SPE Mini Project Report

IMT2018066
Salman.Patel@iiitb.ac.in

***Abstract* – This project's main goal is to create a scientific calculator programme using DevOps tools. The goal is to learn, understand, and apply DevOps tools to Continuous Integration, Continuous Deployment, and Continuous Monitoring.**

***Keywords* – Maven, Jenkins, GitHub, Intellij IDEA, Ansible, Docker, ELK stack**

GitHub Repository:

github.com/mdsalman991/CalculatorUsingDevops

Docker Repository:

hub.docker.com/repository/docker/kimchi1503/calculator

Docker Image:

Cmd: docker pull kimchi1503/calculator:latest

Note: By pulling this image the calculator instance can be run on your local system.

I. DevOps Tool Chain

This project mainly focuses on building the Calculator project using Devops framework.

- You can use any set of DevOps tool chains you want, but the pipeline would be the same. The pipeline includes
- Using a source control management tool - like GitHub, GitLab, BitBucket etc
- Testing - test your code using either JUnit, Selenium, PyUnit and many more
- Build - build your code using tool like Maven, Gradle, Ant and many more
- Continuous Integration - Continuous integrate your code using tool like Jenkins, GitLab CLI, Travis CLI, and many more
- Containerize - Containerize your code using Docker
- Push your created Docker image to Docker hub.
- Deployment - Do configuration management and deployment using either Chef, Puppet, Ansible, Rundeck. Using these do configuration management and pull your docker image and run it on the managed hosts.
- For Deployment you can either do it on your local machine or on Kubernetes cluster or OpenStack cloud. You can also use Amazon AWS or Google Cloud or some other 3rd party cloud.
- Monitoring - for monitoring use the ELK stack. Use a log file to do the monitoring. Generate the log file for your mini project and pass in your ELK stack.

Tools Used:

These are the tools used for building the whole pipeline and monitoring it.

Java(JDK8), Maven, Intellij IDEA, Git, GitHub, JUnit, Log4j, Docker, DockerHub, Jenkins, Ansible, Amazon Aws, ELK Stack

Part-1 : Setting up Tools:

'Java(JDK8)': Java is a class-based, object-oriented programming language with a low number of implementation dependencies.

Installation and Configuration on mac:

- Download open the jdk.dmg extension file from the web browser.
- Configure the JAVA_HOME path in .bash_profile file.
- Add this path at the end of the file.
- export JAVA_HOME =$(/usr/libexec/java_home)
- cmd source .bash_profile.
- Verify JAVA_HOME path.
- cmd echo $JAVA_HOME

```
#JAVA
export JAVA_HOME=$(/usr/libexec/java_home)
```

Fig. 1. Java path in bash_profile

```
salmanpatel@Salmans-MacBook-Pro ~ % /usr/libexec/java_home
/Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home
salmanpatel@Salmans-MacBook-Pro ~ % mate .bash_profile
salmanpatel@Salmans-MacBook-Pro ~ % source .bash_profile
salmanpatel@Salmans-MacBook-Pro ~ % echo $JAVA_HOME
/Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home
```

Fig. 2. Echo Java path

```
salmanpatel@Salmans-MacBook-Pro ~ % java -version
java version "1.8.0_321"
Java(TM) SE Runtime Environment (build 1.8.0_321-b07)
Java HotSpot(TM) 64-Bit Server VM (build 25.321-b07, mixed mode)
```

Fig. 3. Java version

'Maven' is a tool for project management and development. Plugins can extend Maven to use a variety of other development tools for reporting or the build process.

Installation and Configuration on mac:

Download maven from the official page. Or For mac we can use brew install

Cmd: brew install maven

```
salmanpatel@Salmans-MacBook-Pro ~ % mvn -version
Apache Maven 3.8.5 (3599d3414f046de2324203b78ddcf9b5e4388aa0)
Maven home: /usr/local/Cellar/maven/3.8.5/libexec
Java version: 1.8.0_321, vendor: Oracle Corporation, runtime: /Library/Internet
Plug-Ins/JavaAppletPlugin.plugin/Contents/Home
Default locale: en_IN, platform encoding: UTF-8
OS name: "mac os x", version: "12.3.1", arch: "x86_64", family: "mac"
```

Fig. 4. Maven version

'Intellij IDEA' is an integrated development environment written in java for developing software.

Installation and Configuration on mac:

- Download Intellij IDEA from jetbrains.com/download
- Extract it and it is ready to run!
- Go to settings/build, executions, deployment/build tools/maven
- Select use plugin registry then click on save.

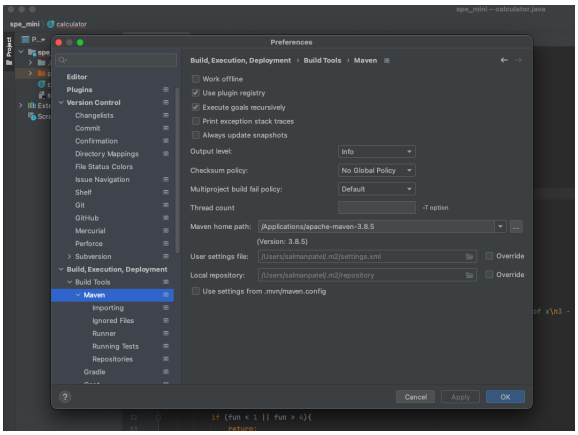


Fig. 5. Maven plugin registry

- Signup at github.
- Create a new repository.
- Enter repo name and click create.
- In Intellij go to setting/Version Control/GitHub and click +
- Log via Github or token

'Git' is a distributed version control system, It is a tool to manage project source code history. Git is one of the most widely-used popular version control systems in use today.

Installation and Configuration on mac:

Install Git on Mac OS X. There are several ways to install Git on a Mac. In fact, if you've installed XCode (or it's Command Line Tools), Git may already be installed.

git config –global user.name "Your username".

git config –global user.email "Your email"

git -–version

```
salmanpatel@Salmans-MacBook-Pro ~ % git --version
git version 2.32.0 (Apple Git-132)
```

Fig. 6. Git version

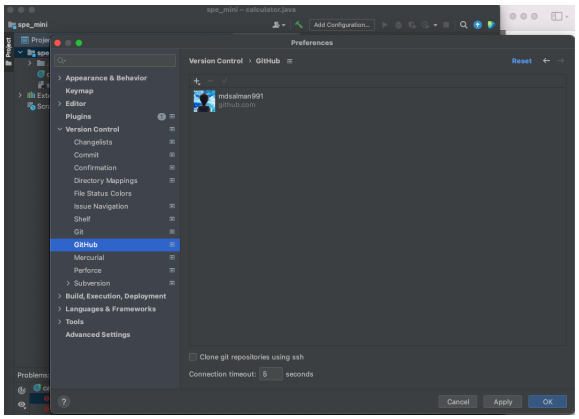


Fig. 7. Version Control Settings in Intellij IDEA

'GitHub' Create a repository in Github for the project and add the remote repository to the project that we have created locally.

- git init
- git add .
- git commit -m "Added all the src files along with log code"
- git remote add origin "respective repo link"
- git push -u origin master

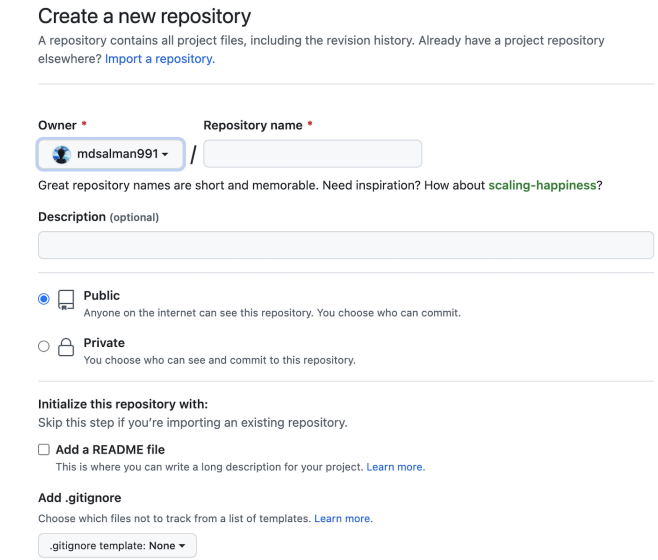
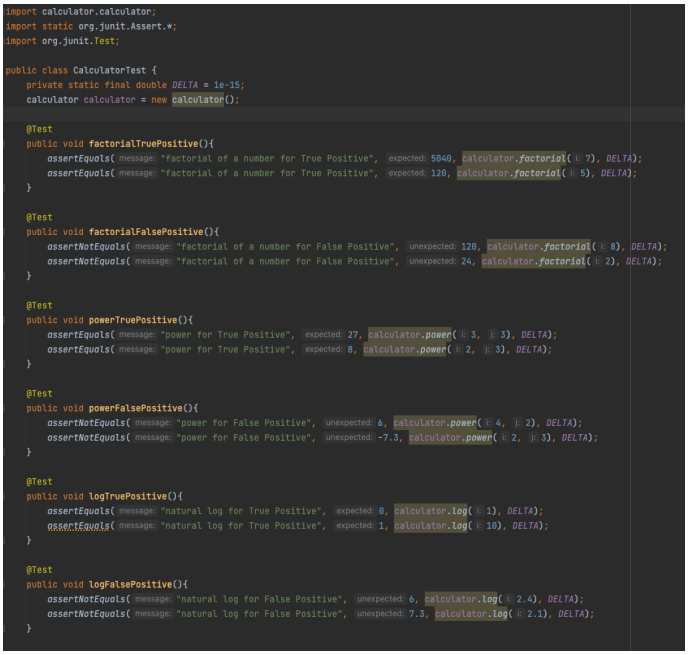


Fig. 8. GitHub Create new Repo

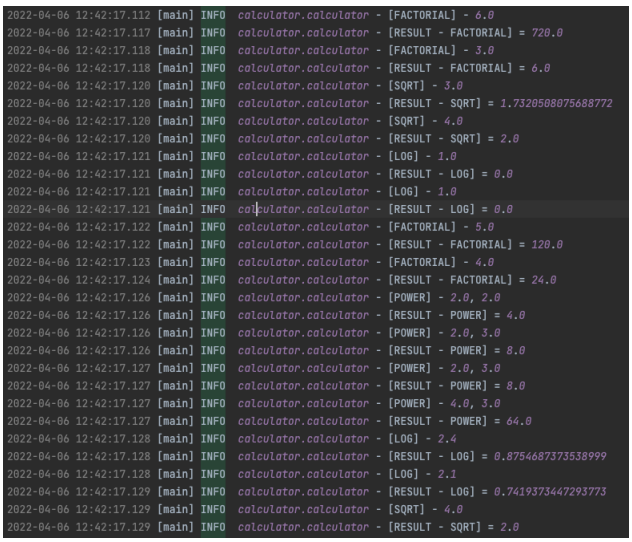
'Junint' is a framework for unit testing in the Java programming language. A JUnit test fixture is a Java object that enables the creation of multiple test cases. @Test must be used to annotate test methods. If necessary, the @BeforeEach (or @AfterEach) and @BeforeAll (or @AfterAll) annotations can be used to define a method to run before (or after) each (or all) of the test methods.



```
import calculator.calculator;
import static org.junit.Assert.*;
import org.junit.Test;

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

    @Test
    public void factorialTruePositive(){
        assertEquals( message: "factorial of a number for True Positive",  expected: 5040, calculator.factorial( i: 7), DELTA);
        assertEquals( message: "factorial of a number for True Positive",  expected: 120, calculator.factorial( i: 5), DELTA);
    }

    @Test
    public void factorialFalsePositive(){
        assertNotEquals( message: "factorial of a number for False Positive",  unexpected: 120, calculator.factorial( i: 8), DELTA);
        assertNotEquals( message: "factorial of a number for False Positive",  unexpected: 24, calculator.factorial( i: 2), DELTA);
    }

    @Test
    public void powerTruePositive(){
        assertEquals( message: "power for True Positive",  expected: 27, calculator.power( i: 3,  j: 3), DELTA);
        assertEquals( message: "power for True Positive",  expected: 8, calculator.power( i: 2,  j: 3), DELTA);
    }

    @Test
    public void powerFalsePositive(){
        assertNotEquals( message: "power for False Positive",  unexpected: 6, calculator.power( i: 4,  j: 2), DELTA);
        assertNotEquals( message: "power for False Positive",  unexpected: -7.3, calculator.power( i: 2,  j: 3), DELTA);
    }

    @Test
    public void logTruePositive(){
        assertEquals( message: "natural log for True Positive",  expected: 0, calculator.log( i: 1), DELTA);
        assertEquals( message: "natural log for True Positive",  expected: 1, calculator.log( i: 10), DELTA);
    }

    @Test
    public void logFalsePositive(){
        assertNotEquals( message: "natural log for False Positive",  unexpected: 6, calculator.log( i: 2.4), DELTA);
        assertNotEquals( message: "natural log for False Positive",  unexpected: 7.3, calculator.log( i: 2.1), DELTA);
    }
```

Fig. 9. Junit Testing Example

'Log4j' Developers use Log4j to track what happens in their software applications or online services. It's essentially a massive log of a system's or application's activity. This activity is known as logging, and it is used by developers to keep an eye out for problems that users may encounter.



```
2022-04-06 12:42:17.112 [main] INFO  calculator.calculator - [FACTORIAL] - 6.0
2022-04-06 12:42:17.117 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 720.0
2022-04-06 12:42:17.118 [main] INFO  calculator.calculator - [FACTORIAL] - 3.0
2022-04-06 12:42:17.118 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 6.0
2022-04-06 12:42:17.120 [main] INFO  calculator.calculator - [SQRT] - 3.0
2022-04-06 12:42:17.120 [main] INFO  calculator.calculator - [RESULT - SQRT] = 1.7320508075688772
2022-04-06 12:42:17.120 [main] INFO  calculator.calculator - [SQRT] - 4.0
2022-04-06 12:42:17.120 [main] INFO  calculator.calculator - [RESULT - SQRT] = 2.0
2022-04-06 12:42:17.121 [main] INFO  calculator.calculator - [LOG] - 1.0
2022-04-06 12:42:17.121 [main] INFO  calculator.calculator - [RESULT - LOG] = 0.0
2022-04-06 12:42:17.121 [main] INFO  calculator.calculator - [LOG] - 1.0
2022-04-06 12:42:17.121 [main] INFO  calculator.calculator - [RESULT - LOG] = 0.0
2022-04-06 12:42:17.122 [main] INFO  calculator.calculator - [FACTORIAL] - 5.0
2022-04-06 12:42:17.122 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 120.0
2022-04-06 12:42:17.123 [main] INFO  calculator.calculator - [FACTORIAL] - 4.0
2022-04-06 12:42:17.124 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 24.0
2022-04-06 12:42:17.126 [main] INFO  calculator.calculator - [POWER] - 2.0, 2.0
2022-04-06 12:42:17.126 [main] INFO  calculator.calculator - [RESULT - POWER] = 4.0
2022-04-06 12:42:17.126 [main] INFO  calculator.calculator - [POWER] - 2.0, 3.0
2022-04-06 12:42:17.126 [main] INFO  calculator.calculator - [RESULT - POWER] = 8.0
2022-04-06 12:42:17.127 [main] INFO  calculator.calculator - [POWER] - 2.0, 3.0
2022-04-06 12:42:17.127 [main] INFO  calculator.calculator - [RESULT - POWER] = 8.0
2022-04-06 12:42:17.127 [main] INFO  calculator.calculator - [POWER] - 4.0, 3.0
2022-04-06 12:42:17.127 [main] INFO  calculator.calculator - [RESULT - POWER] = 64.0
2022-04-06 12:42:17.128 [main] INFO  calculator.calculator - [LOG] - 2.4
2022-04-06 12:42:17.128 [main] INFO  calculator.calculator - [RESULT - LOG] = 0.8754687373538999
2022-04-06 12:42:17.128 [main] INFO  calculator.calculator - [LOG] - 2.1
2022-04-06 12:42:17.129 [main] INFO  calculator.calculator - [RESULT - LOG] = 0.7419373447293773
2022-04-06 12:42:17.129 [main] INFO  calculator.calculator - [SQRT] - 4.0
2022-04-06 12:42:17.129 [main] INFO  calculator.calculator - [RESULT - SQRT] = 2.0
```

Fig. 10. Calculator log output file

'Jenkins' Jenkins is a Java-based open source automation tool with Continuous Integration plugins. Jenkins is a tool for developing and testing software projects. It enables developers to deliver software on a continuous basis by integrating with a wide range of testing and deployment technologies.

Installation and Configuration on mac:

Sample commands:

- Install the latest LTS version: *brew install jenkins-lts*
- Start the Jenkins service: *brew services start jenkins-lts*
- Restart the Jenkins service: *brew services restart jenkins-lts*

After starting the Jenkins service, browse to localhost:8080 and follow the instructions to complete the installation.

Print default password to login into jenkins

```
Salmans-MacBook-Pro:secrets salmanpatel$ cd /Users/salmanpatel/.jenkins/secrets
Salmans-MacBook-Pro:secrets salmanpatel$ ls
initialAdminPassword                                    master.key
jenkins.model.Jenkins.crumbSalt                         org.jenkinsci.main.modules.instance_identity.InstanceIdentity.KEY
Salmans-MacBook-Pro:secrets salmanpatel$ cat initialAdminPassword
3817cefd35d44c359311800411ffb164
```

Fig. 11. Jenkins Default Password

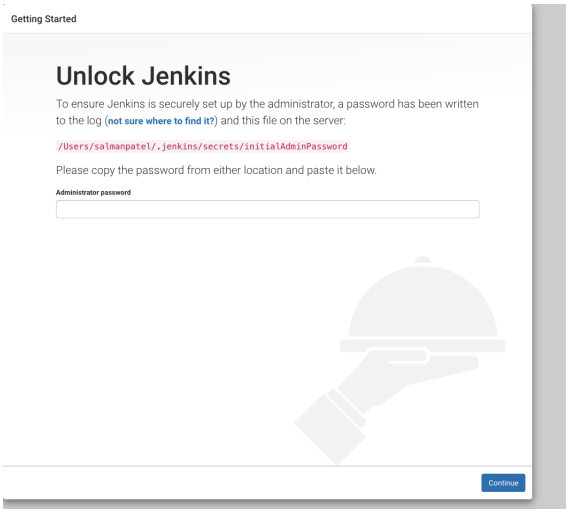


Fig. 12. Paste the above copied Password

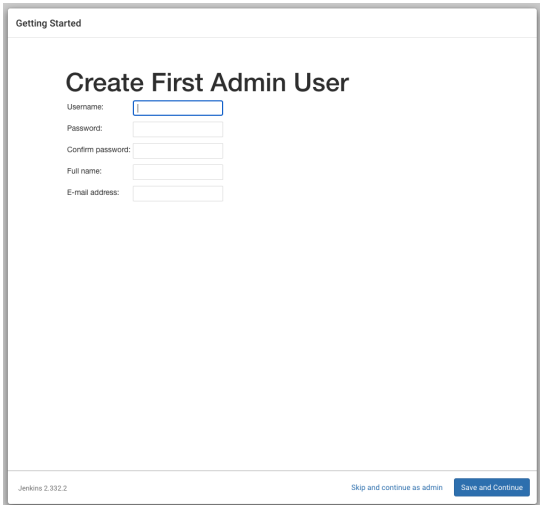


Fig. 13. Jenkins Sign-in Dashboard

Next, Choose install suggested plugins, configure user.

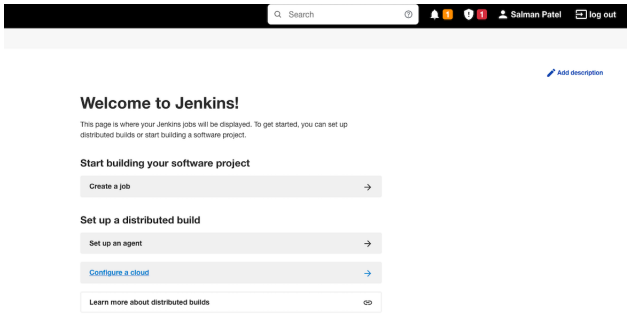


Fig. 14. Jenkins Dashboard

'Docker' Docker is an open source platform for developing, shipping and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.

Installation and Configuration on mac:

```
salmanpatel@Salmans-MacBook-Pro ~ % docker --version
Docker version 20.10.13, build a224086
```

Fig. 15. Docker Version

- *docker build -t kimchi1503/calculator*
- *docker push kimchi1503/calculator*
- *docker pull kimchi1503/calculator*
- *docker run -it kimchi1503/calculator*

'Docker Hub' offers an easy way to create, manage, and deliver teams' container applications. Here, we can find official images created by companies as well as customized images from different users. We create our own repository.

- Creating an account at DockerHub and creating a repository:
- Signin into https://hub.docker.com/ account create a repository.

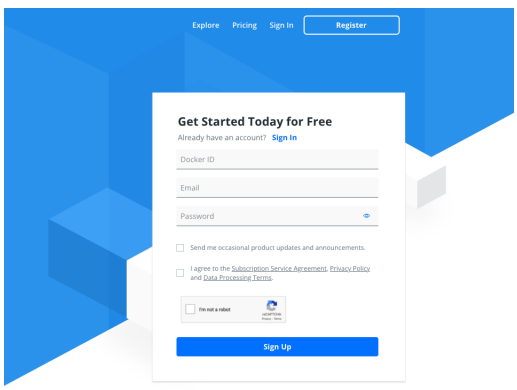


Fig. 16. DockerHub Signin

'Ansible' is an Open Source automation platform, an Automation Engine that runs ansible playbooks. Playbooks are defined tasks, where we define environments and workflows. Ansible connects to the hosts it manages using openssh or winrm and run tasks. These tasks are small programs pushed to the hosts.

Before installing ansible, python and ssh must be installed.

python installation:

brew install pyenv
pyenv install 3.9.2

ssh installation:

brew install hudochenkov/sshpass/sshpass

Ansible installation:

brew install ansible

```
salmanpatel@Salmans-MacBook-Pro ~ % ansible --version
ansible [core 2.12.4]
  config file = None
  configured module search path = ['/Users/salmanpatel/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/Cellar/ansible/5.6.0/libexec/lib/python3.10/site-packages/ansible
  ansible collection location = /Users/salmanpatel/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.10.2 (main, Feb  2 2022, 06:19:27) [Clang 13.0.0 (clang-1300.0.29.3)]
  jinja version = 3.1.1
  libyaml = True
```

Fig. 17. Ansible version

Implementing Playbooks

As a one-time command, ad hoc commands can perform a single, simple task against a set of targeted hosts. However, the true power of Ansible lies in learning how to use playbooks to run multiple, complex tasks against a set of targeted hosts in an easily repeatable manner. A play is an ordered set of tasks that are run against hosts from your inventory. A playbook is a text file that contains a list of one or more plays that must be executed in a specific order. A playbook is a text file saved with the extension yml that is written in YAML format. To indicate the structure of its data, the playbook employs indentation with space characters.

—

- name: Run id to playbook hosts: all

tasks:

- name: Run id command command:

cmd: id

'Amazon Web Services' is an online platform that provides scalable and cost-effective cloud computing solutions.

AWS is a broadly adopted cloud platform that offers several on-demand operations like compute power, database storage, content delivery, etc., to help corporates scale and grow.

Open the Amazon Web Services (AWS) home page.

Choose Create an AWS Account.

AWS Educate $100: In order to receive the benefit, the prospective members must be eligible for AWS Educate, and their application must be approved.

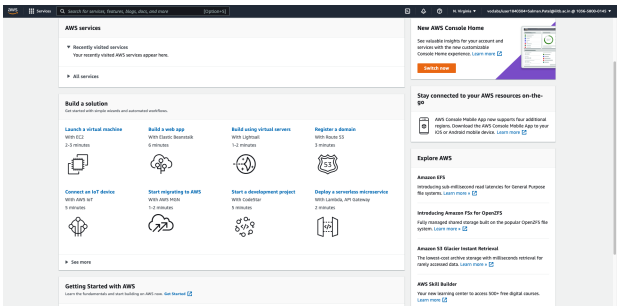


Fig. 18. Amazon Web Services

Configuration:

- Signup on https://signin.aws.amazon.com/
- Go to Dashboard, then click on Compute Engine in the menu.
- Click on VM Instances and then CREATE INSTANCE.
- Name your Instance.
- Select Ubuntu 20.04 LTS minimal Boot Disk or any OS of your choice.
- Select allow http traffic.
- Click on create.
- Login to the instance by clicking on SSH in the VM instances section.
- Now install Docker using the same installation procedure as before.

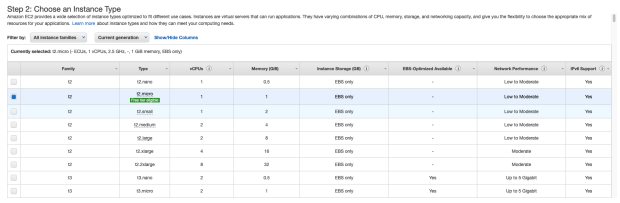


Fig. 19. Choose an Instance

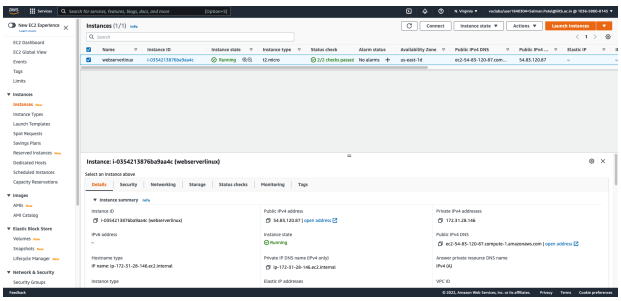


Fig. 20. Instance Running

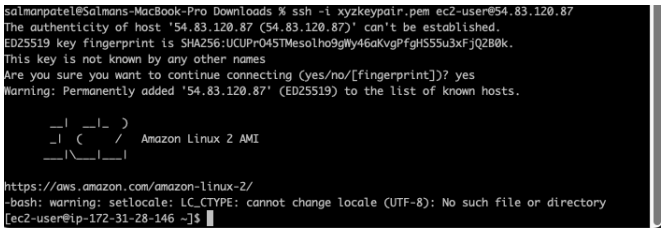


Fig. 21. Amazon Linux 2 Connected through CLI

'ELK Stack' consists of three open source softwares.

- Elasticsearch: It is a search and analytics engine
- Logstash: It is a server side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "Stash" like Elasticsearch
- Kibana: It lets users visualize data with charts and graphs made from elasticsearch

Installation and Configuration on mac:

Download and extract Elasticsearch, Logstash and kibana from the following links

- https://www.elastic.co/downloads/elasticsearch
- https://www.elastic.co/downloads/kibana
- https://www.elastic.co/downloads/logstash

Part-2: Developing the project (CI/CD)

Now that all the tools are set up, we can start developing our project.

Process

- Before we can begin working on the scientific calculator, we must first integrate all of our tools. In part one, we installed and configured all of our tools. We must now integrate them.
- First we have to set up a Jenkins pipeline, it has to pull our project from our github, run it, test it, build it, create a docker file, push it to docker hub and trigger ansible.
- Next, we create a ansible job to deploy the docker image from docker hub onto the deployment server.
- Now we are going to use Intellij to develop a scientific calculator in java. Intellij is integrated with various other tools and utilities such as git, github, maven, log4j, junit.
- Intellij creates a file system for our project initially and we can start adding files to our project at the appropriate places.
  - The Source code i.e the java code, junit and log4j files in src.

  - In the project folder we keep pom.xml, Dockerfile, yml files and other files needed for other tools.

- Every time we update any part of the project we make a commit in the local git.
- Once a stable version is developed we push the local repository code to our github.
- Once the code is pushed a Jenkins pipeline is triggered which starts executing a pre-defined pipeline.
- In our pipeline,
  - First Jenkins pulls the new code from github.
  - It will then compile, run and test the whole project using maven.
  - If the tests are successful it will then build the project and create a docker image based on the Dockerfile.
  - After building a docker image successfully it will then push the image to docker hub and then triggers ansible.
- The ansible job we defined pulls the new docker image onto the deployment server/s and runs them.
- Now, the deployment servers are ready for use and they start creating logs as they are being used.
- Every day, we can download all of the log files to the Continuous Monitoring machine and use the ELK stack to visualise all of the log files. To monitor logs in real time, we can use beats to transfer all log files in real time to the logging/monitoring server.

'Scientific Calculator'

The Scientific calculator has the following functions:

1. Square Root of x
2. Factorial of x
3. Natural log of x
4. x power b

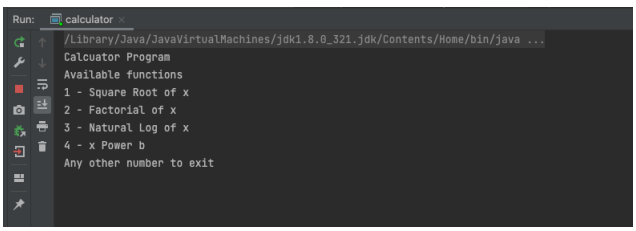


Fig. 22. Calculator through CLI

'Intellij IDEA'

- Setting up a new project
- Open Intellij and select a new project.
- Select maven from the menu.
- Select project sdk 1.8 and click next.
- Give a name and click Finish.

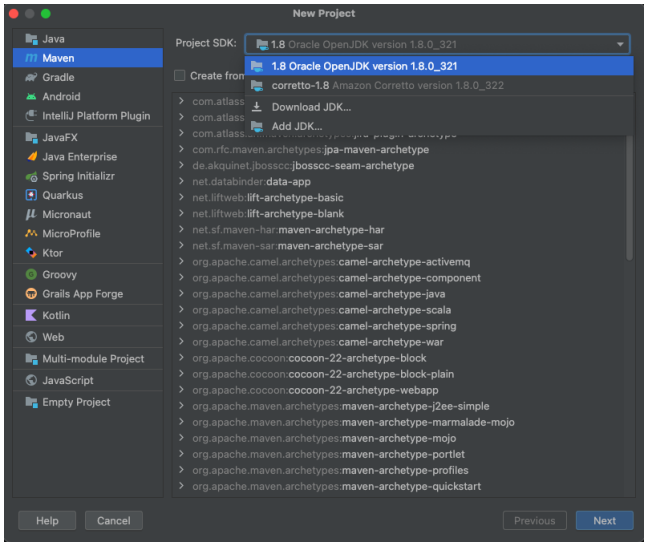


Fig. 23. Maven in Intellij

- Intellij creates a file structure for our project.
- Go to src/main. Right click on java and select new/package.
- Write the scientific calculator code here.
- Next, in src/main/resources create a lo4j2.xml file and write the following code. This file creates a template for all the log files.

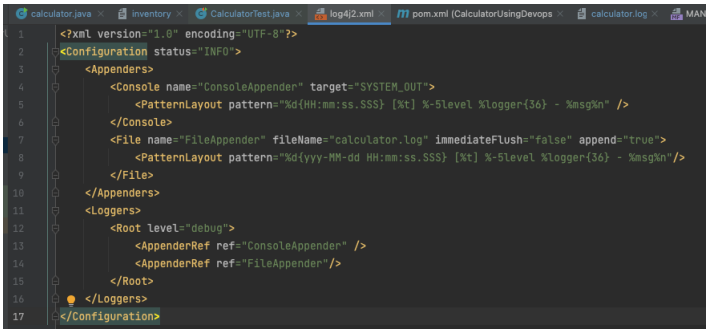


Fig. 24. Log4j2.xml

- Next, in src/java create a new java file to write all the tests.
- Next, in the project folder create the following files:

- Pom.xml
  - Dependencies can be easily added by the generate tool (Right click).
  - All the necessary plugins and dependencies of the java project have to be added here.
  - Maven reads this file and installs all the listed dependencies and plugins.
  - We add a maven assembly plugin to create a jar file. Which can be used to easily execute our project on any computer.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>CalculatorUsingDevops</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <archive>
                                <manifest>
                                    <mainClass>calculator.calculator</mainClass>
                                </manifest>
                            </archive>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>

        </plugins>
    </build>
```

Fig. 25. plugins

```xml
    <dependencies>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.17.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.14.0</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

</project>
```

Fig. 26. Dependencies

- Dockerfile
  - We create a dockerfile to let docker know how to build an image.
  - Here we use an openjdk base image and once the image runs it will copy the calculator jar file and execute it.

```
FROM openjdk:8
MAINTAINER Salman  Salman.Patel@iiitb.org
COPY ./target/CalculatorUsingDevops-1.0-SNAPSHOT-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java", "-cp", "CalculatorUsingDevops-1.0-SNAPSHOT-jar-with-dependencies.jar", "calculator.calculator"]
#SALMAN
```

Fig. 27. Dockerfile

- Logstash.conf
  - This is a configuration file for logstash.

```
input{
    plugin{
        settings
    }
}

filter{
    plugin{
        settings
    }
}

output{
    plugin{
        settings
    }
}
```

Fig. 28. Logstash.conf

- All java packages,test files can be run independently to check for errors at any point of time.
- Once the project is ready, click on build and run the code for testing before committing changes to it.
- After this click on commit and push to push the code to github.

The maven project has a ‘src’ directory which contains all the implementation and test files of our project. In src we have ‘main’ and ‘test’ directories where the main contains all the project java files and test contains all test files. And when we build the maven project a directory/ folder named ‘target’ is formed which contains the .jar/.war packaged file(along with the executable classes etc). For all this to happen the main file required is pom.xml which contains all the dependencies, plugins and properties etc.

Now we have to add all the dependencies in pom.xml as show before, For example I am using Junit4 which has to be added in the xml and For log4j also we have to add the dependencies. Etc.

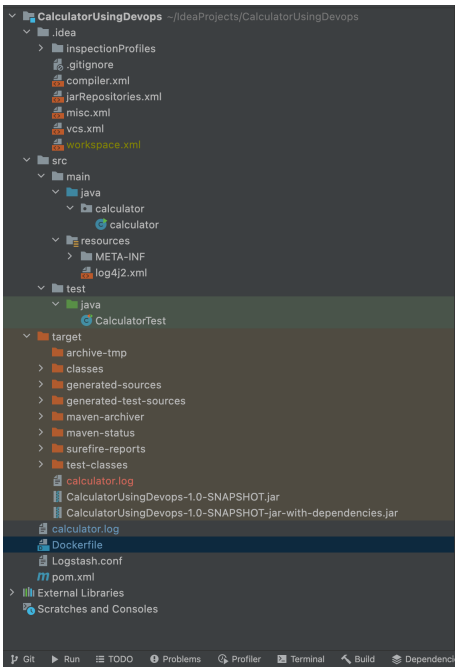


Fig. 29. File structure

'Jenkins Pipeline'

- Login to jenkins on localhost:8080.
- Go to Dashboard -> manage Jenkins -> manage plugins -> available.

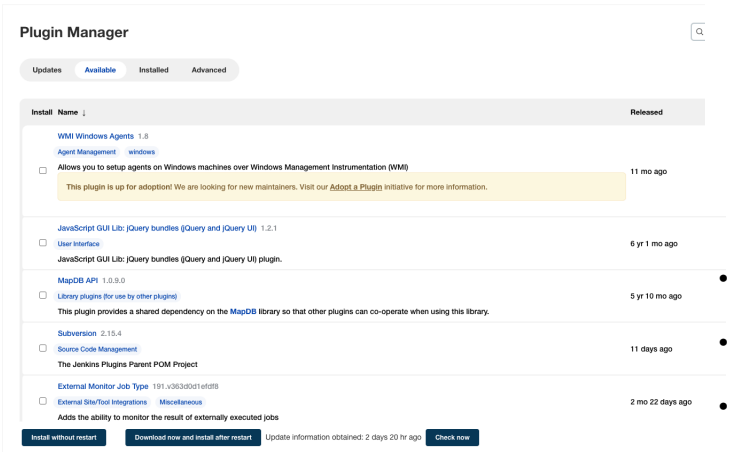


Fig. 30. Available Jenkins Plugins

- Install the following plugins:

  Docker, Docker plugin, Git, Github Integration Plugin, Junit plugin, Maven integration,Pipeline, Ansible plugin
- Click on install without restart after selecting plugins.
- Go to manage jenkins/manage credentials/ jenkins/ Global Credentials/
- Click on Add Credentials.
- Enter Docker Hub username, password, and give an ID to these credentials.

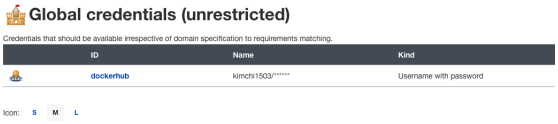


Fig. 31. Global Credentials

- Go to Dashboard/manage Jenkins/Global Tool Configuration

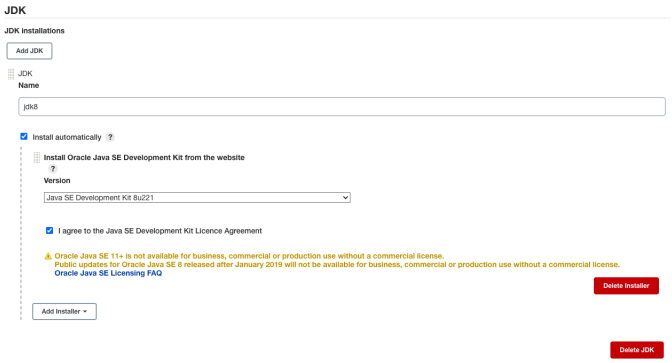


Fig. 32. JDK 8

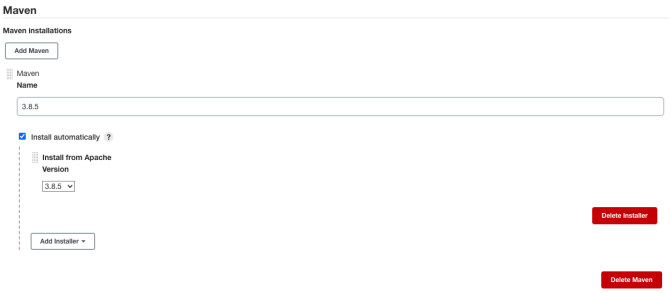


Fig. 33. Maven 3.8.5

- Click on new item -> pipeline.
- Next in the pipeline add the following code.
- Update the values in the pipeline and save.

Fig. 34. Pipeline Script

‘Run Program Locally’

we can clone an already existing maven project and make changes to it.

After this we can make many changes to the maven project but till now we did not integrate jenkins to this. Now we can add test cases to the java project that we have built and use the maven commands to package them locally.

mvn clean install

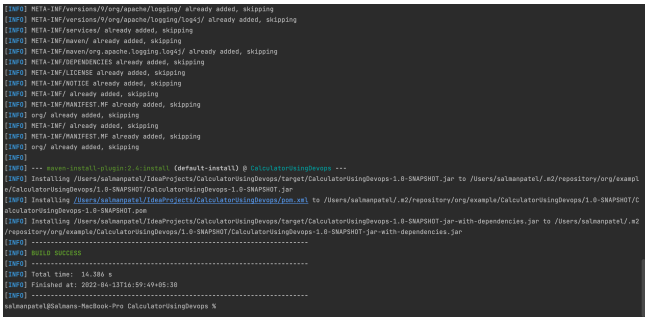


Fig. 35. mvn clean install build sucess

The above commands can be used to package the project with all dependencies.

mvn test

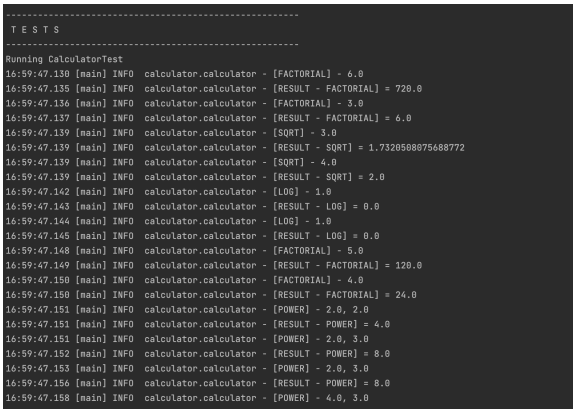


Fig. 36. mvn test

We can use this to test all the Junit test cases that we have written. In Junit we can have many test cases along with the enhanced version of using Parameters for Parameterized class. This image could be seen in the next page which contains the test class for the factorial method of Calculator class.

After the code is being changed we can build the code locally using maven now, (this will be automated later using jenkins) and check whether the package is working fine or not.

CMD: *java -cp CalculatorUsingDevops-1.0-SNAPSHOT-jar-with-dependencies.jar calculator.calculator*

Fig. 37. java run command

Now we will get the menu of our java CLI application which we deploy using the docker image and Ansible.

Here we can also see that we have generated log files using the log4j. We can use these logfiles and filter, Transform and analyse them using the ELK stack which we will discuss later in this report.

Till now we were able to use Github and maven to build and test the project but this happened manually. So we now use Jenkins pipepline script as above for Automated build and test.

Now we can check this docker file by manually building it and then running it as a container. This makes our debugging process easy. So now we have automated build, test and we can add the docker build script in the jenkins script file and push it into docker hub. One way could be to just add the shell commands discussed previously or the other could be to use jenkins script.

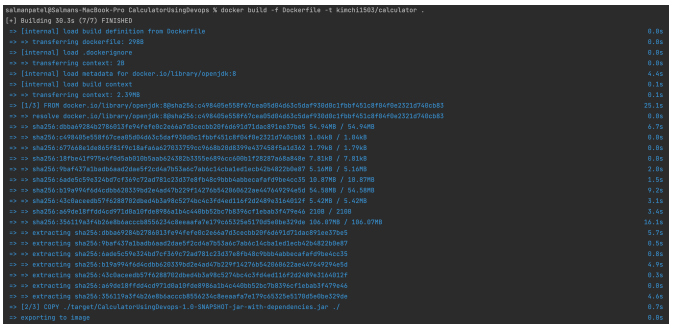


Fig. 38. Docker image Build

Docker Commands run locally

docker push kimchi1503/calculator:latest

Fig. 39. Docker image push

docker pull kimchi1503/calculator:latest

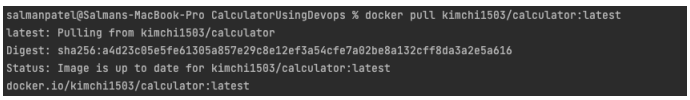


Fig. 40. Pull Dokcer image locally

docker run -it kimchi1503/calculator:latest

```
salmanpatel@Salmans-MacBook-Pro CalculatorUsingDevops % docker run -it kimchi1503/calculator:latest
Calcuator
Available functions
1 - Square Root of x
2 - Factorial of x
3 - Natural Log of x
4 - x Power b
Any other number to exit
2
Enter x
5
14:26:18.279 [main] INFO  calculator.calculator - [FACTORIAL] - 5.0
14:26:18.288 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 120.0
Factorial of x =120.0
Calcuator
Available functions
1 - Square Root of x
2 - Factorial of x
3 - Natural Log of x
4 - x Power b
Any other number to exit
```

Fig. 41. Run Docker Image

'Ansible'

The ansible-playbook command is used to run playbooks. The command is executed on the control node and the name of the playbook to be run is passed as an argument:

$ ansible-playbook playbook_file_name.yml

When you run the playbook, output is generated to show the play and tasks being executed. The output also reports the results of each task executed.

When you rerun the playbook, tasks in Ansible Playbooks are idempotent, and it is safe to run a playbook multiple times. If the targeted managed hosts are already in the correct state, no changes should be made.

Syntax Verification:

$ ansible-playbook –syntax-check playbook_file_name.yml

Executing a dry run:

$ ansible-playbook playbook_file_name.yml -i inventory_file

```
suchit@suchit-Lenovo-ideapad-330-15IKB:~$ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED         SIZE
aryavrdhn/scientificcalc    latest    94eb7b5cf95a   15 hours ago    528MB
kimchi1503/calculator       latest    1177a70b206d   26 hours ago    528MB
```

Fig. 42. Docker image deploy in friends laptop

```
suchit@suchit-Lenovo-ideapad-330-15IKB:~$ docker run -it kimchi1503/calculator:latest
Calcuator
Available functions
1 - Square Root of x
2 - Factorial of x
3 - Natural Log of x
4 - x Power b
Any other number to exit
2
Enter x
5
10:35:45.541 [main] INFO  calculator.calculator - [FACTORIAL] - 5.0
10:35:45.551 [main] INFO  calculator.calculator - [RESULT - FACTORIAL] = 120.0
Factorial of x =120.0
Calcuator
Available functions
1 - Square Root of x
2 - Factorial of x
3 - Natural Log of x
4 - x Power b
Any other number to exit
```

Fig. 43. Run deployed docker image

'Final Step' To automate whole process.

- Go to jenkins, click on the pipeline we have created.
- Click on Build Now.
- The whole pipeline should work and our CI/CD is ready.

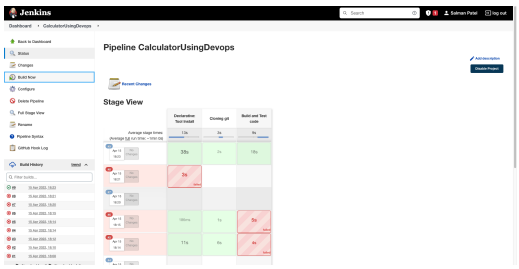


Fig. 44. jenkins intermediate progress image

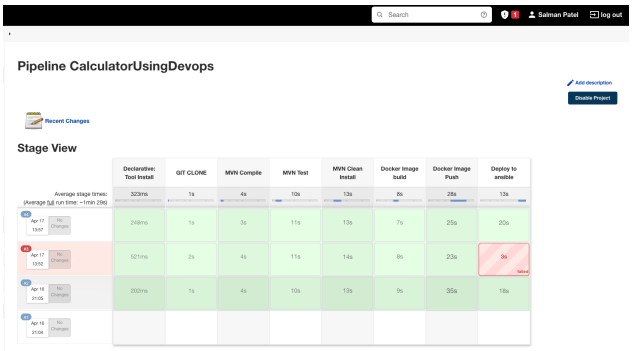


Fig. 45. Final Jenkins pipeline

'Monitoring' To automate whole process.

ELK Stack

Once the project is deployed logs are generated when it is used.

Download the log file (calculator.log) from the VM Instance.

Create a logstash configuration file in the local computer and update the absolute path to the log file we just downloaded.

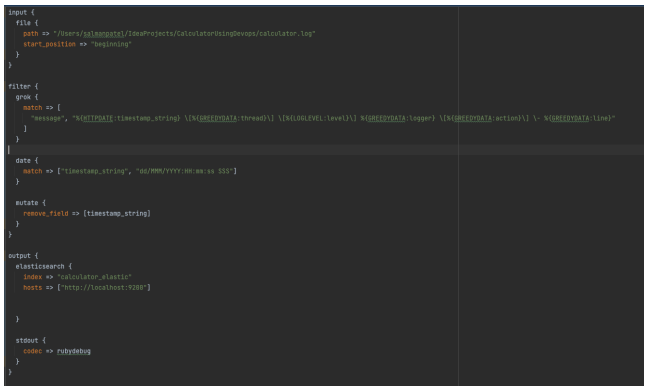


Fig. 46. log file

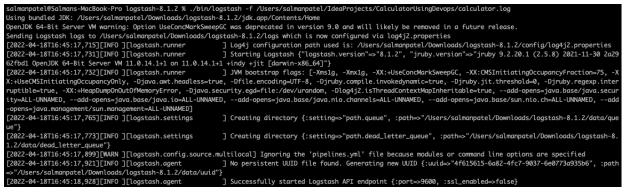


Fig. 47. File path

Now, Run ELK Stack

Go to elasticsearch directory, then $ ./bin/elasticsearch

Elasticsearch runs on port number 9200, you can verify it by visiting http://localhost:9200

Next go to kibana directory and $ ./bin/kibana

Kibana runs on port number 5601, you can verify it by visiting http://localhost:5601

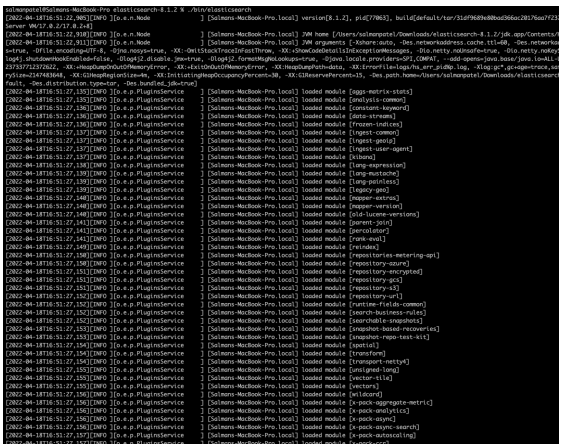


Fig. 48. Configuration

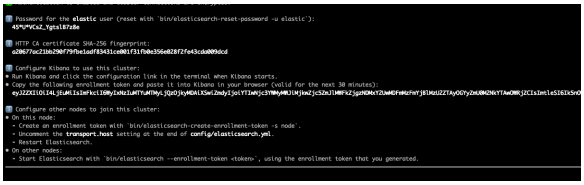


Fig. 49. Configuration2

Next go to logstash directory and $ ./bin/logstash -f /path/to/configuration/file

Open Kibana in a web browser at http://localhost:5601
Go to Management -> Stack Management

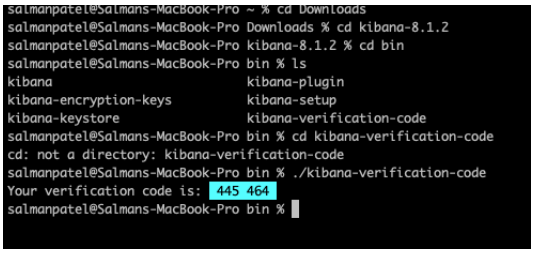


Fig. 50. Verification code path

Look for Kibana → Index Patterns → Create Index Pattern, set your index pattern based on the index pattern provided in logstash configuration file followed by *

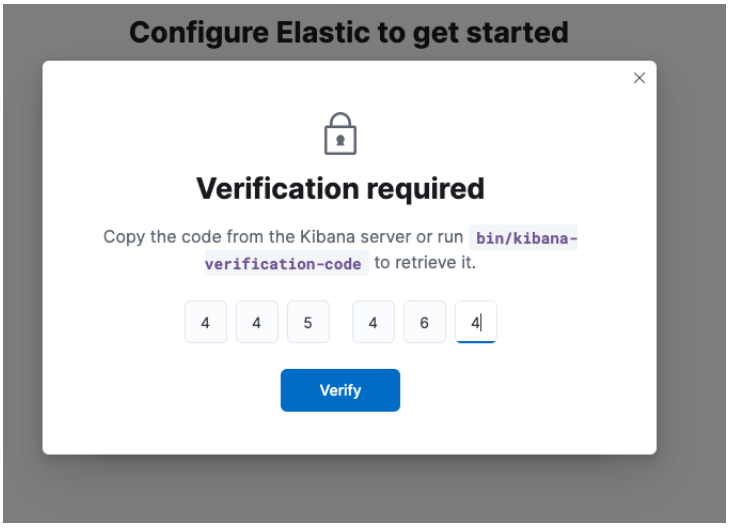


Fig. 51. Verifcation code

And in the next step select @timestamp as your Time field

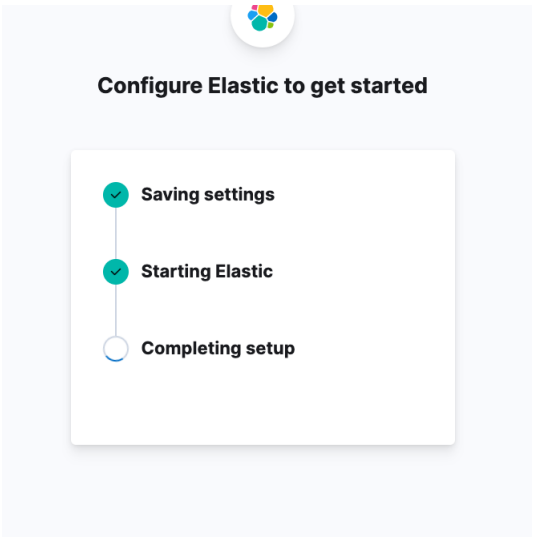


Fig. 52. Setting up

Hit Create Index Pattern, and you are ready to analyze the data.

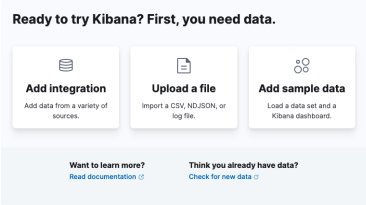


Fig. 53. input log file here

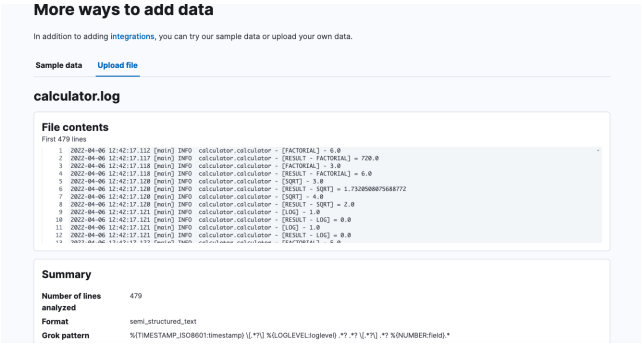


Fig. 54. Import Settings

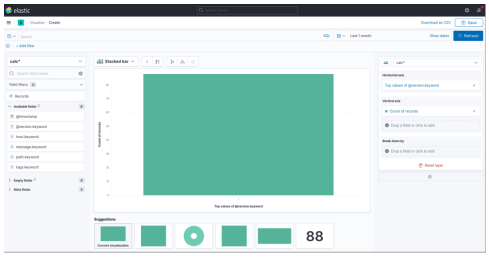


Fig. 55. Final Ouput

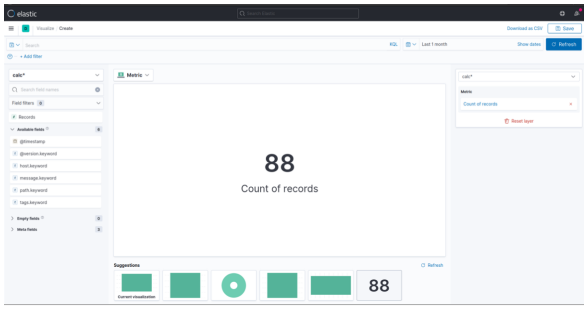


Fig. 56. Final output count

GitHub Repository:

github.com/mdsalman991/CalculatorUsingDevops

Docker Repository:

hub.docker.com/repository/docker/kimchi1503/calculator