchestration, and provisioning.

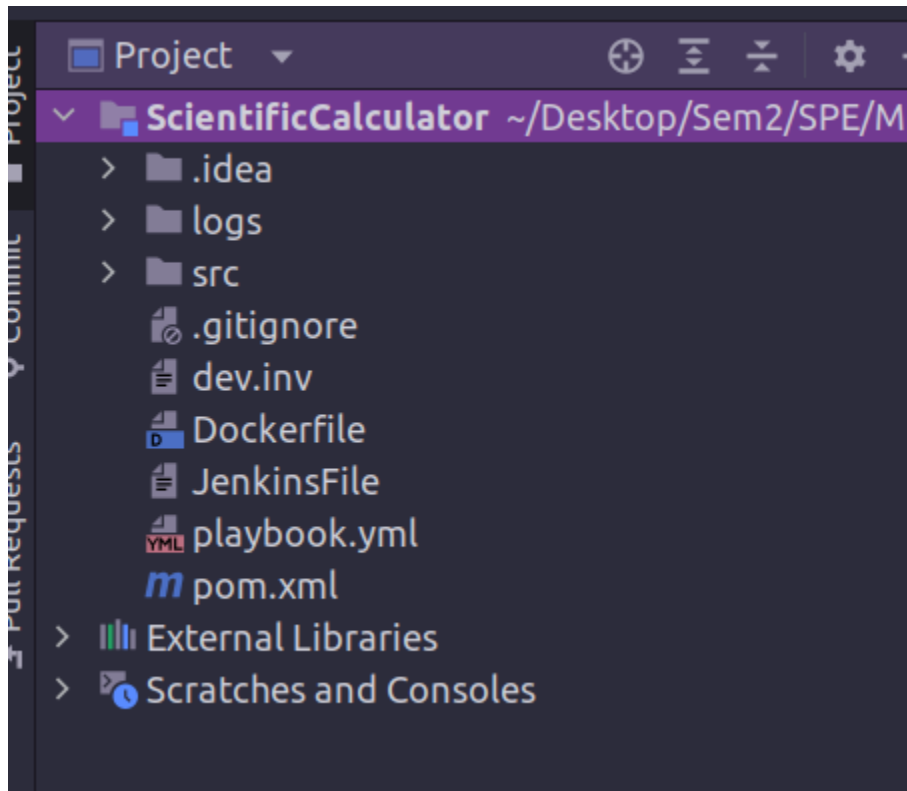
Important Links

GitHub Repository Link -> <https://github.com/samir11096/ScientificCalculator>

DockerHub Image Repository Link -> <https://hub.docker.com/r/samiraghour/calculator>

Steps

We start by creating a maven project for our Scientific Calculator Project with the following directory structure.



The src folder has two parts: **main** and **test**.

The main folder that contains the application i.e. ScientificCalculator.java

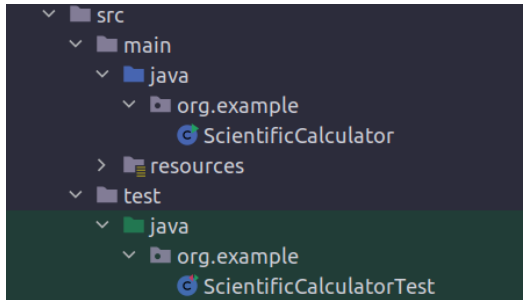
For details view of application see this ->

<https://github.com/samir11096/ScientificCalculator/blob/master/src/main/java/org/example/ScientificCalculator.java>

The test folder contains the unit test file for the application i.e. ScientificCalculatorTest.java

For detailed view of test file see this ->

<https://github.com/samir11096/ScientificCalculator/blob/master/src/test/java/org/example/ScientificCalculatorTest.java>



Dependencies needed are mentioned in **pom.xml**.

For unit testing of our application we will be using Junit. Following is the dependency for the Junit library in our pom.xml

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

For executing our jar file we need something called manifest which specifies the main class of our project. We use the following maven in our pom.xml and specify our main class i.e. **ScientificCalculator**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.5.0</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <archive>
              <manifest>

<mainClass>org.example.ScientificCalculator</mainClass>
              </manifest>
            </archive>
            <descriptorRefs>
              <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>
```

What is the difference between Maven Assembly Plugin and Maven Shade Plugin?

Maven Assembly Plugin is used to create an assembly of your project and its dependencies. This means that it copies all the required files (code, resources, libraries, etc.) into a single directory or archive (such as a JAR or ZIP file) that can be distributed and executed as a standalone application. This is useful for applications that need to be distributed as a single package, rather than requiring the user to manually download and configure dependencies.

Maven Shade Plugin, on the other hand, is used to create a "shaded" JAR file, which is a single JAR file that contains all the code from your project and its dependencies, with conflicting classes renamed to avoid conflicts. This is useful for projects that have dependencies with overlapping class names, which can cause issues when they are combined. The shaded JAR can be used as a drop-in replacement for the original JAR, and it allows you to avoid potential classpath issues that can arise when using multiple JAR files.

In summary, Maven Assembly Plugin is used to create an assembly of your project and its dependencies, while Maven Shade Plugin is used to create a single "shaded" JAR file that contains all the code from your project and its dependencies, with conflicting classes renamed to avoid conflicts.

Source Code Management->

- It is used to keep track of the development line , helping with features such as **rolling back** to previous commits if anything goes wrong in the development cycle.
- For this project we are using **GitHub**.
- Link to repo -> <https://github.com/samir11096/ScientificCalculator>

Jenkins Pipeline ->

The Jenkins Pipeline consists of the following steps

- **Pulling the code from Github to the local machine .**

```
• stage("Git Pull"){
•   steps{
•     git url:
      'https://github.com/samir11096/ScientificCalculator.git',
```

```

●         branch: 'master'
●     }
● }

```

- **Building the Maven Project and running the Tests.**

```

● stage('Maven Build and Test'){
●     steps{
●         sh 'mvn clean install'
●     }
● }

```

- **Building a new Java Docker Image to run our ScientificCalculator**

- We build a docker image of our application ,so that it can run in any server or machine in the form of a container without the need of any external package or dependency.
- To build a new docker image ,we need a file called **DockerFile**.
- We provide the following instruction in the Docker File.

- The image we want to use as the base container image in our application. For this we are using a free source Java Platform openjdk.

```

■ FROM openjdk:11

```

- Then we copy the Jar file of my application using the **COPY** command from the target folder to the docker container image.

```

■ COPY ./target/ScientificCalculator-1.0-SNAPSHOT.jar ./

```

- Finally, we execute the jar file using the **CMD** command , launching our Scientific Calculator application.

```

■ CMD ["java", "-jar",
"ScientificCalculator-1.0-SNAPSHOT.jar"]

```

We then run a pipeline stage in our JenkinsFile to build this Dockerfile and create docker image in our local machine.

```

stage("Build Docker Image"){
    steps{
        sh 'docker build -t samiraghouri/calculator:version1 .'
    }
}

```

- **Push local docker image to DockerHub.**

- Pushing docker image to DockerHub , helps us to access our application docker image from anywhere and run it in any other server.
- To push the docker image to DockerHub we need to add the credentials of our DockerHub to Jenkins Global Credentials.
- For this we go to **Dashboard > Manage Jenkins > Manage Credentials > Global Credentials (Unrestricted)**
- We click on Add Credentials . We then provide the username , password and the ID for which the credentials will be accessed by Jenkins.

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Username ?

samiraghour

☐ Treat username as secret ?

Password ?

 Concealed Change Password

ID ?

docker-hub-creds

Description ?

DockerHub

- Pipeline Stage to push docker image to DockerHub and remove it from local.

```

stage("Push Image to DockerHub"){
    steps {
        withCredentials([usernamePassword(credentialsId:
        'docker-hub-creds', usernameVariable:
        'DOCKERHUB_CREDENTIALS_USR', passwordVariable:
        'DOCKERHUB_CREDENTIALS_PSSW')]) {
            sh "docker login -u
            ${DOCKERHUB_CREDENTIALS_USR} -p
            ${DOCKERHUB_CREDENTIALS_PSSW}"
            sh "docker push
            ${DOCKERHUB_CREDENTIALS_USR}/calculator:version1"
        }
        sh 'docker rmi samiraghour/calculator:version1'
    }
}

```



```
○  
○  
○ }
```

○

- **Deploy the Application to another server /machine using Ansible.**

- Ansible is an open-source automation tool used for configuring and managing systems, applications, and infrastructure.
- Ansible works by using SSH to remotely connect to and execute commands on managed hosts. It uses YAML, a human-readable data serialization language, to define the desired state of a system or application. Ansible then applies these configurations to the managed hosts to ensure that they are in the desired state.
- To help ansible work via SSH , we establish SSH connection between host and the target machine using OpenSSH.
- OpenSSH is a suite of secure networking utilities based on the Secure Shell protocol, which provides a secure channel over an unsecured network in a client–server architecture.
- To generate an SSH key pair, follow these steps:
 - Open a terminal window on your local machine.
 - Type the following command: `ssh-keygen`.
 - Press Enter to accept the default location for the key pair files.
 - Enter a passphrase when prompted, or leave it blank for no passphrase. A passphrase adds an extra layer of security to your SSH key pair, but it is not required.
 - The command will generate a public key file (ending in `.pub`) and a private key file (with no file extension).
 - Your public key can be shared with anyone you want to grant access to your server. The private key should be kept secure and never shared with anyone.
- We can check our connection by doing ssh to the target machine.

```
samir@swift-sf314-52:~$ ssh ansible_usr@127.0.0.1  
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-35-generic x86_64)
```

○

- After our ssh connection is ready with the target machine we define the Jenkins pipeline stage for Ansible Deployment.
- Jenkins Pipeline syntax looks like following:

```

    }
    stage("Ansible Deployment"){
        steps{
            ansiblePlaybook credentialsId: 'private-key', dis
            sh 'ansible-playbook -i dev.inv playbook.yml'
        }
    }
}

```

- We create the inventory file(**dev.inv**) containing the target IP , target username , target password.

```

[webserver]
127.0.0.1 ansible_user=ansible_usr ansible_ssh_pass=1234

```

- We create the playbook(**playbook.yml**) for the same inventory file which will configure our target system.
- In the playbook we define following tasks:

- To pull the docker images to target machine

```

- name: Pull an image
  docker_image:
    name: "{{dockerhub_username}}/{{docker_image_name}}"
    source: pull

```

- To create a container out of the pulled images in the target machine and use interactive shell to run it.

```

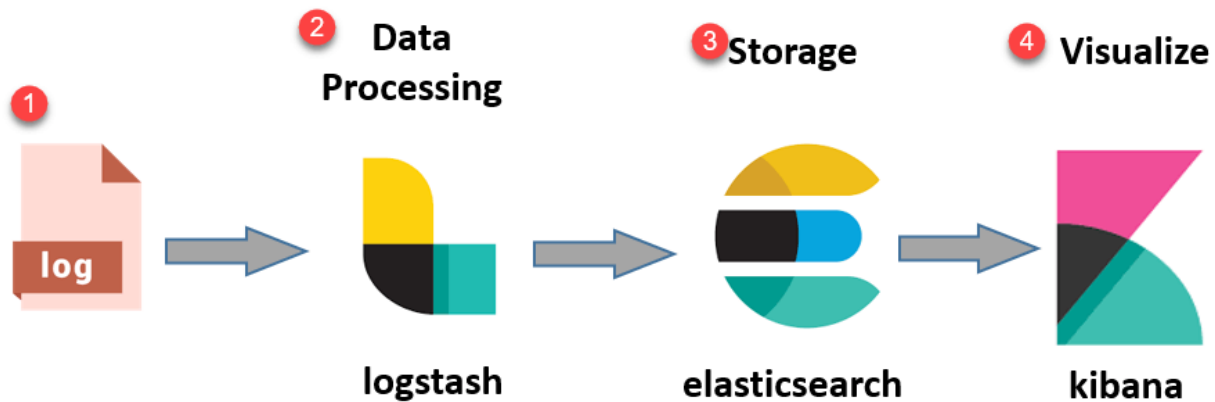
- name: Create docker container with interactive shell
  docker_container:
    name: "{{ docker_container_name }}"
    image: "{{ dockerhub_username }}/{{
docker_image_name }}"
    state: present
    tty: true
    interactive: true

```

- Here tty stands for teletype and it allocated a pseudo-tty for the container .This is needed to allow the container to interact with the user's terminal and show output on the screen.

- Interactive specifies that the container should run in interactive mode allowing the users to interact with the container's shell.

After completing the Jenkins Pipeline ,we add logging functionality in our maven project. This logging functionality will help to us to monitor our application using **ELK Stack**.



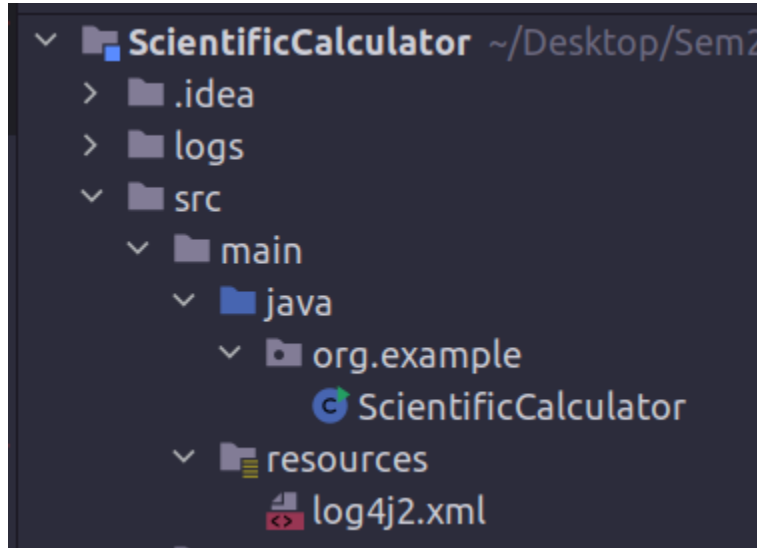
© guru99.com

For the sake of generating log files we add the following dependency in our **pom.xml** file.

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.13.0</version>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
  <version>2.13.0</version>
</dependency>
```

log4j-api, as its name says, is an API. So basically it just contains interfaces that you can use to interface your code with the log4j framework. *log4j-core* is the implementation of this interface, so it contains actual code. It implements every interface in the API.

We then add a configuration file for log4j i.e. log4j2.xml. This is added in our resource package of the project.



This is the main configuration file having all runtime configurations used by log4j. This file will contain log4j appenders information, log level information and output file names for file appenders.

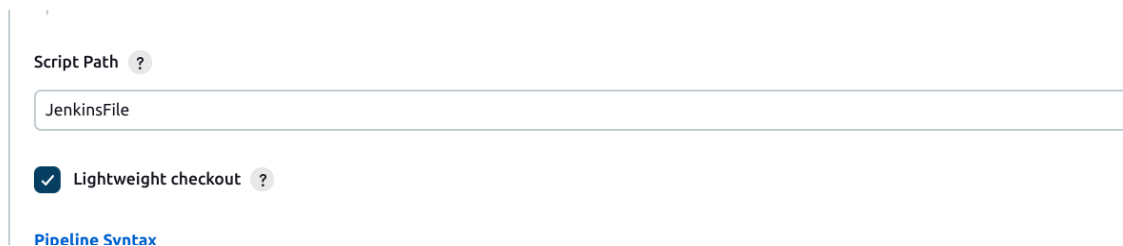
```
<Appenders>
  <Console name="Console" target="SYSTEM_OUT">
    <PatternLayout pattern="%d %p %c{1.} [%t] %m%n" />
  </Console>
  <RollingFile name="RollingFile" fileName="logs/app.log"
    filePattern="logs/app-%d{MM-dd-yy-HH-mm-ss}.log.gz">
    <PatternLayout>
      <pattern>%d %p %c{1.} [%t] %m%n</pattern>
    </PatternLayout>
    <Policies>
      <SizeBasedTriggeringPolicy size="10 MB" />
    </Policies>
    <DefaultRolloverStrategy max="20" />
  </RollingFile>
</Appenders>
```

Log4j provides Appender objects which are primarily responsible for printing logging messages to different destinations such as console, files, NT event logs, Swing components, JMS, remote UNIX syslog daemons, sockets, etc.

After all the configurations and setup for our logger, we have to put logging functionality to our main application . Following is the snippet of code for the same:

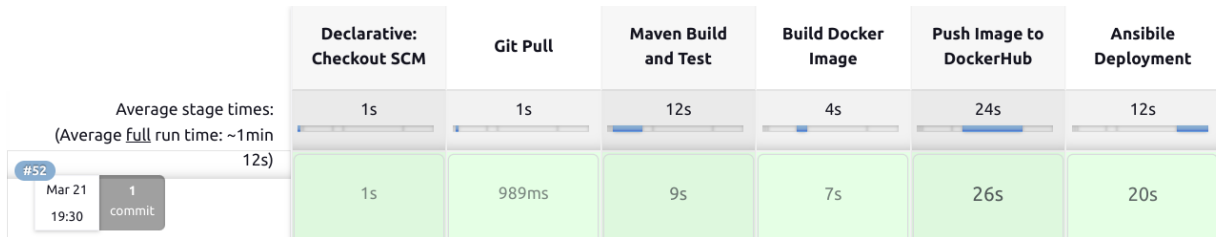
```
try {
    int choice = scanner.nextInt();
    switch (choice) {
        case 1:
            System.out.print("Enter a number: ");
            double x = scanner.nextDouble();
            logger.info(s: "[SQUARE-ROOT-INPUT] - "+String.format("%.2f",x));
            double result = squareRoot(x);
            System.out.printf("The square root of %.2f is %.2f\n.",x,result);
            logger.info(s: "[SQUARE-ROOT-OUTPUT] - "+String.format("%.2f", result));
            break;
    }
}
```

We now push our whole project on Github and provide Jenkins access to it either by keeping it public or providing credentials for the same. We also specify in Jenkins the path of our JenkinsFile along with the github repos url.



The screenshot shows the Jenkins Pipeline configuration interface. At the top, there is a 'Script Path' field with a question mark icon, containing the text 'JenkinsFile'. Below this, there is a checkbox labeled 'Lightweight checkout' with a question mark icon, which is currently checked. At the bottom, there is a link labeled 'Pipeline Syntax'.

After this we Click and Build in our Jenkins Pipeline Project and the following pipeline stages are executed in a sequential manner.:



We can now go to our target machine which must have docker already installed in it. We if the required docker images and docker container are created.

```
ansible_usr@swift-sf314-52:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
samiraghouri/calculator	version1	a5edbb7fcd7f	About an hour ago	656MB
kayrohit/scientific_calc	latest	e580f3774608	2 weeks ago	654MB
docker-elk_setup	latest	968dca2aa589	3 weeks ago	1.29GB
docker.elastic.co/kibana/kibana	8.6.2	65e53ffb7df5	5 weeks ago	727MB
docker-elk_kibana	latest	65e53ffb7df5	5 weeks ago	727MB
docker.elastic.co/elasticsearch/elasticsearch	8.6.2	04485c81cc2d	5 weeks ago	1.29GB
docker-elk_elasticsearch	latest	04485c81cc2d	5 weeks ago	1.29GB
docker.elastic.co/logstash/logstash	8.6.2	5bc835694772	5 weeks ago	732MB
docker-elk_logstash	latest	5bc835694772	5 weeks ago	732MB

```
ansible_usr@swift-sf314-52:~$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
37d59c9da95c	samiraghouri/calculator:version1	"java -jar Scientifi..."	About an hour ago	Exited (0) About an hour ago		calcy

We execute the docker container using the docker start command .

```
ansible_usr@swift-sf314-52:~$ docker start -ai calcy
```

```
Scientific Calculator
1. Square root
2. Factorial
3. Natural logarithm
4. Power
5. Exit
Enter your choice:

```

Once we use the application and exit it . The logs for our application has been generated in the docker container itself. The only thing that we need to do is copy the log from the container in our desired location in the target machine .

```
Goodbye!
ansible_usr@swift-sf314-52:~$ docker cp calcy:/logs/app.log /home/ansible_usr/myapp.log
ansible_usr@swift-sf314-52:~$
```

Now we can go to the above mentioned location and see our logs for the application.

```

ansible_usr@swift-sf314-52:~$ ls
Desktop Documents Downloads Music myapp.log mylog.log Pictures Public Templates ty Videos
ansible_usr@swift-sf314-52:~$ cat mylog.log
2023-03-21 14:02:22,230 INFO o.e.ScientificCalculator [main] [SQUARE-ROOT-INPUT] - 2.00
2023-03-21 14:02:22,243 INFO o.e.ScientificCalculator [main] [SQUARE-ROOT-OUTPUT] - 1.41
2023-03-21 14:02:24,624 ERROR o.e.ScientificCalculator [main] [ERROR-OPTION] - Error while taking input f
java.util.InputMismatchException: null
    at java.util.Scanner.throwFor(Scanner.java:939) ~[?:?]
    at java.util.Scanner.next(Scanner.java:1594) ~[?:?]
    at java.util.Scanner.nextInt(Scanner.java:2258) ~[?:?]
    at java.util.Scanner.nextInt(Scanner.java:2212) ~[?:?]
    at org.example.ScientificCalculator.main(ScientificCalculator.java:27) [ScientificCalculator-1.0-SNAPSHOT.jar:27]
2023-03-21 14:02:27,858 INFO o.e.ScientificCalculator [main] [NATURAL-LOG-INPUT] - 4.00
2023-03-21 14:02:27,866 INFO o.e.ScientificCalculator [main] [NATURAL-LOG-OUTPUT] - 1.39

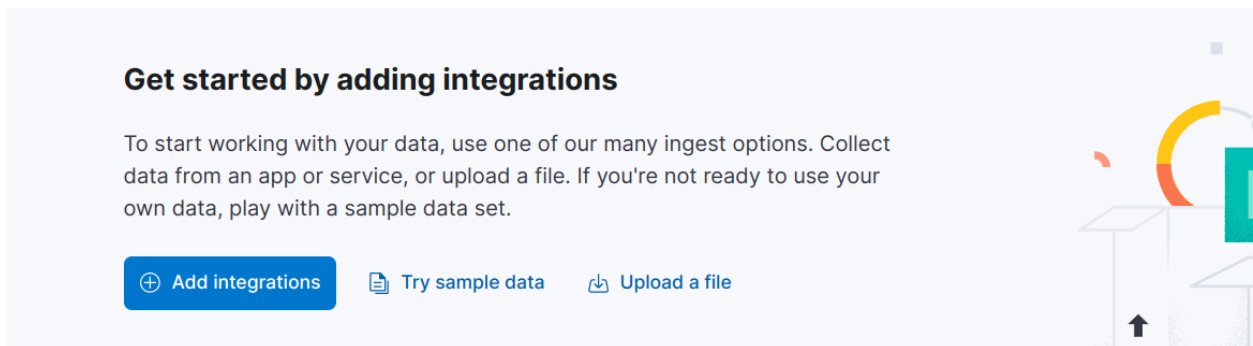
```

We take this log file and give it to ELK Stack for monitoring purpose.

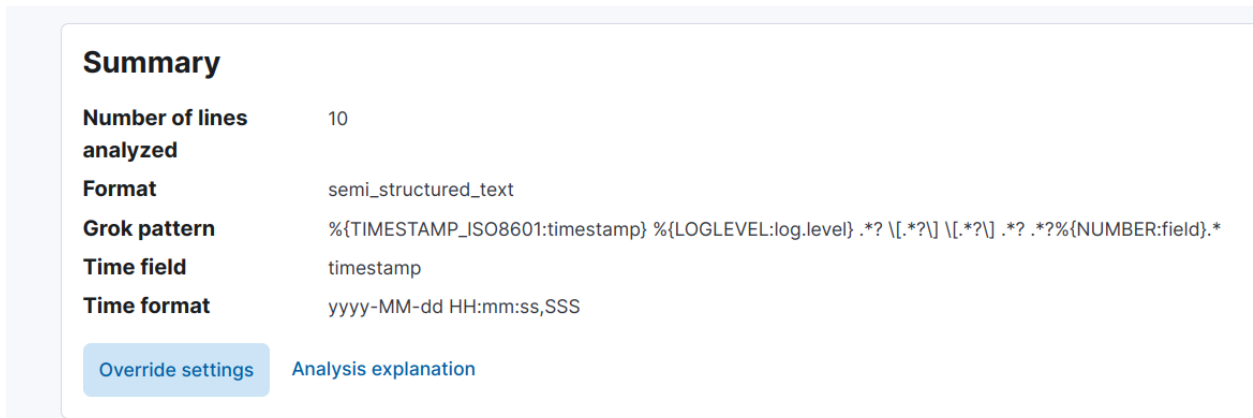
ELK Stack

For this we need to create an account in the <https://www.elastic.co/>

We can also install the ELK dashboard in our local machine but due to laptop hardware restriction we would be using the online version of it.



While uploading our log file we will have an option to give our own grok pattern by overriding the settings..



Grok is a tool that can be used to extract structured data out of a given text field within a document. You define a field to extract data from, as well as the Grok pattern for the match. Grok sits on top of regular expressions. However, unlike regular expressions, Grok patterns are made up of reusable patterns, which can themselves be composed of other Grok patterns.

After selecting our desired grok pattern , we need create an index database

mylog.log

Import data
[Simple](#) [Advanced](#)

Index name
ex

☒ Create data view

Reset

✓

File processed

✓

Index created

✓

Ingest pipeline created

✓

Data uploaded

✓

Data view created

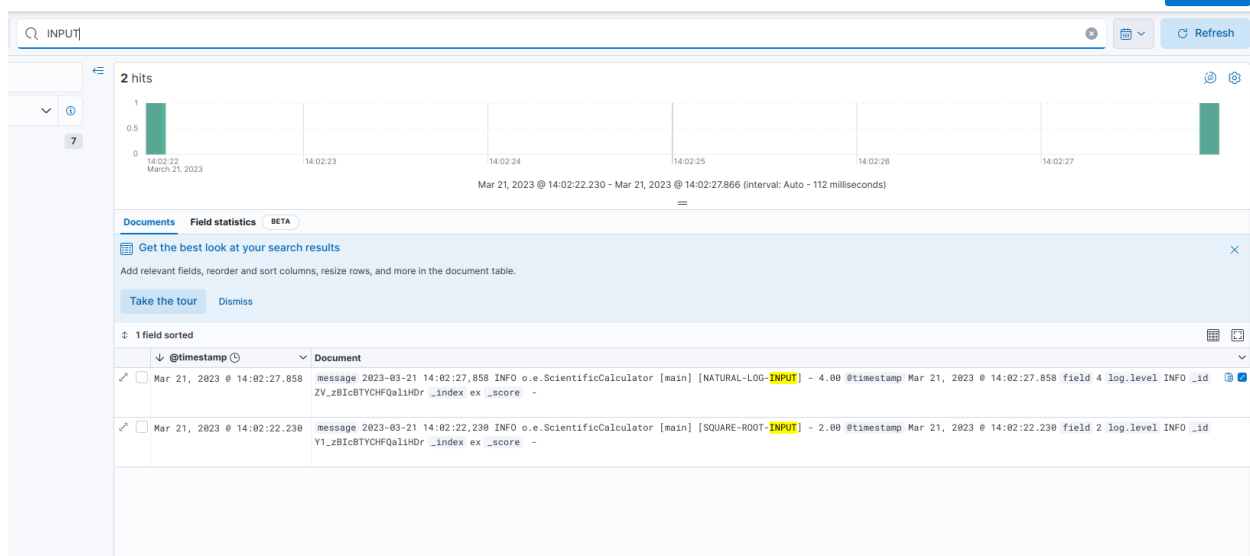
✓ Import complete

⚠️ Some documents could not be imported

Kibana(Discover Tab)

We then go to Discover for any search related option in our index database created from the log file. With Discover, you can quickly search and filter your data, get information about the structure of the fields, and display your findings in a visualization. You can also customize and save your searches and place them on a dashboard.

Now if we try to find all the logs information of let say **INPUT** in our log file we can



We can see that each of our log messages is stored in key value format with all the necessary information in the index database.

```
1  {
2    "@timestamp": [
3      "2023-03-21T08:32:27.858Z"
4    ],
5    "field": [
6      4
7    ],
8    "log.level": [
9      "INFO"
10   ],
11   "message": [
12     "2023-03-21 14:02:27,858 INFO o.e.ScientificCalculator [main] [NATURAL-LOG-INPUT] - 4.00"
13   ],
14   "_id": "ZV_zBIcBTYCHFQaliHDr",
15   "_index": "ex",
16   "_score": null
17 }
```

We can also customize our Kibana Dashboard according to our needs and requirements.