Software Production Engineering

CI/CD Pipeline on Calculator Program

Manav Ketan Desai, MT2019060

May 9, 2020

Contents

1. Introduction 3
2. Source Code and Output 3
3. Software Development Life Cycle (SDLC) 5
3.1 Source Control Management (SCM) 6
3.2 Build and Test 6
3.3 Continuous Integration (CI) 7
3.3.1 Jenkins Installation 7
3.3.2 Jenkins Pipeline 9
3.3.3 Jenkins Pipeline Job 1: calculator-maven-build-pipeline 10
3.4 Continuous Delivery 12
3.4.1 Docker Installation 12
3.4.2 Building and Publishing Images 13
3.4.3 Jenkins Pipeline Job 2: calculator-build-publish-image 13
3.5 Continuous Deployment 16
3.5.1 Rundeck Installation 16
3.5.2 Creating Rundeck Project and Job 17
3.5.3 Jenkins Pipeline Job 3: calculator-deploy-image 20
3.6 Monitoring 23
3.6.1 Installing the Elastic Stack 23
3.6.2 Configuring Filebeat 24
3.6.3 Visualizing Logs in Kibana 27
4. Conclusion 30
5. References 30

1. Introduction

In this report, I aim to automate the development, testing and deployment of a calculator program by creating a CI/CD pipeline on it. The calculator program developed here is a Spring Boot project hosted on a Tomcat web server running in a Docker container. The calculator currently supports the four basic mathematical operations and can be accessed as a service. The service can be called via GET request either through curl or through a web browser.

The CI/CD pipeline is built using the following tools:

1. Development: IntelliJ IDEA, Git, Github
2. Testing: JUnit
3. Integration: Apache Maven, Jenkins
4. Delivery: Docker, Docker hub, Jenkins
5. Deployment: Docker, Rundeck, Jenkins
6. Monitoring: Elastic Stack

The entire source code can be found here: https://github.com/mkd1997/devops-on-calculator

2. Source Code and Output

The source code is divided into 2 layers – Controller and Model. The Controller layer has the CalculatorController class that is responsible for accepting the requests from users and calling the necessary operation function in the Model layer depending on the request parameters.

```
package com.mkd.devopsdemo.Controllers;

import com.mkd.devopsdemo.Models.CalculatorModel;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@Controller
public class CalculatorController {

    @GetMapping("/calculate/{leftOperand}/{rightOperand}/{operator}")
    public ResponseEntity<String> index(@PathVariable Integer leftOperand, @PathVariable String operator, @PathVariable Integer
rightOperand) {

        CalculatorModel calculator = new CalculatorModel();
        double res = 0;
        switch (operator) {
            case "+": res = calculator.add(leftOperand, rightOperand);
                break;
            case "-": res = calculator.subtract(leftOperand, rightOperand);
                break;
            case "*": res = calculator.multiply(leftOperand, rightOperand);
                break;
            case "/": res = calculator.divide(leftOperand, rightOperand);
                break;
        }
        return new ResponseEntity<>( "Your result is " + res, HttpStatus.OK);
    }
}
```

The CalculatorModel class implements the four basic functions of the calculator program.

```java
package com.mkd.devopsdemo.Models;

/**
 * Caculator
 */
public class CalculatorModel {
    public int add(int x, int y) {
        return (x + y);
    }

    public int subtract(int x, int y) {
        return (x - y);
    }

    public int multiply(int x, int y) {
        return (x * y);
    }

    public double divide(int x, int y) {
        double quotient;
        try {
            quotient = x / y;
        } catch (Exception e) {
            quotient = Double.MAX_VALUE;
        }
        return quotient;
    }

    public int modulus(int x, int y) {
        return (x % y);
    }
}
```

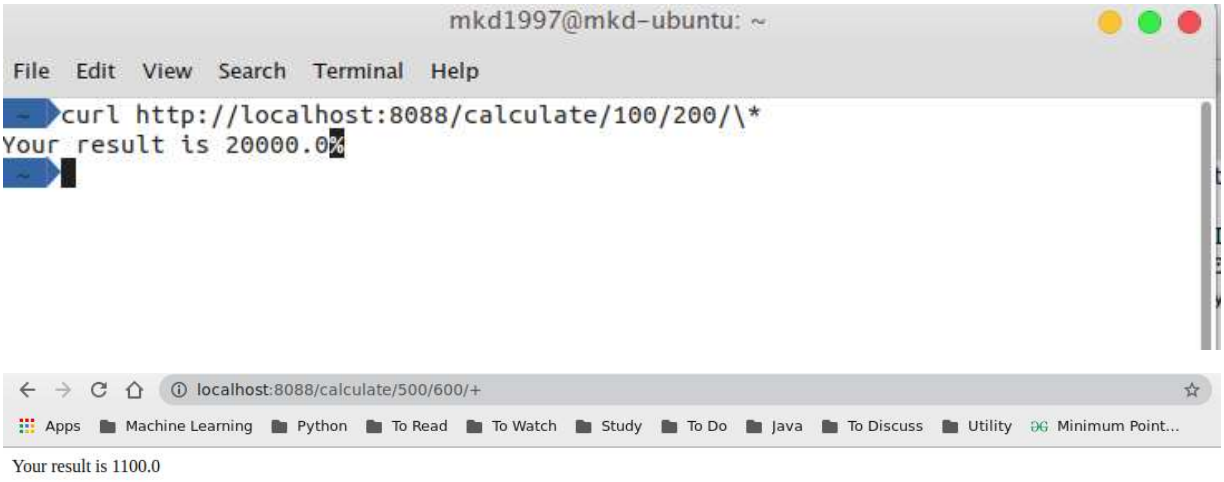
The Spring Boot application is started by running the Main class.

```java
package com.mkd.devopsdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CalculatorDriver {
    public static void main(String[] args) {
        SpringApplication.run(CalculatorDriver.class, args);
    }
}
```

I am here using the Embedded Tomcat server that is present in all jar packages created using Spring Boot framework. Here, I configure the Embedded Tomcat server to run on port 8088. Since I will be monitoring Tomcat access logs using Elastic Search later, I here configure Tomcat access logs to be generated in /val/log/tomcat directory.

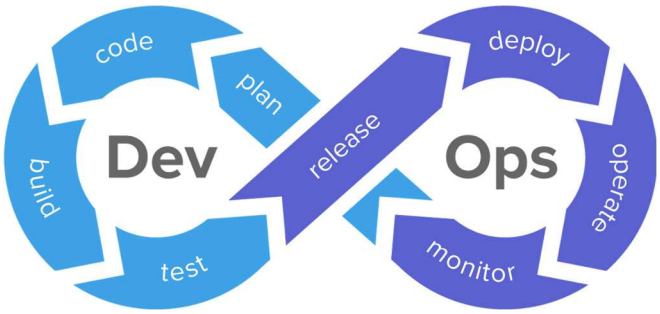
```
server.port=8088
server.tomcat.accesslog.enabled=true
server.tomcat.accesslog.directory=/var/log/tomcat
server.tomcat.accesslog.suffix=.log
server.tomcat.accesslog.prefix=access_log
server.tomcat.accesslog.file-date-format=.yyyy-MM-dd
```

The output of the program, as seen using curl and Google Chrome web browser is shown below.



3. Software Development Life Cycle (SDLC)

The various stages of Software Development Life Cycle (SDLC) are automated using various DevOps tools. The stages, and how are the automated, is described below:



3.1 Source Control Management (SCM)

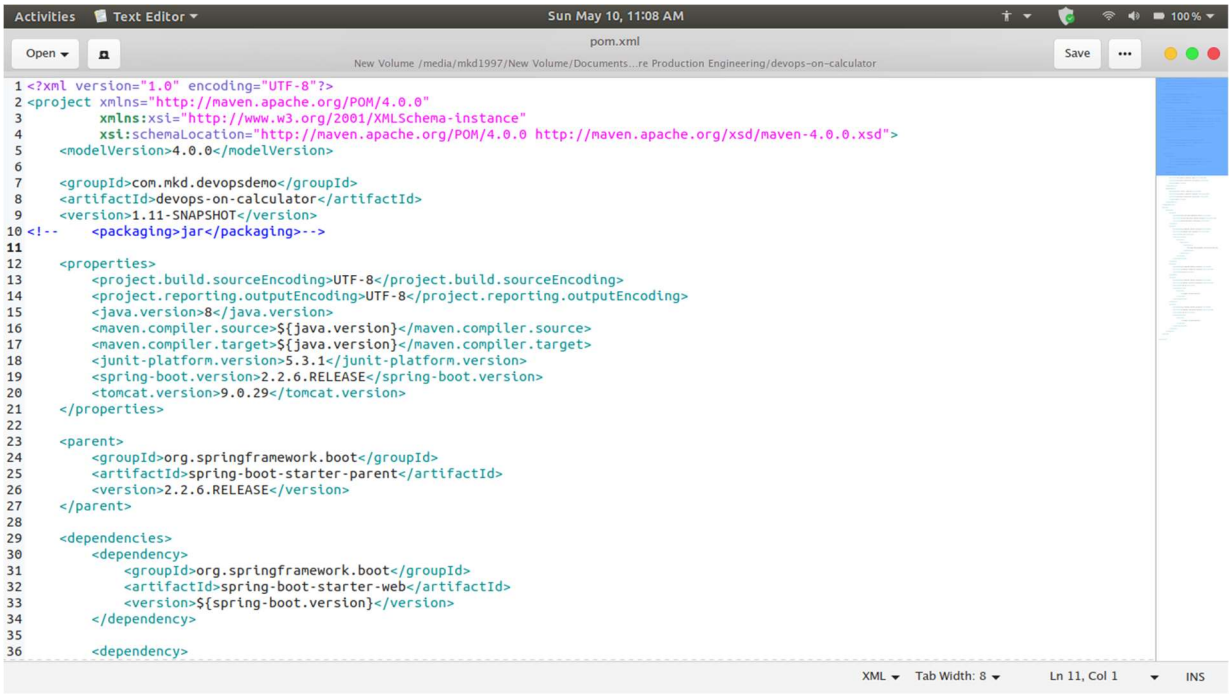
Source control refers to the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources. I use Git as the SCM here. Github is an online repository hosting service.

The development of this program has been done incrementally. Some of the important increments done are listed below. Rest can be seen at https://github.com/mkd1997/devops-on-calculator/commits/master:

1. Add addition and subtraction operations.
2. Add multiplication and division operations.
3. Convert to Spring Boot application.
4. Enable tomcat access logging.

3.2 Build and Test

In the calculator program, Apache Maven is responsible for managing dependencies and building the project. It is Maven who finally outputs the SNAPSHOT jar of the project that has the compiled classes along with other classes the project depends on. The pom.xml file of the project is shown here.




```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>com.mkd.devopsdemo</groupId>
    <artifactId>devops-on-calculator</artifactId>
    <version>1.11-SNAPSHOT</version>
<!--    <packaging>jar</packaging>-->

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>8</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>
        <junit-platform.version>5.3.1</junit-platform.version>
        <spring-boot.version>2.2.6.RELEASE</spring-boot.version>
        <tomcat.version>9.0.29</tomcat.version>
    </properties>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.6.RELEASE</version>
    </parent>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>${spring-boot.version}</version>
        </dependency>

        <dependency>
```

JUnits is a unit testing framework that is used to write unit test cases. The maven-surefire-plugin runs the test classes at maven build time. The test class for the calculator project is shown here.

```
package com.mkd.devopsdemo;

import com.mkd.devopsdemo.Models.CalculatorModel;

import static org.junit.jupiter.api.Assertions.*;

class CalculatorModelTest {

    @org.junit.jupiter.api.Test
    void addWithInverseShouldReturnZero() {
        CalculatorModel calcTest = new CalculatorModel();
        assertEquals(0, calcTest.add(-5, 5), "-5 + 5 should be 0");
    }

    @org.junit.jupiter.api.Test
    void subtractionOfEqualNumbersShouldBeZero() {
        CalculatorModel calcTest = new CalculatorModel();
        assertEquals(0, calcTest.subtract(5, 5), "5 - 5 should be 0");
    }
}
```

To build the project after running the test cases use:

> mvn clean package –U

To build the project without running the tests, use:

> mvn clean package –U -DskipTests

3.3 Continuous Integration (CI)

Continuous Integration (CI) refers to the practice of integrating code changes with the existing code as and when it is written. This includes building the project and running the test cases too automatically as described above. Jenkins is the tool that I have used for Continuous Integration. It keeps watching the SCM system for changes in the code and builds it as and when it detects the changes. Apache Maven is integrated with Jenkins so that Jenkins can trigger maven builds.

3.3.1 Jenkins Installation

To install Jenkins, follow the steps given below:

1. Download and install the necessary GPG key

   > wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

2. Add the necessary repository

   > sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

3. Add the universe repository

   > sudo add-apt-repository universe

4. Update apt

   > sudo apt update

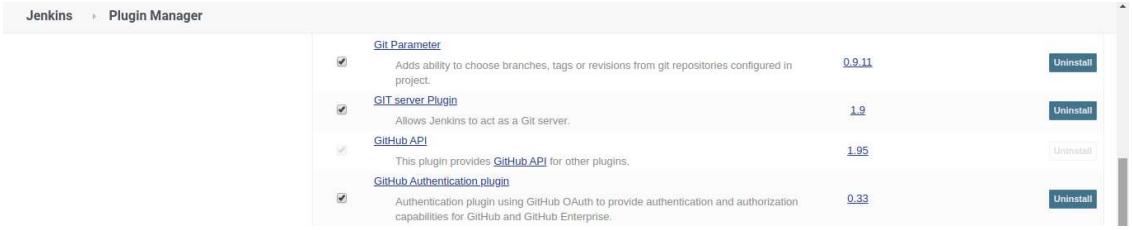
5. Install Jenkins

   > sudo apt-get install jenkins –y

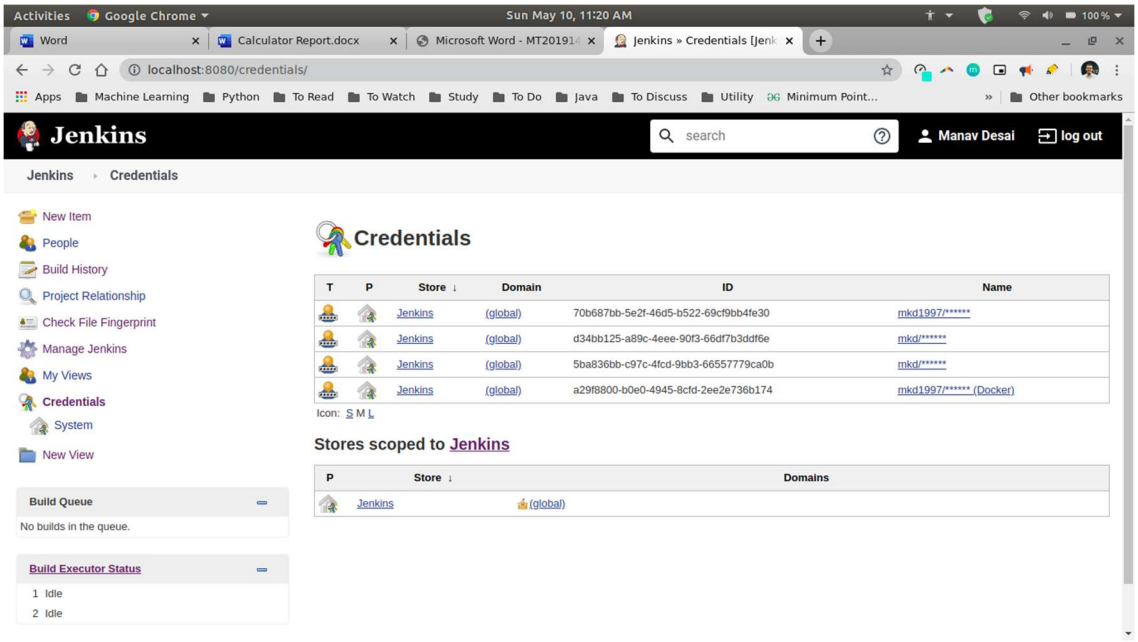
6. Start Jenkins

   > sudo systemctl start jenkins

7. Install git, Maven, JUnit plugin by going to Manage Jenkins -> Manage Plugins



8. Add git credentials to Jenkins by going to Credentials.

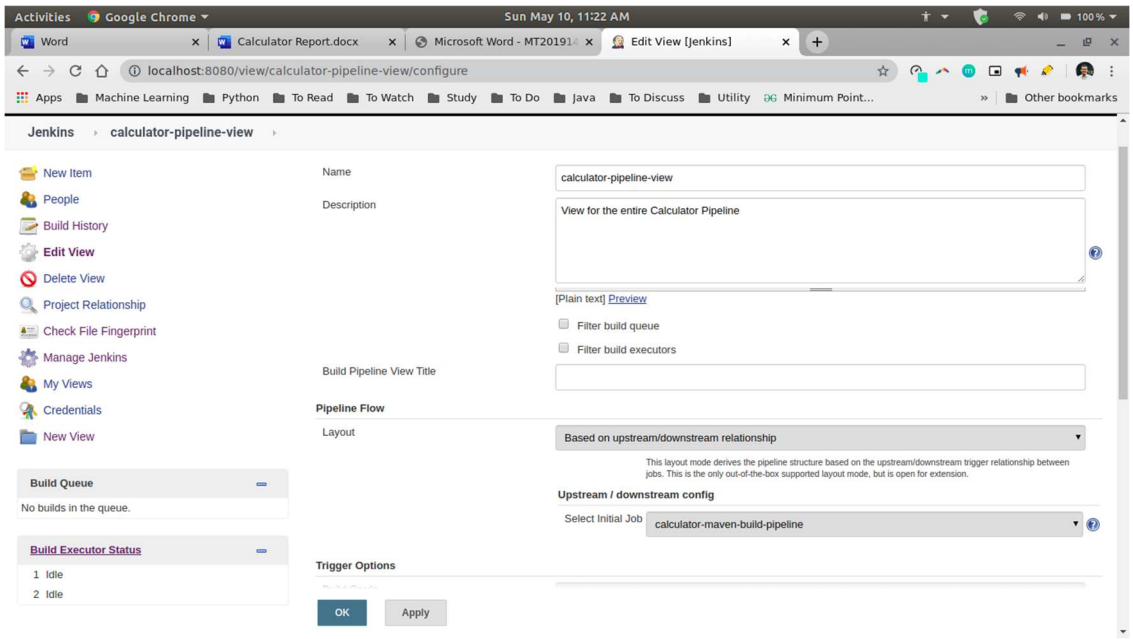


3.3.2 Jenkins Pipeline

A Jenkins pipeline gives us a graphical view of the various steps of a CI/CD pipeline. It allows us to link different Jenkins jobs and allows us to check their progress during execution.

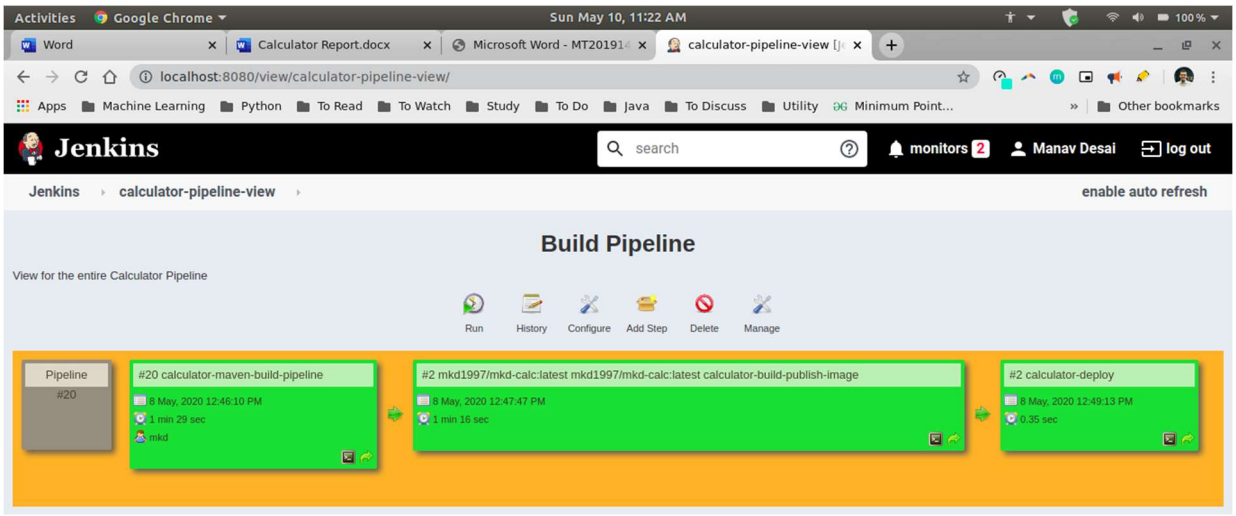
To create Jenkins pipeline, follow the given steps:

1. Click on the '+' sign above the list of Jenkins projects on the main page of Jenkins.
2. Provide a pipeline view name, an optional description and the initial job.



3. Click OK.

The final pipeline is shown below.

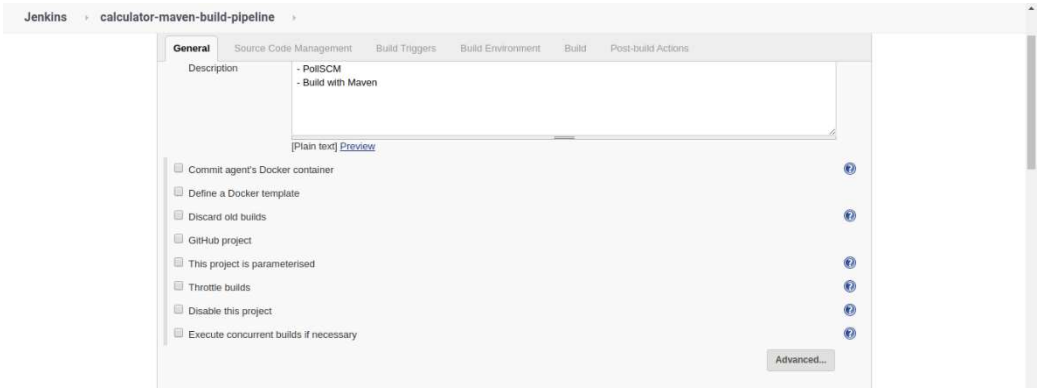


3.3.3 Jenkins Pipeline Job 1: calculator-maven-build-pipeline

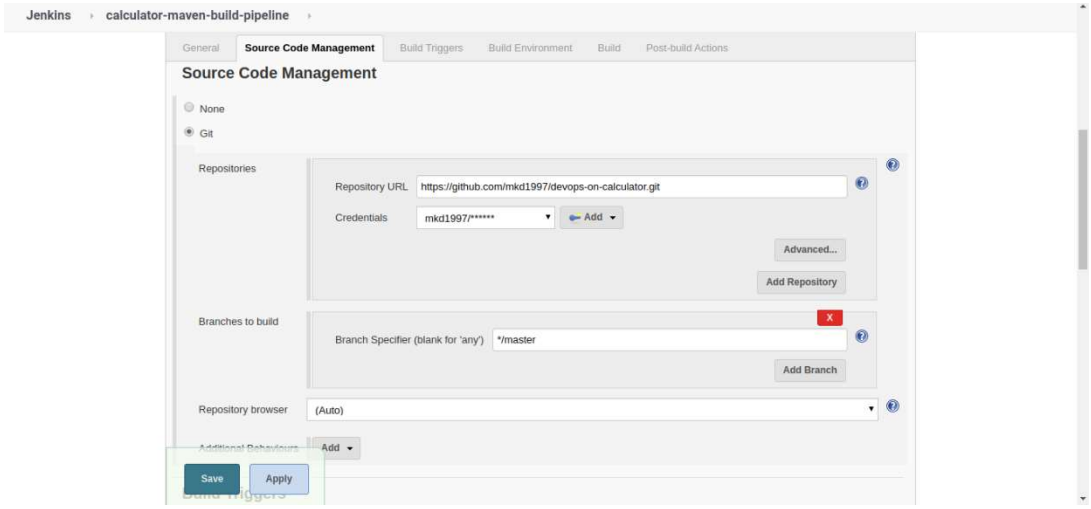
This Jenkins project is responsible for polling the Github repository every 15 minutes and checking for code changes. As soon as it finds that new code has been pushed, it pulls the code into Jenkins workspace and starts a maven build with the specified goal. The goal specified here is clean package –U which also runs the JUnit tests.

To create this project, follow the given steps:

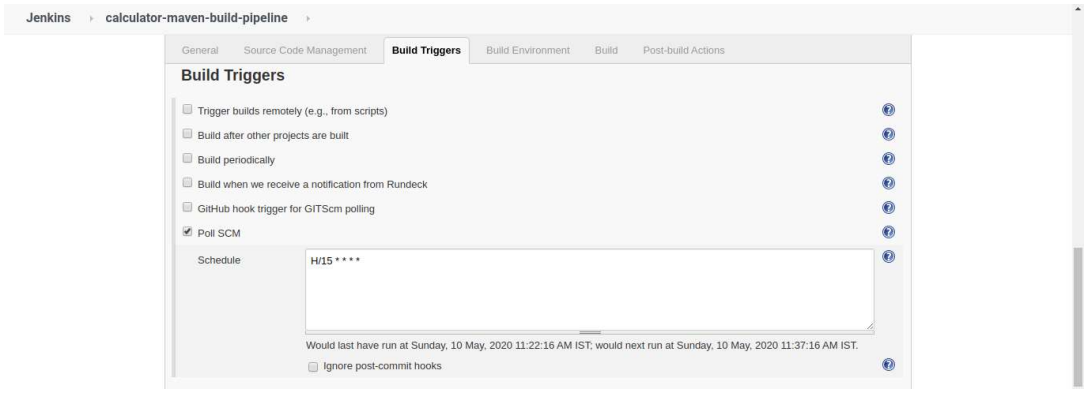
1. Create a new freestyle project and do the following configuration.
    a. General configuration



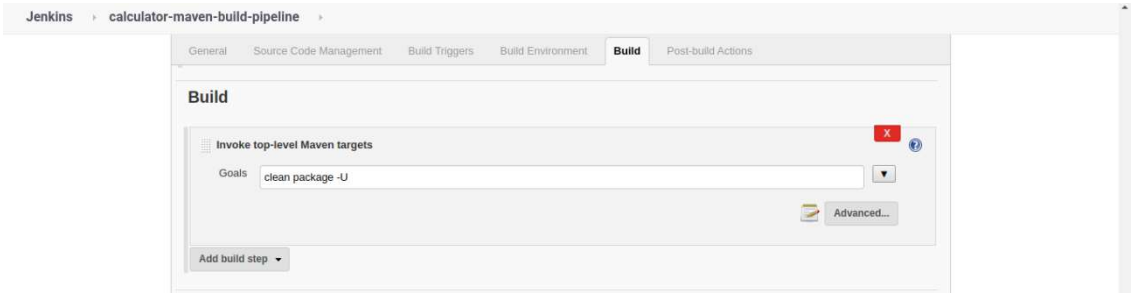
    b. Source Code Management



c. Build Triggers

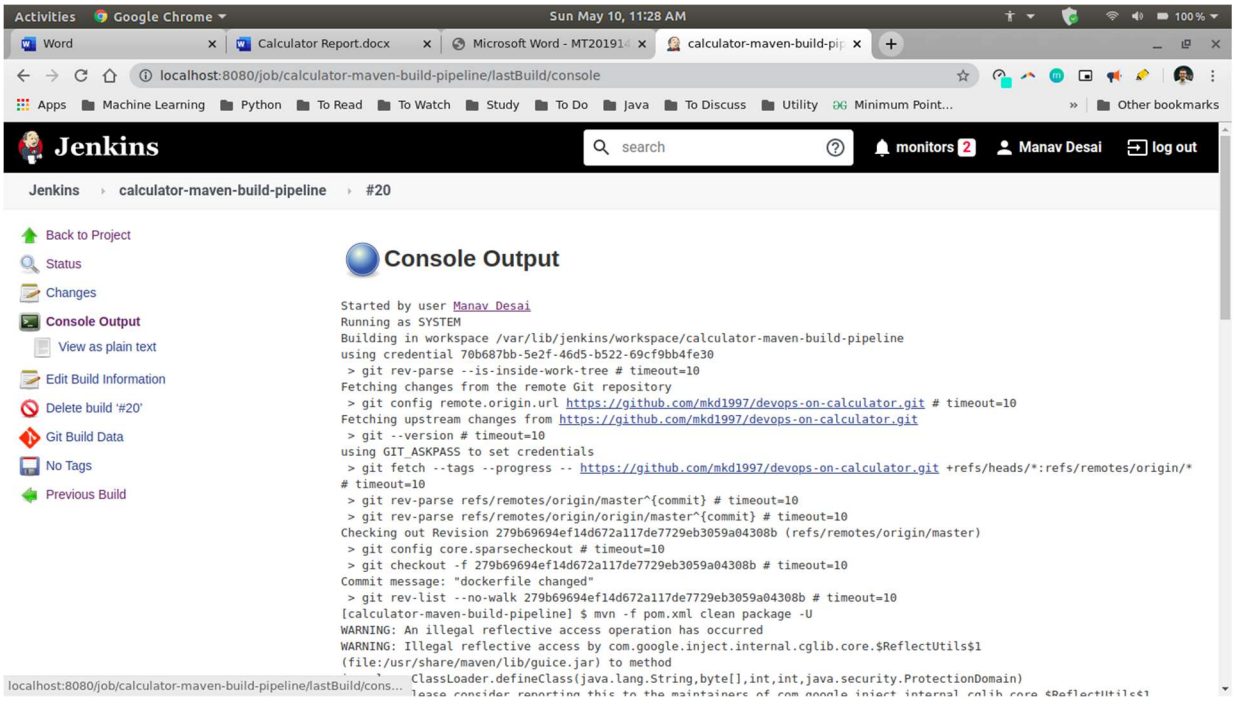


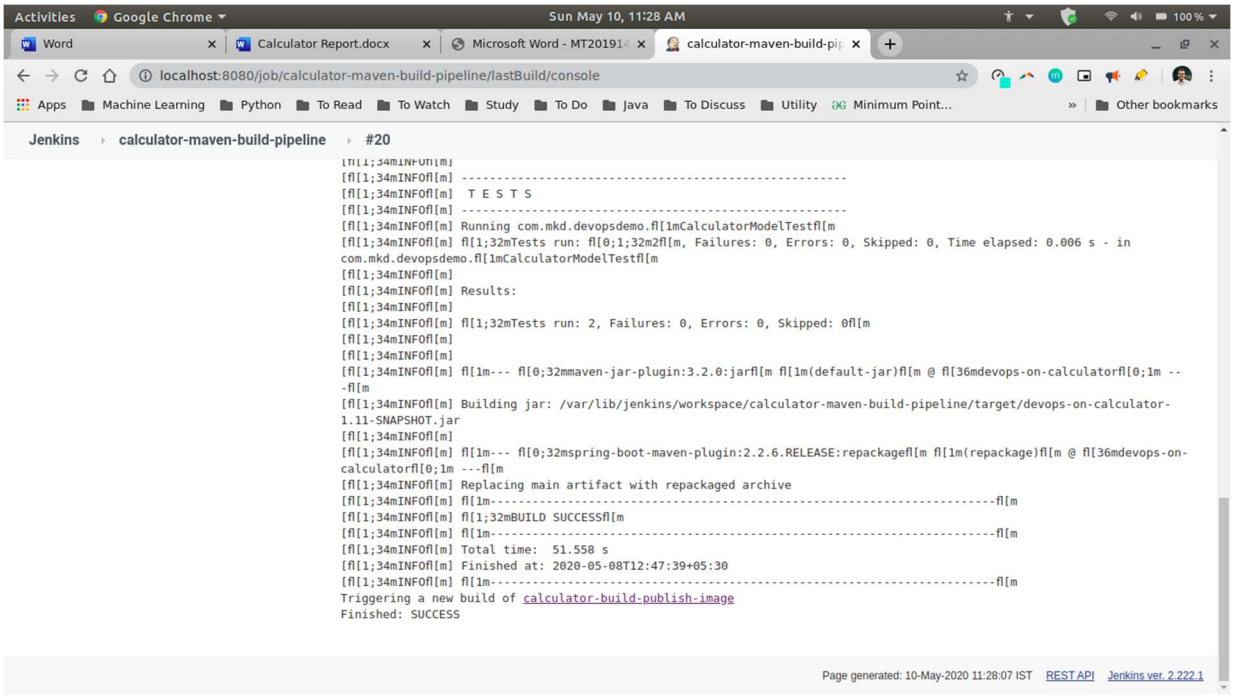
d. Build



2. Click Save.

Console output of the calculator-maven-build-pipeline is shown below





3.4 Continuous Delivery

A deliverable in Software Engineering is an artifact that is ready to be delivered to the client. It thus is the end product of a SDLC. Continuous Delivery (CD) is the practice of generating deliverable as soon as code changes happen. CD requires that CI pipeline be in place first. Hence, CI is a prerequisite of CD.

Here, the deliverable is in the form of a docker image. The image consists of everything that our project requires to run including OS, OpenJDK and Tomcat server.

The tools used by me for creating a CD pipeline are Docker and Jenkins.

3.4.1 Docker Installation

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and deploy it as one package.

To install Docker, follow the given steps:

1. Type the following commands on the terminal

```
> sudo apt-get update
> sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
> curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

> sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
> sudo apt-get update
> sudo apt-get install docker-ce docker-ce-cli containerd.io

2. Check whether Docker has been installed or not

   > sudo docker run hello-world

3. Configure Docker to run without sudo and also give Jenkins permission to run docker commands without sudo

   > sudo groupadd docker
   > sudo usermod -aG docker mkd1997
   > sudo usermod –aG docker jenkins

3.4.2 Building and Publishing Images

Maven build outputs a jar file consisting of the compiled classes, their dependencies and also the embedded Tomcat server. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

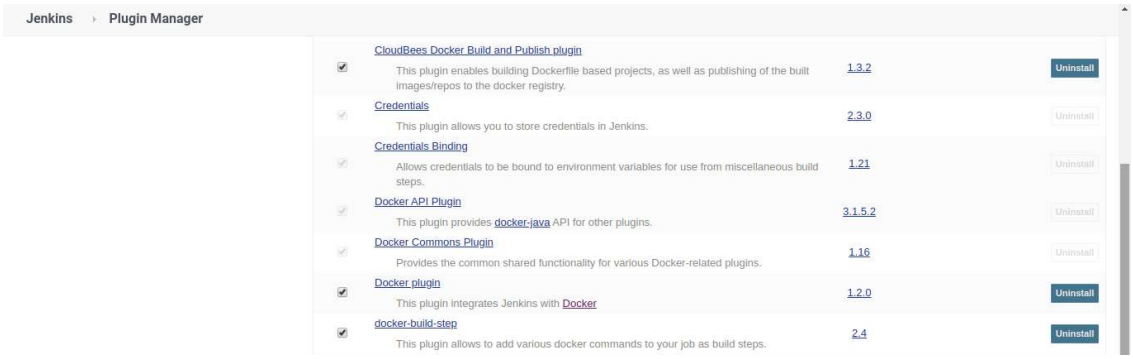
So, I first created a Dockerfile. The Dockerfile is placed in the root directory of the project. It is shown below.

```
FROM openjdk:8
ADD target/devops-on-calculator-1.11-SNAPSHOT.jar devops-on-calculator-1.11-SNAPSHOT.jar
RUN mkdir -p -m 777 /var/log/tomcat
EXPOSE 8088
ENTRYPOINT ["java", "-jar", "devops-on-calculator-1.11-SNAPSHOT.jar"]
```

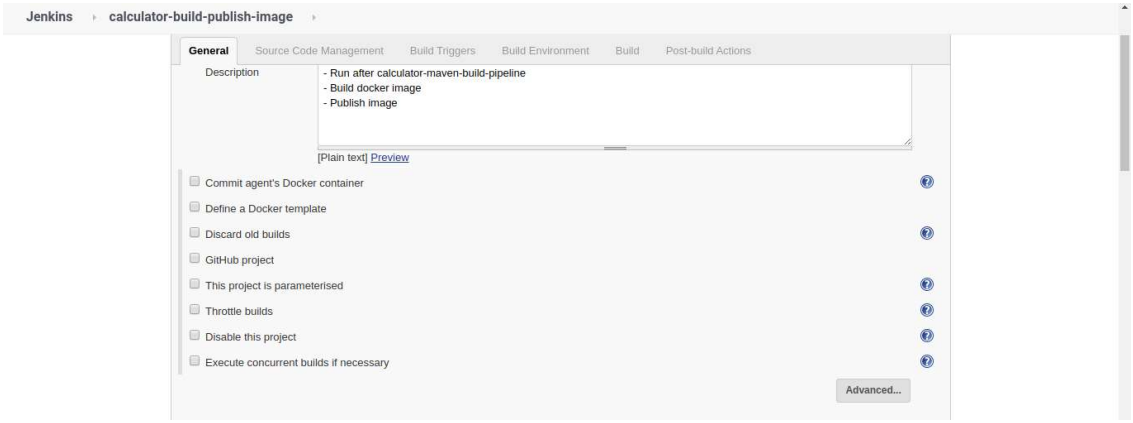
3.4.3 Jenkins Pipeline Job 2: calculator-build-publish-image

I now use this Jenkins project to build the Docker image and publish it to Dockerhub. To do this, follow the given steps:

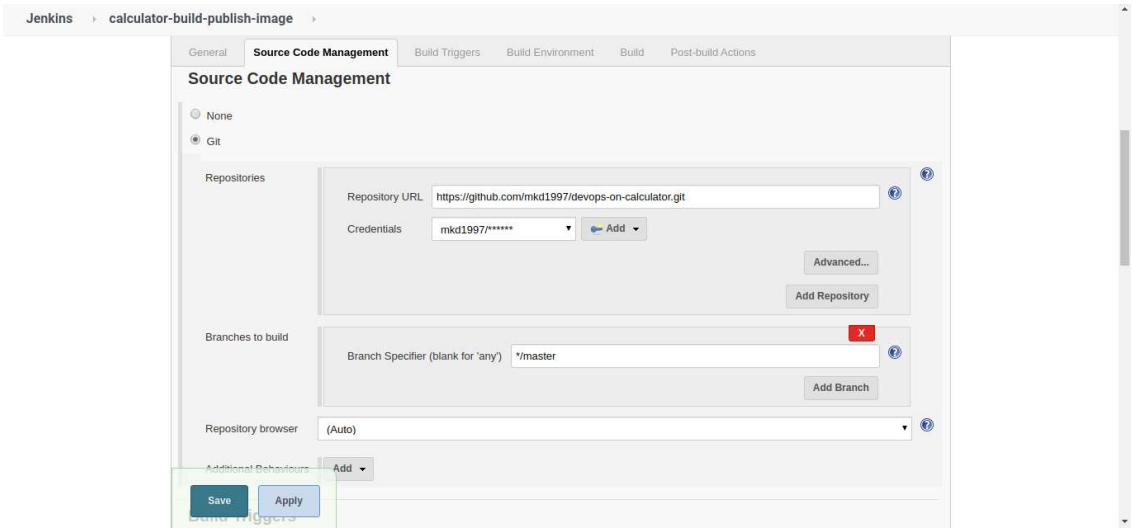
1. On the Jenkins main dashboard, go to Manage Jenkins -> Manage Plugins and Docker plugins.



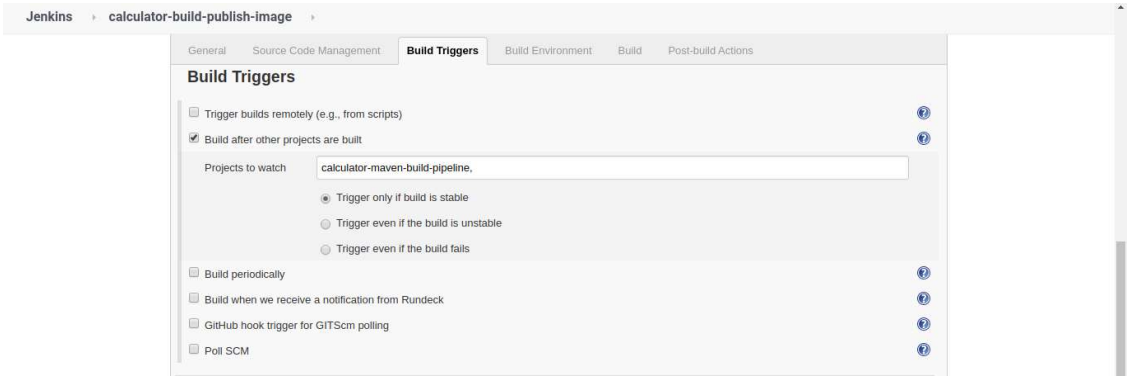
2. Create a new freestyle project and do the following configuration.
    a. General



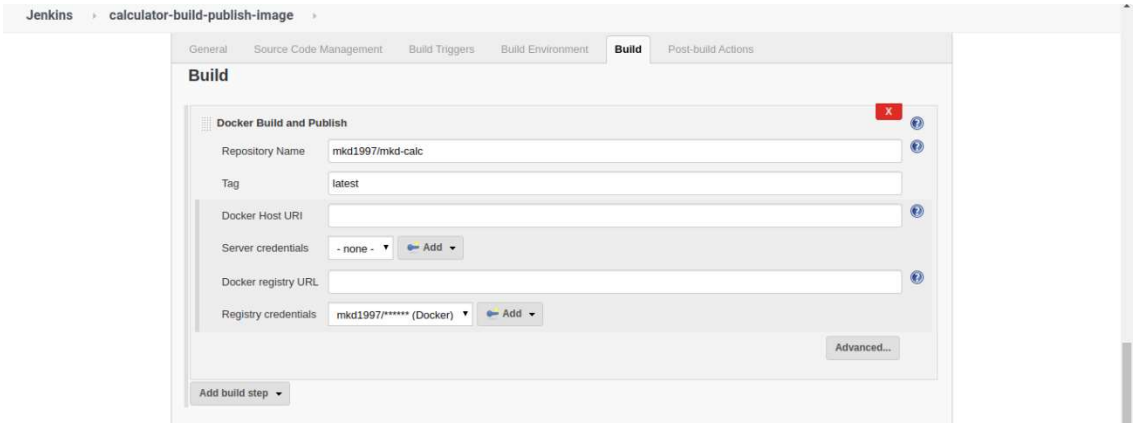
    b. Source Code Management



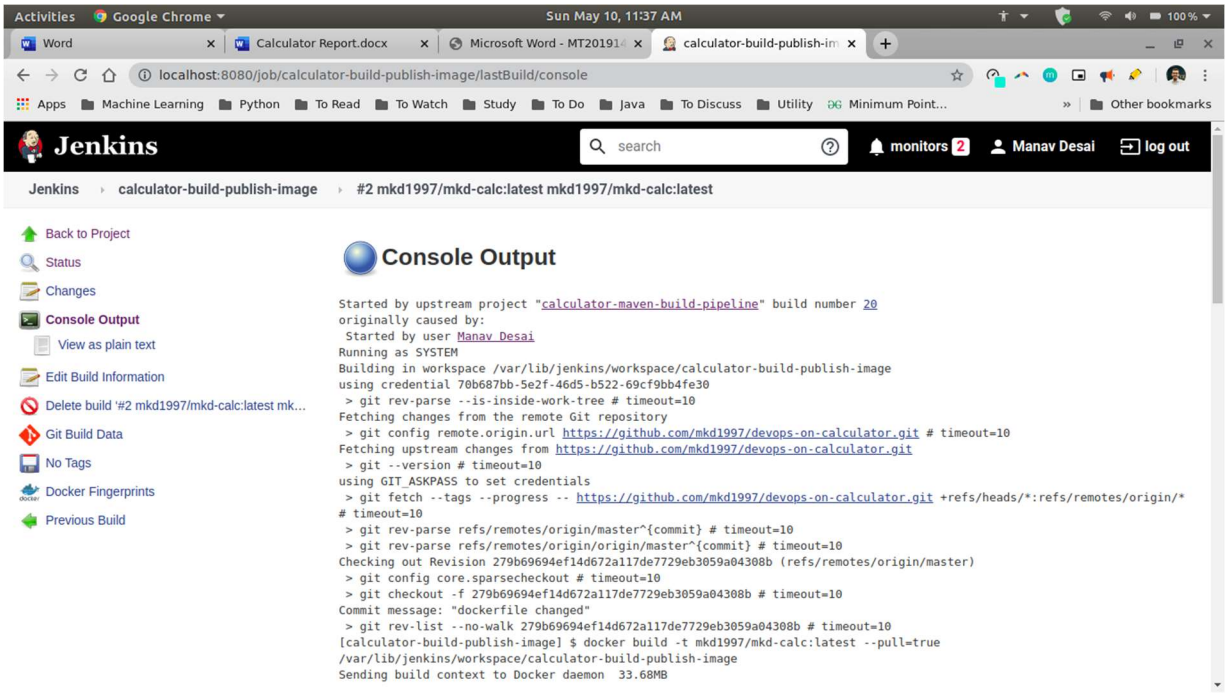
    c. Build Trigger

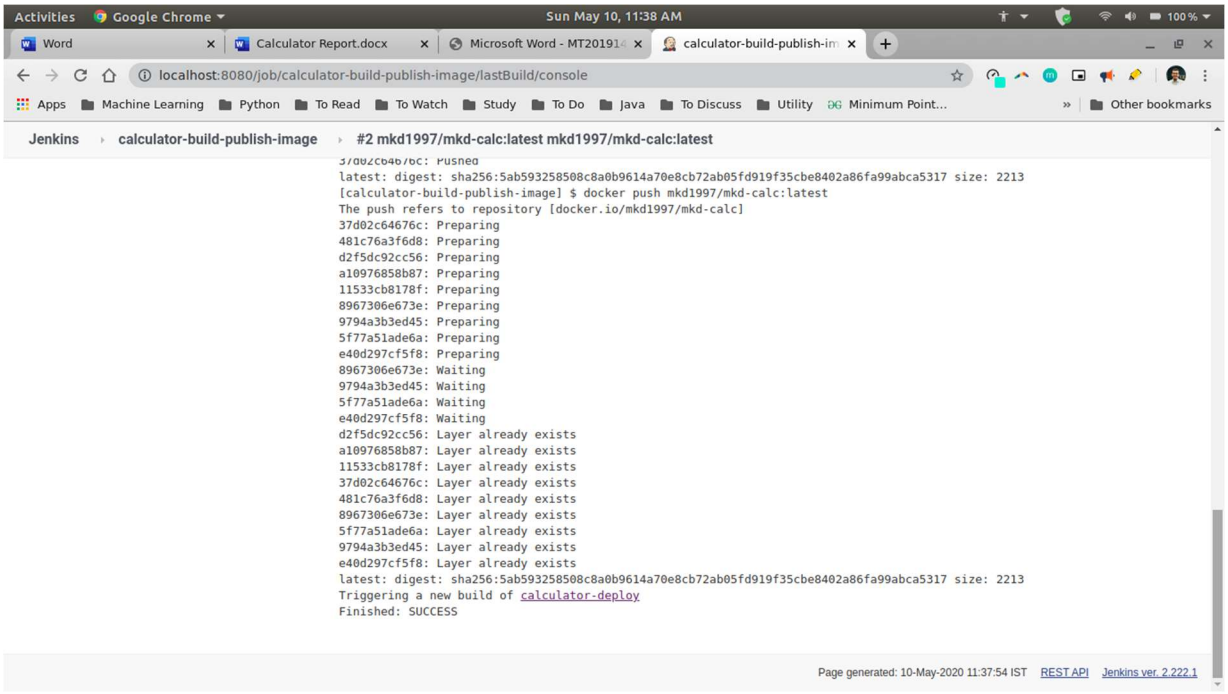


d. Build



Console output of the calculator-build-publish-image is shown below





3.5 Continuous Deployment

Continuous deployment is a strategy for software releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.

After the deliverable (image in my case) is created and published to dockerhub, I use Rundeck to fetch the image and deploy it in a container. The Rundeck job is invoked by Jenkins.

3.5.1 Rundeck Installation

Rundeck is an automation tool that executes Rundeck jobs on Rundeck nodes. Rundeck jobs can be thought of a sequence of instructions and Rundeck nodes could be anything like a web server, container, etc.

To install Rundeck, follow the given steps:

1. Rundeck requires that you have Java 8 on your system. So, first check the Java version on your system.

   > java –version

2. If you already have Java 8 installed, skip this step. Else, install Java 8 and configure your system to use Java 8 by default.

   > sudo apt-get update
   > sudo apt-get install openjdk-8-jdk
   > sudo update-alternatives --config java # then select Java 8

3. Download deb package from http://rundeck.org/download/deb/ and run the command

   > sudo dpkg -i rundeck_3.2.5.20200403-1_all.deb

4. Start Rundeck

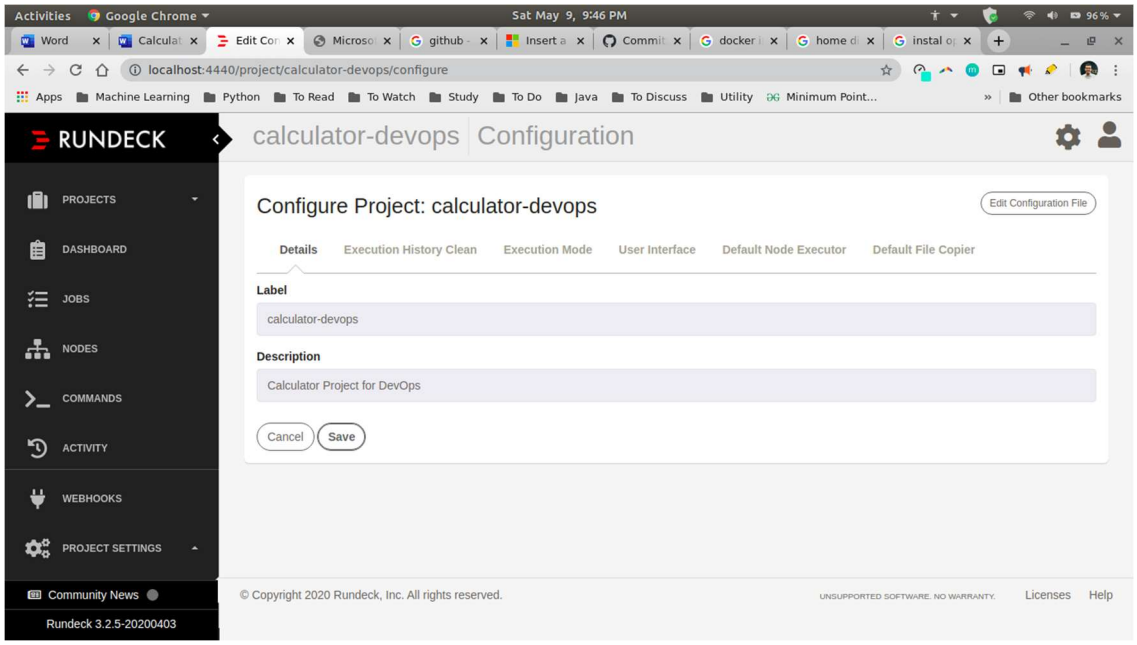
   > sudo service rundeckd start

5. Rundeck runs at http://localhost:4440 by default and default username and password are admin and admin respectively.

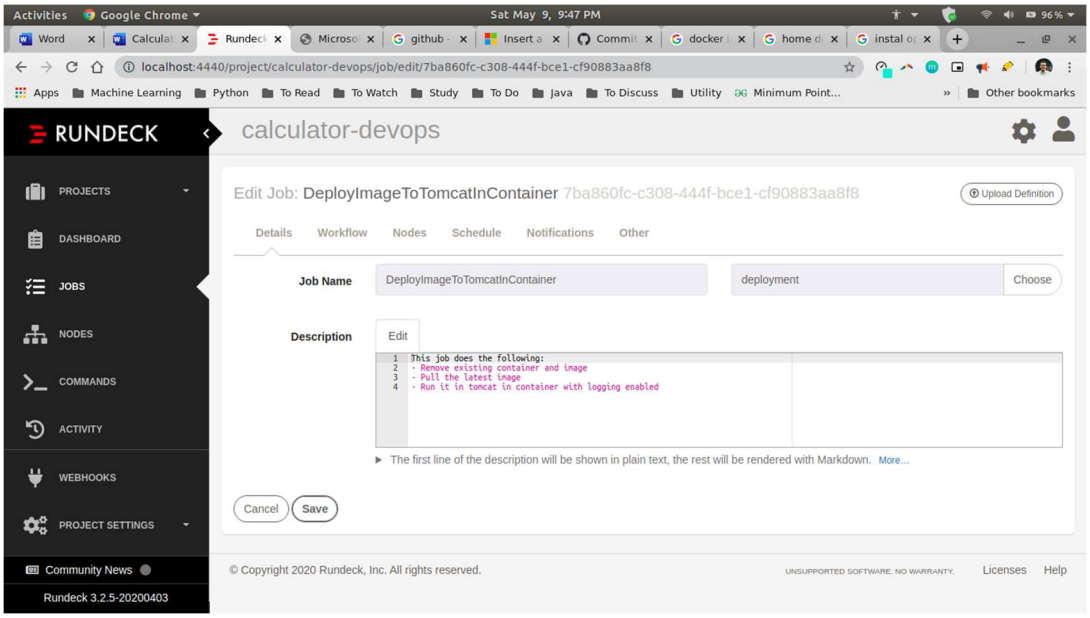
3.5.2 Creating Rundeck Project and Job

I now create a Rundeck project and Rundeck job that deploys the generated Docker image to a new container. To do so, follow the given steps:

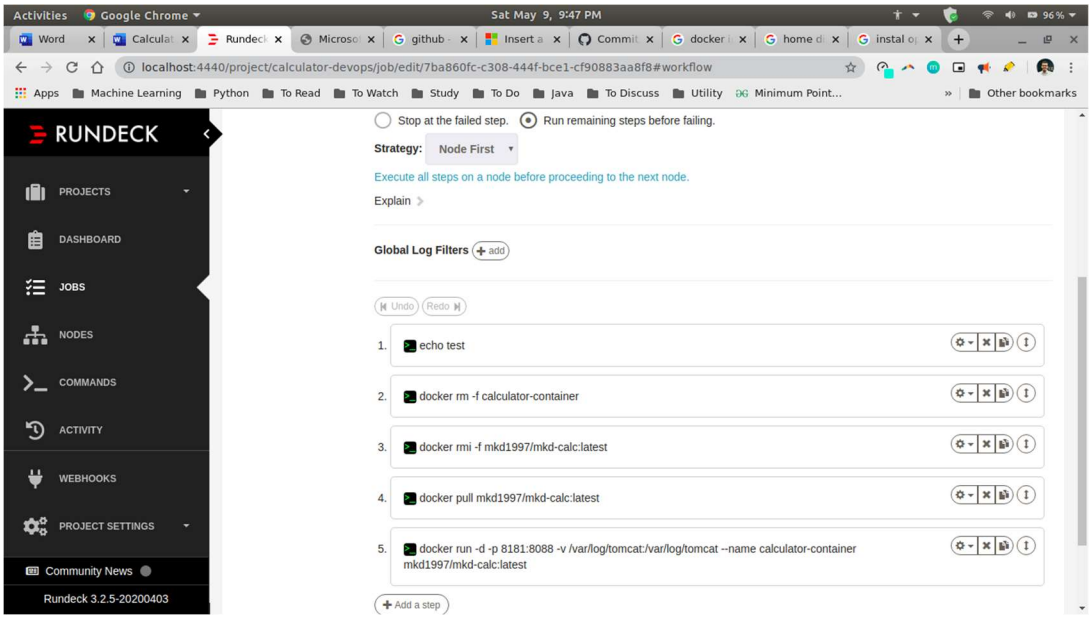
1. Create a new Rundeck project.



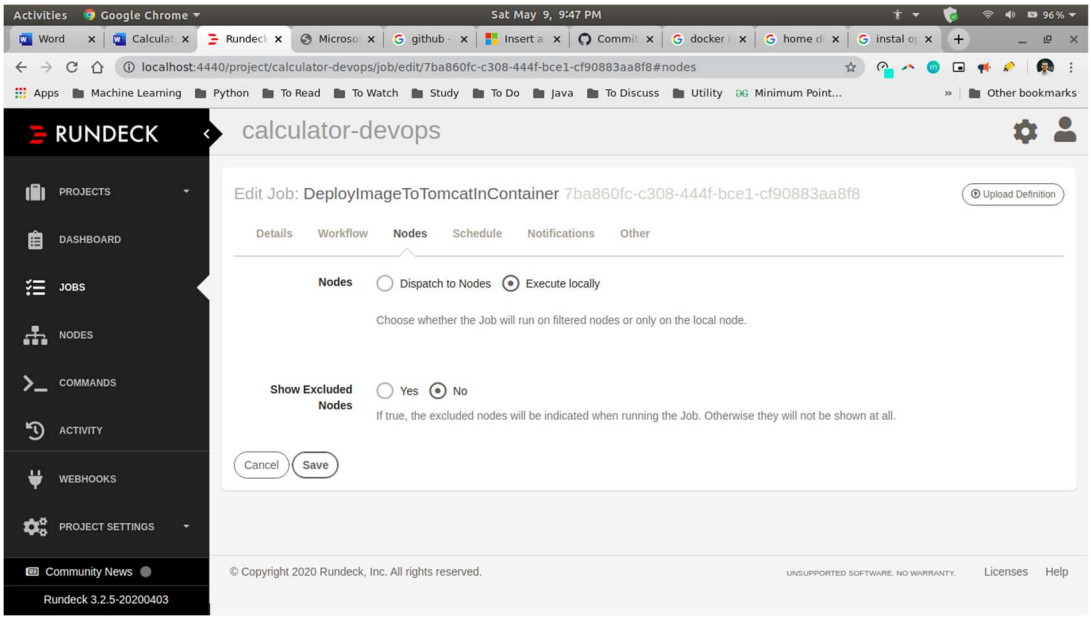
2. Create a new Rundeck job in the project and do the following configurations:
    a. Details



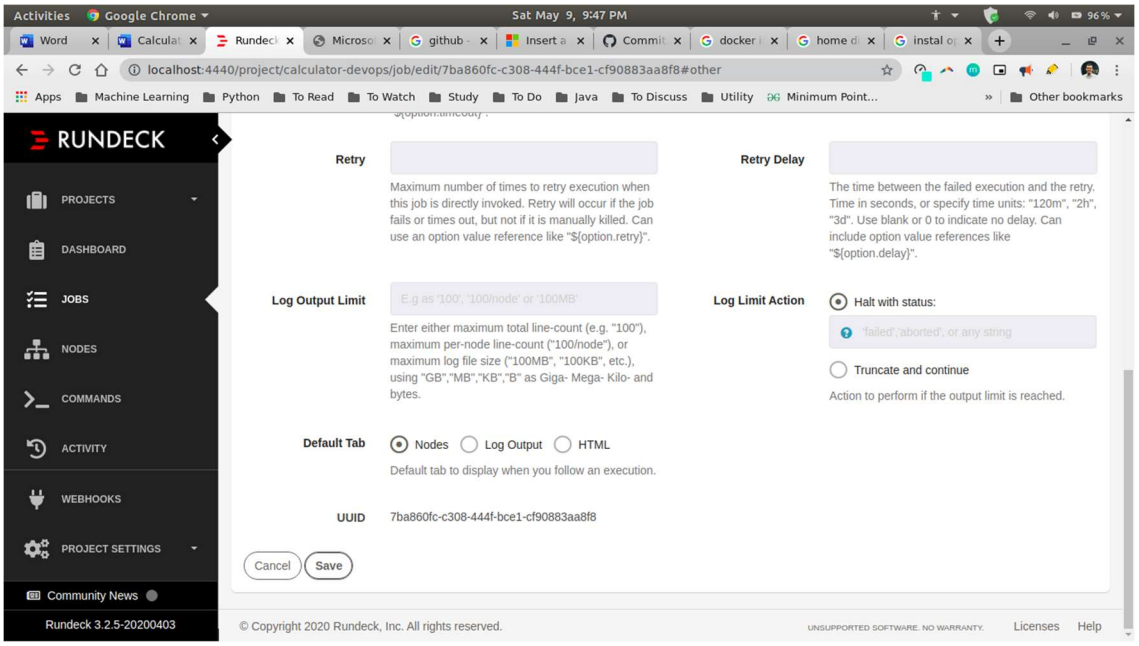
    b. Workflow



c. Nodes



3. Note the UUID of the job in the Other tab.

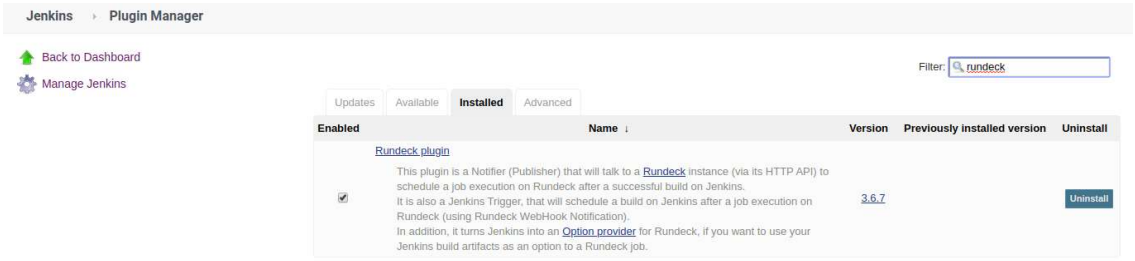


3.5.3 Jenkins Pipeline Job 3: calculator-deploy-image

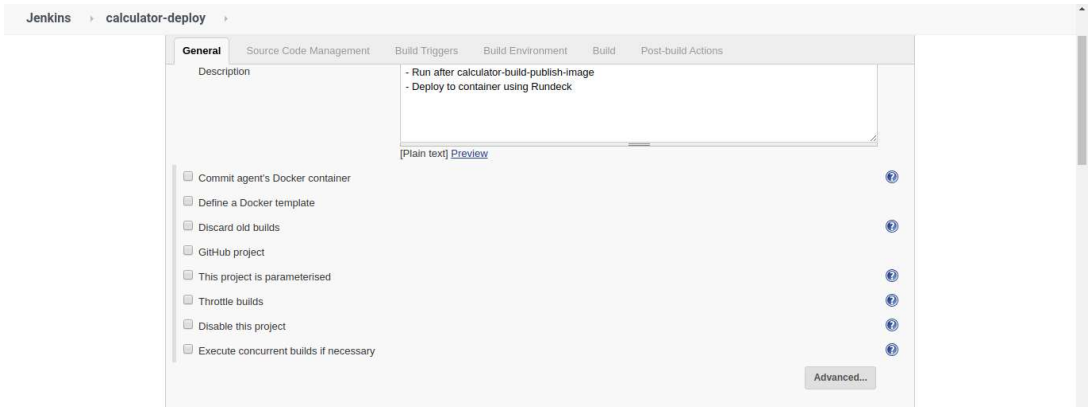
I use this Jenkins job to invoke the Rundeck job that deploys the image to a container.

To do this, follow the given steps:

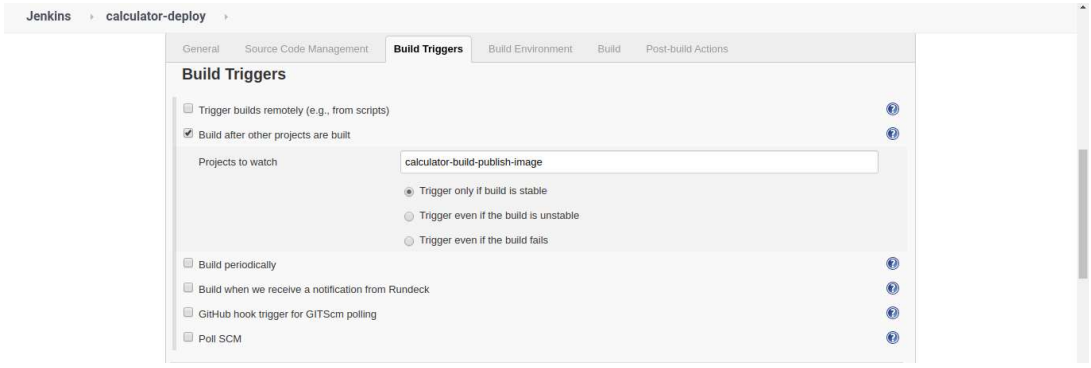
1. On the Jenkins main dashboard, go to Manage Jenkins -> Manage Plugins and install Rundeck plugin.



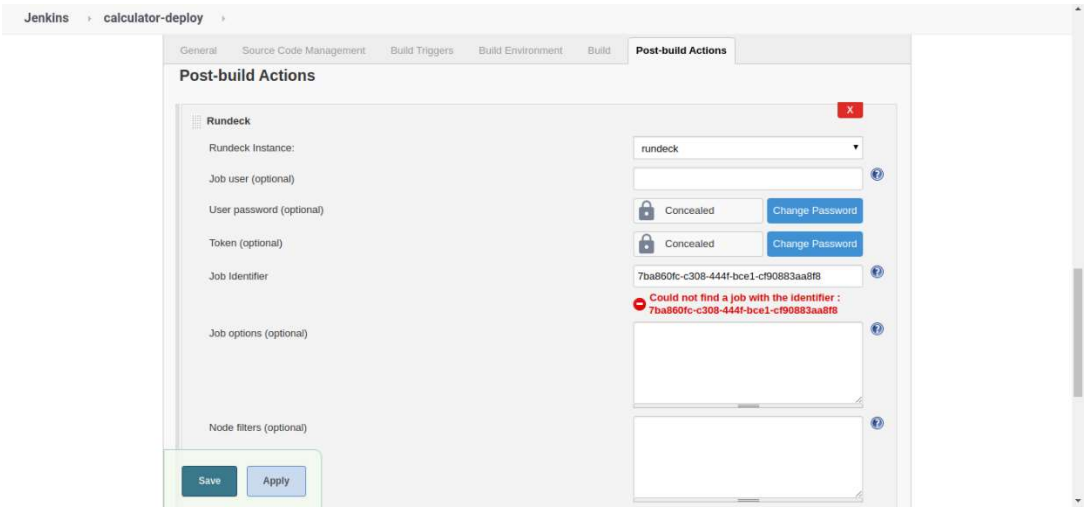
2. Create a new freestyle project and do the following configuration.
    a. General



    b. Build Trigger

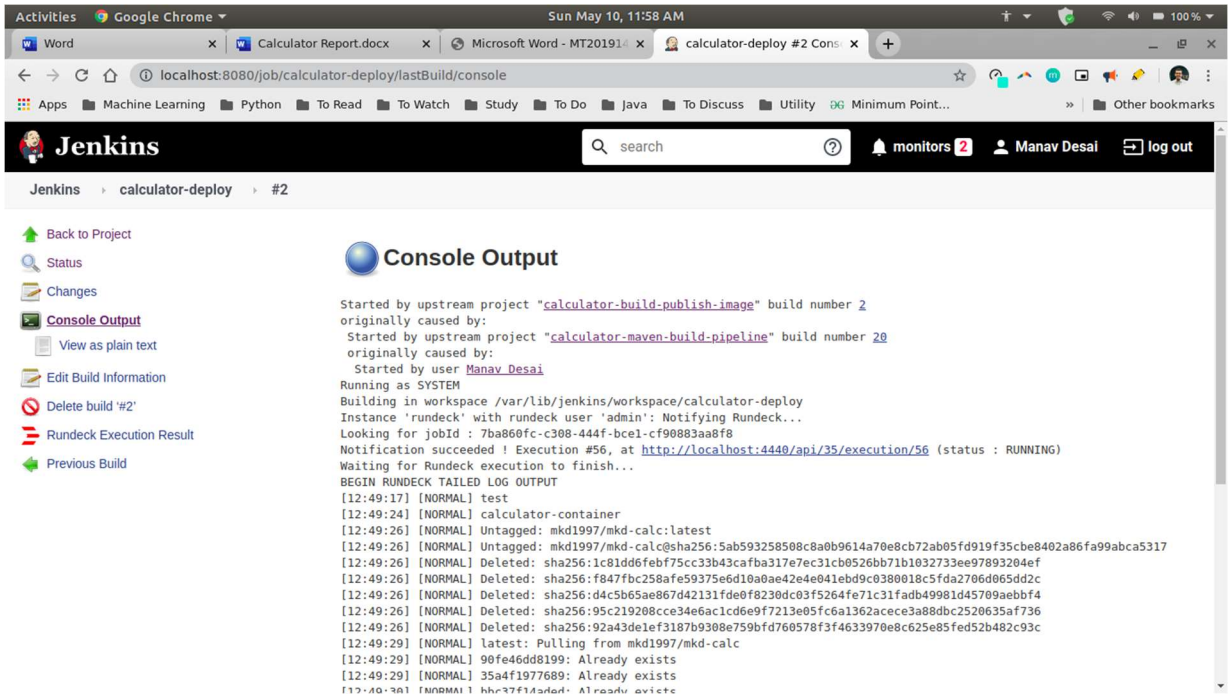


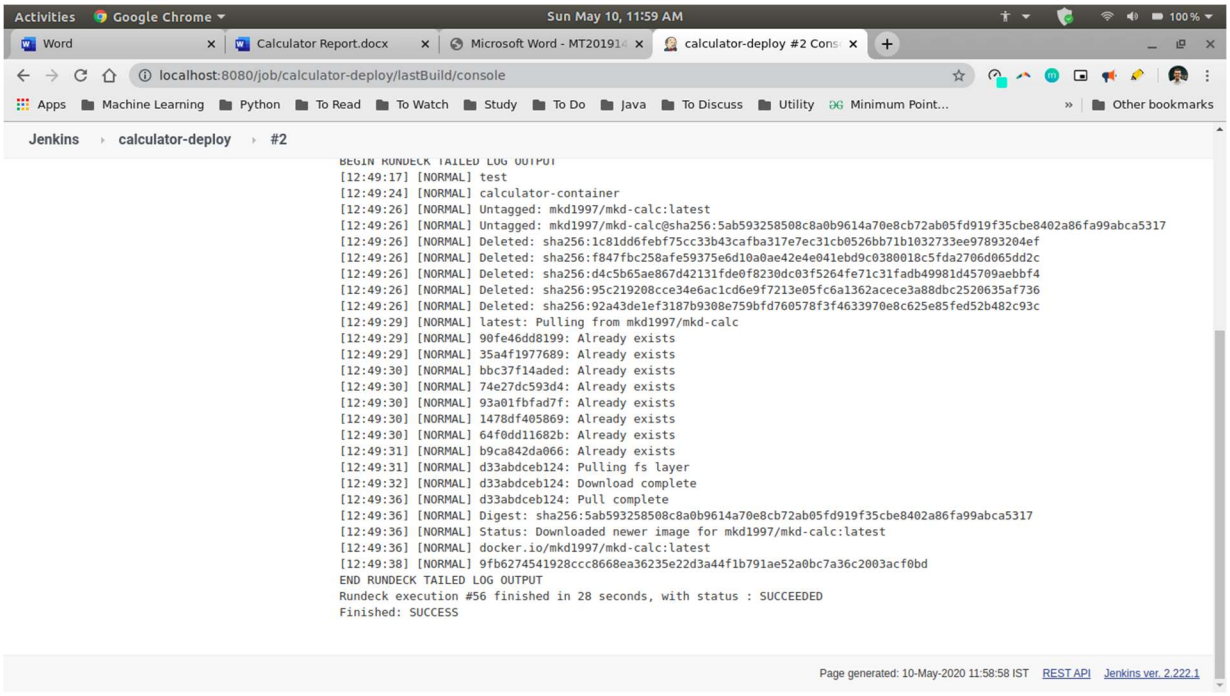
c. Post-Build Actions

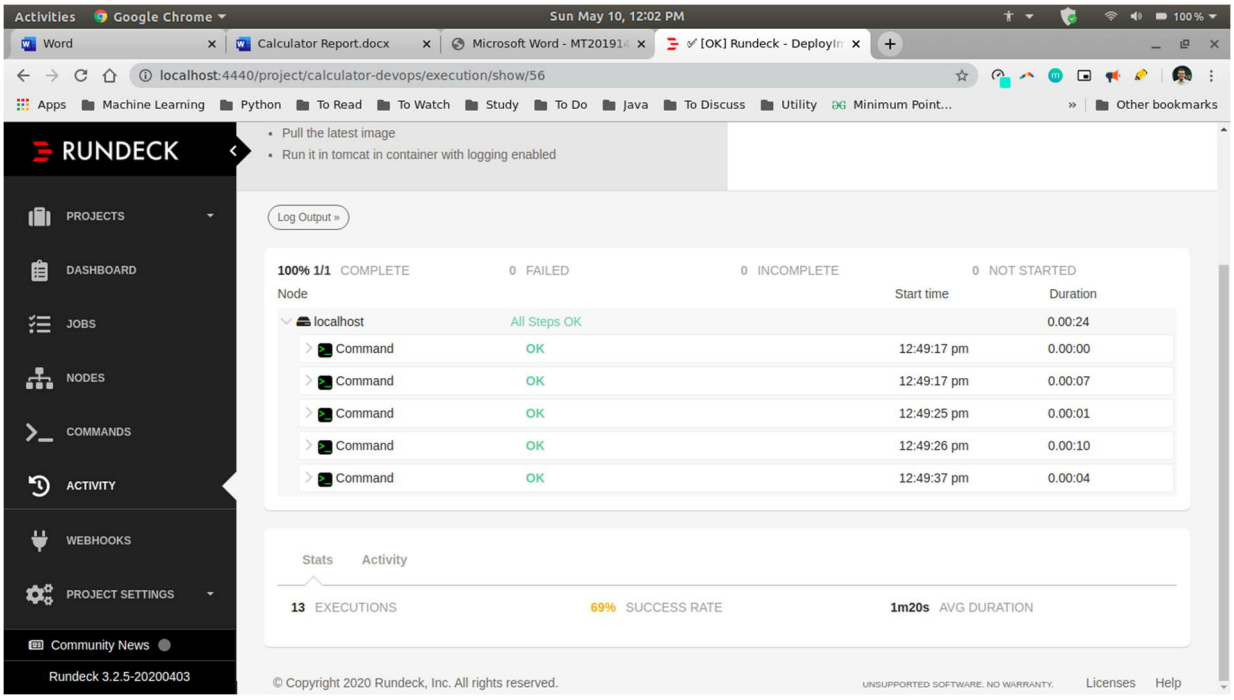


3. Click Save.

The output of the Jenkins job can be seen in Jenkins as well as Rundeck as shown below.




Activities
Google Chrome
Sun May 10, 11:59 AM
Word
Calculator Report.docx
Microsoft Word - MT201914
calculator-deploy #2 Cons
localhost:8080/job/calculator-deploy/lastBuild/console
Apps
Machine Learning
Python
To Read
To Watch
Study
To Do
Java
To Discuss
Utility
Minimum Point...
Other bookmarks
Jenkins
calculator-deploy
#2
BEGIN RUNDECK TAILED LOG OUTPUT
[12:49:17] [NORMAL] test
[12:49:24] [NORMAL] calculator-container
[12:49:26] [NORMAL] Untagged: mkd1997/mkd-calc:latest
[12:49:26] [NORMAL] Untagged: mkd1997/mkd-calc@sha256:5ab593258508c8a0b9614a70e8cb72ab05fd919f35cbe8402a86fa99abca5317
[12:49:26] [NORMAL] Deleted: sha256:1c81dd6febf75cc33b43cafba317e7ec31cb0526bb71b1032733ee97893204ef
[12:49:26] [NORMAL] Deleted: sha256:f847fbc258afe59375e6d10a0ae42e4e041ebd9c0380018c5fda2706d065dd2c
[12:49:26] [NORMAL] Deleted: sha256:d4c5b65ae867d42131fde0f8230dc03f5264fe71c31fadb49981d45709aebbf4
[12:49:26] [NORMAL] Deleted: sha256:95c219208cce34e6ac1cd6e9f7213e05fc6a1362acece3a88dbc2520635af736
[12:49:26] [NORMAL] Deleted: sha256:92a43de1ef3187b9308e759bfd760578f3f4633970e8c625e85fed52b482c93c
[12:49:29] [NORMAL] latest: Pulling from mkd1997/mkd-calc
[12:49:29] [NORMAL] 90fe46dd8199: Already exists
[12:49:29] [NORMAL] 35a4f1977689: Already exists
[12:49:30] [NORMAL] bbc37f14aded: Already exists
[12:49:30] [NORMAL] 74e27dc593d4: Already exists
[12:49:30] [NORMAL] 93a01fbfad7f: Already exists
[12:49:30] [NORMAL] 1478df405869: Already exists
[12:49:30] [NORMAL] 64f0dd11682b: Already exists
[12:49:31] [NORMAL] b9ca842da066: Already exists
[12:49:31] [NORMAL] d33abdceb124: Pulling fs layer
[12:49:32] [NORMAL] d33abdceb124: Download complete
[12:49:36] [NORMAL] d33abdceb124: Pull complete
[12:49:36] [NORMAL] Digest: sha256:5ab593258508c8a0b9614a70e8cb72ab05fd919f35cbe8402a86fa99abca5317
[12:49:36] [NORMAL] Status: Downloaded newer image for mkd1997/mkd-calc:latest
[12:49:36] [NORMAL] docker.io/mkd1997/mkd-calc:latest
[12:49:38] [NORMAL] 9fb6274541928ccc8668ea36235e22d3a44f1b791ae52a0bc7a36c2003acf0bd
END RUNDECK TAILED LOG OUTPUT
Rundeck execution #56 finished in 28 seconds, with status : SUCCEEDED
Finished: SUCCESS
Page generated: 10-May-2020 11:58:58 IST
REST API
Jenkins ver. 2.222.1


Activities
Google Chrome
Sun May 10, 12:02 PM
Word
Calculator Report.docx
Microsoft Word - MT201914
[OK] Rundeck - DeployIn
localhost:4440/project/calculator-devops/execution/show/56
Apps
Machine Learning
Python
To Read
To Watch
Study
To Do
Java
To Discuss
Utility
Minimum Point...
Other bookmarks
RUNDECK
PROJECTS
DASHBOARD
JOBS
NODES
COMMANDS
ACTIVITY
WEBHOOKS
PROJECT SETTINGS
Community News
Rundeck 3.2.5-20200403
Pull the latest image
Run it in tomcat in container with logging enabled
Log Output »
100% 1/1 COMPLETE
0 FAILED
0 INCOMPLETE
0 NOT STARTED
Node
Start time
Duration
localhost
All Steps OK
0:00:24
Command
OK
12:49:17 pm
0:00:00
Command
OK
12:49:17 pm
0:00:07
Command
OK
12:49:25 pm
0:00:01
Command
OK
12:49:26 pm
0:00:10
Command
OK
12:49:37 pm
0:00:04
Stats
Activity
13 EXECUTIONS
69% SUCCESS RATE
1m20s AVG DURATION
© Copyright 2020 Rundeck, Inc. All rights reserved.
UNSUPPORTED SOFTWARE. NO WARRANTY.
Licenses
Help

3.6 Monitoring

Monitoring refers to monitoring the deployed artifacts in real-time to keep a check on faults and measure performance. This is done by analyzing logs generated by the system. The Elastic Stack, also known as BELK Stack, comprises of Elasticsearch, Logstash, Kibana and Beats.

Elasticsearch is a modern search and analytics engine which is based on Apache Lucene. It is used as to store and index logs and can be then queried to extract meaningful insights. It can be used for numerous types of data including textual, numerical, geospatial, structured, and unstructured.

Logstash is a tool that is used for parsing logs. It is very useful in parsing unstructured logs and giving them structure so that logs can be efficiently searched and analyzed. Log aggregated and processed by Logstash go through 3 stages – collection, processing and dispatching.

Kibana adds a visualization layer to the Elastic Stack. It is a browser-based user interface that can be used to search, analyze and visualize the data stored in Elasticsearch indices.

Beats are a collection of open source log shippers that act as agents installed on the different servers in your environment for collecting logs or metrics. These shippers are designed to be lightweight in nature — they leave a small installation footprint, are resource efficient, and function with no dependencies. Common Beats that are used today included Filebeat, Metricbeat, Winlogbeat, Packetbeat, etc.

I use the Elastic Stack to perform log analysis in the Calculator project. I use Filebeat to collect logs generated by Tomcat which are then given to Elasticsearch for analysis. Kibana is used to create a pie chart that shows the ratio of requests that were successful.

3.6.1 Installing the Elastic Stack

To install the Elastic Stack, follow the given steps:

1. First install Elasticsearch using the following commands.

   ```
   > wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
   > sudo apt-get update> sudo apt-get install apt-transport-https
   > echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee -a
   /etc/apt/sources.list.d/elastic-7.x.list
   > sudo apt-get install elasticsearch
   ```

2. Elasticsearch runs at http://localhost:9200. To start Elasticsearch

   ```
   > sudo service elasticsearch start
   ```

3. Now, install Kibana using the following commands.

   ```
   > sudo apt-get install kibana
   ```

4. Make sure you have the following configuration in /etc/kibana/kibana.yml.

   ```
   server.port: 5601
   elasticsearch.url: http://localhost:9200
   ```

5. Kibana runs at http://localhost:5601. To start Elasticsearch

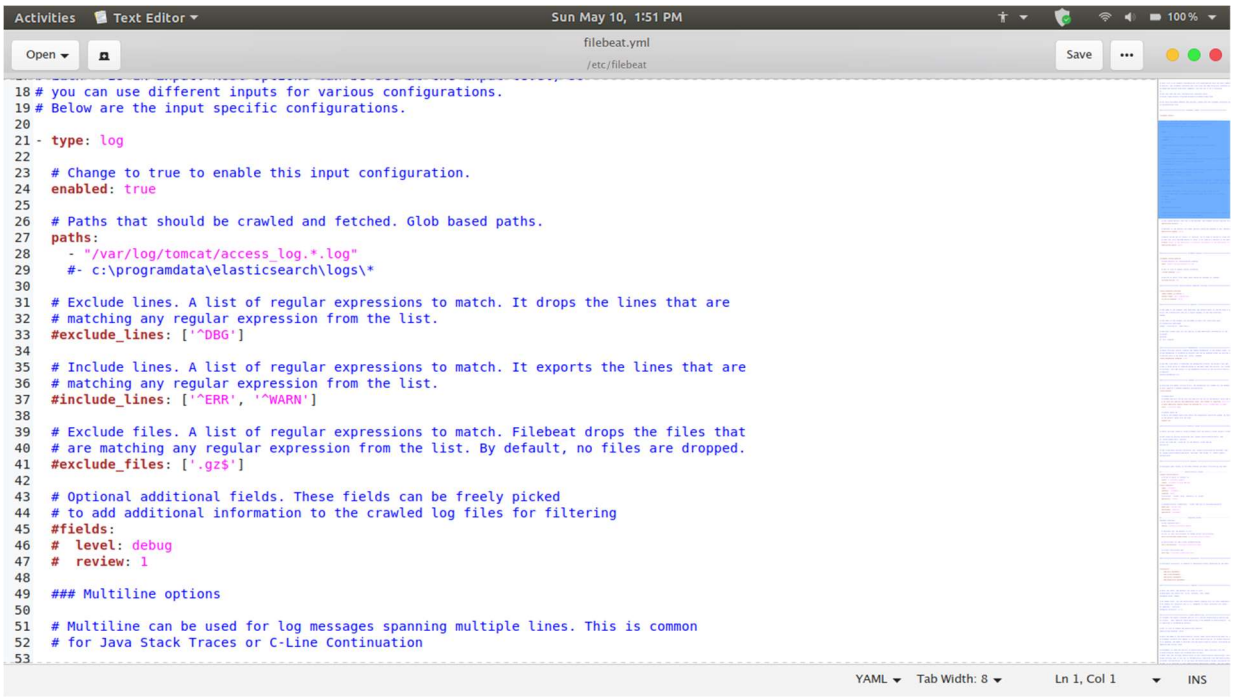
   > sudo service elasticsearch start

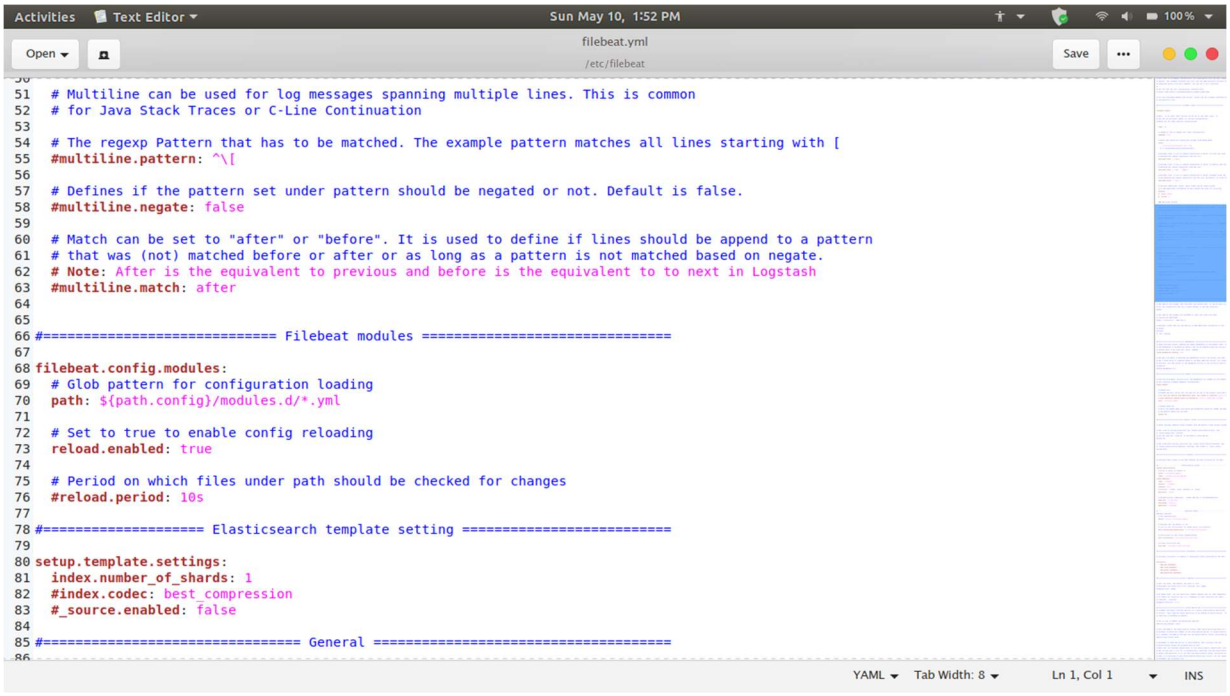
6. Now, install filebeat using the following commands.

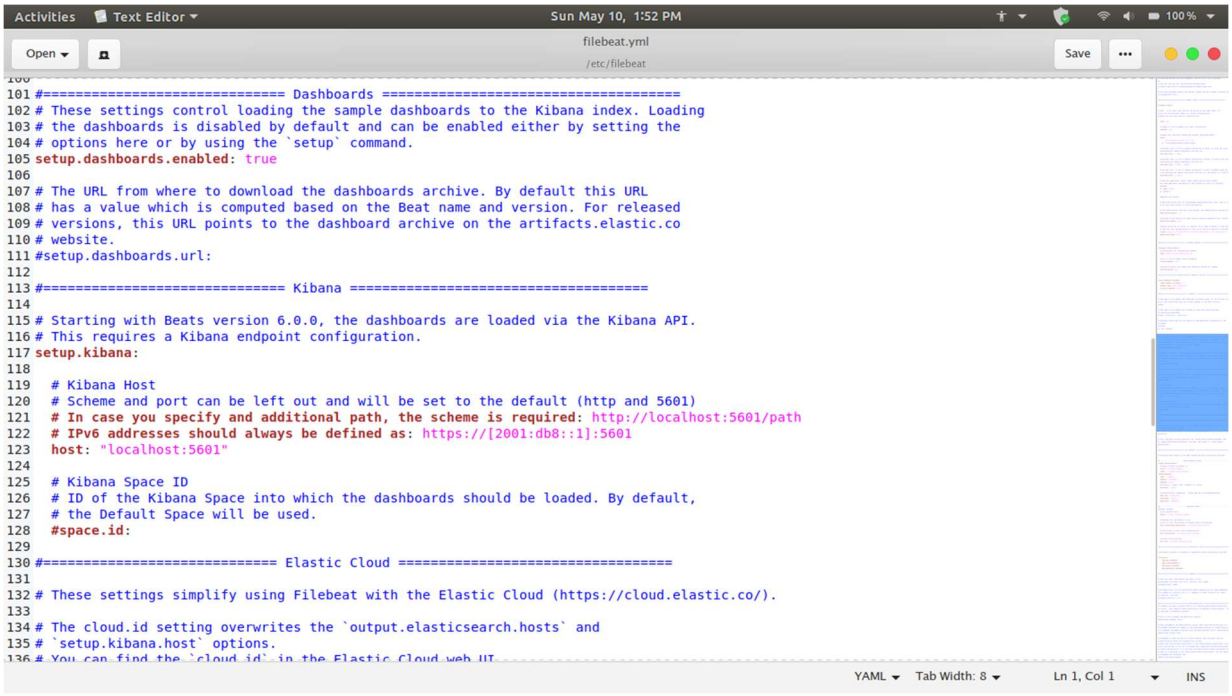
   > sudo apt-get install filebeat

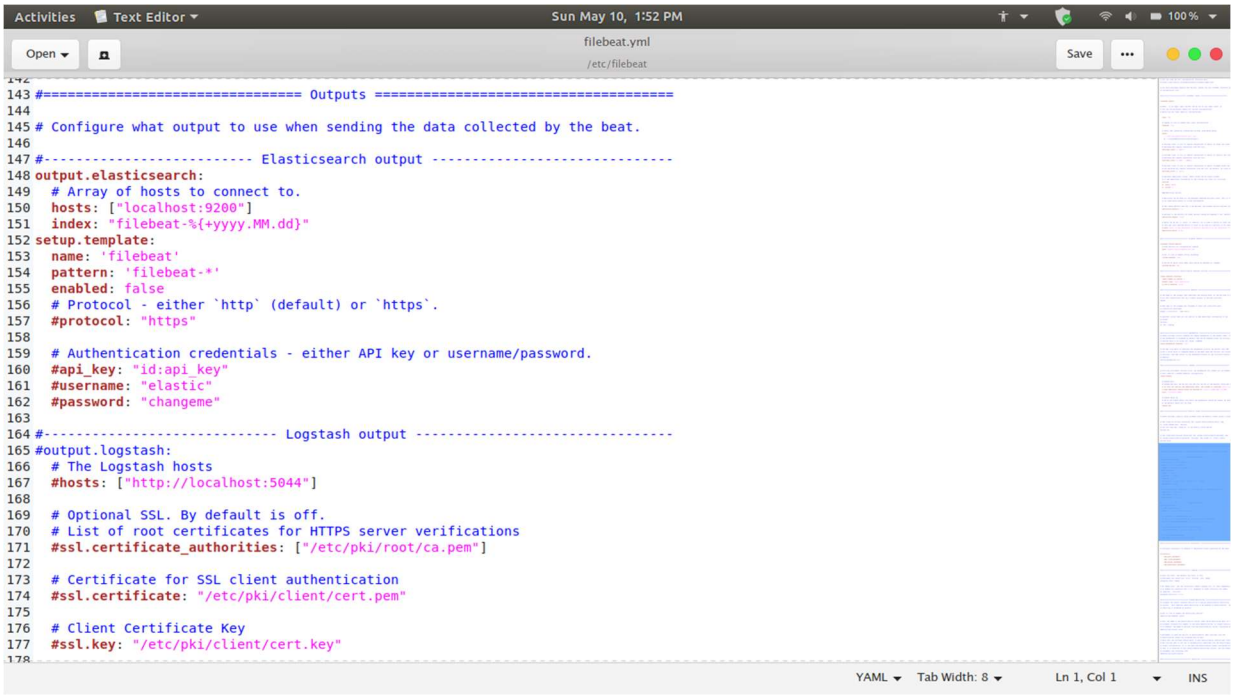
3.6.2 Configuring Filebeat

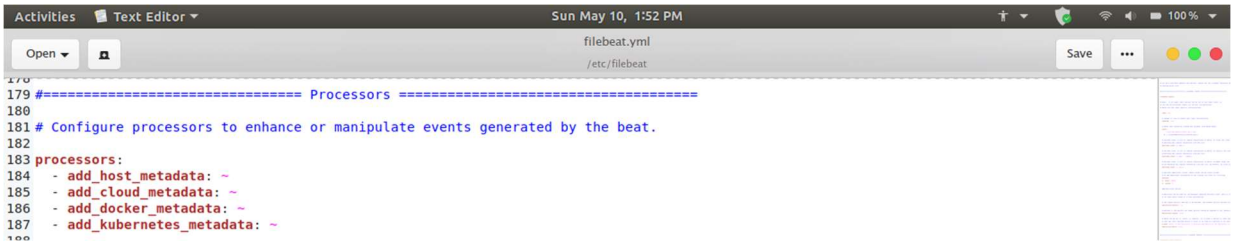
I have configured Filebeat to directly send the Tomcat access logs to Elasticsearch for processing. The imaportance of deploying the image in a container with /var/log/tomcat folder in container mapped to the same folder on host system can now be seen. The Elastic Stack runs outside the container. Since, Filebeat cannot access the folders in the container (where the logs are generated), it accesses the folder on the host system. The path of configuration file for any Beats in Linux is /etc/filebeat/filebeat.yml. To configue Filebeat, edit the yml file as shown in the screenshots below.




Activities
Text Editor
Sun May 10, 1:52 PM
100%
Open
filebeat.yml
/etc/filebeat
Save
Multiline can be used for log messages spanning multiple lines. This is common
for Java Stack Traces or C-Line Continuation
The regexp Pattern that has to be matched. The example pattern matches all lines starting with [
#multiline.pattern: ^\[
Defines if the pattern set under pattern should be negated or not. Default is false.
#multiline.negate: false
Match can be set to "after" or "before". It is used to define if lines should be append to a pattern
that was (not) matched before or after or as long as a pattern is not matched based on negate.
Note: After is the equivalent to previous and before is the equivalent to to next in Logstash
#multiline.match: after
#============================ Filebeat modules ===============================
filebeat.config.modules:
Glob pattern for configuration loading
path: ${path.config}/modules.d/*.yml
Set to true to enable config reloading
reload.enabled: true
Period on which files under path should be checked for changes
#reload.period: 10s
#==================== Elasticsearch template setting ==========================
setup.template.settings:
#index.number_of_shards: 1
#index.codec: best_compression
#_source.enabled: false
#================================ General =====================================
YAML
Tab Width: 8
Ln 1, Col 1
INS


Activities
Text Editor
Sun May 10, 1:52 PM
100%
Open
filebeat.yml
/etc/filebeat
Save
#============================ Dashboards =====================================
These settings control loading the sample dashboards to the Kibana index. Loading
the dashboards is disabled by default and can be enabled either by setting the
options here or by using the `setup` command.
setup.dashboards.enabled: true
The URL from where to download the dashboards archive. By default this URL
has a value which is computed based on the Beat name and version. For released
versions, this URL points to the dashboard archive on the artifacts.elastic.co
website.
#setup.dashboards.url:
#============================ Kibana =====================================
Starting with Beats version 6.0.0, the dashboards are loaded via the Kibana API.
This requires a Kibana endpoint configuration.
setup.kibana:
Kibana Host
Scheme and port can be left out and will be set to the default (http and 5601)
In case you specify and additional path, the scheme is required: http://localhost:5601/path
IPv6 addresses should always be defined as: https://[2001:db8::1]:5601
host: "localhost:5601"
Kibana Space ID
ID of the Kibana Space into which the dashboards should be loaded. By default,
the Default Space will be used.
#space.id:
#============================ Elastic Cloud ==================================
These settings simplify using Filebeat with the Elastic Cloud (https://cloud.elastic.co/).
The cloud.id setting overwrites the `output.elasticsearch.hosts` and
`setup.kibana.host` options.
You can find the `cloud.id` in the Elastic Cloud web UI.
YAML
Tab Width: 8
Ln 1, Col 1
INS





Restart Filebeat after editing the configuration file.

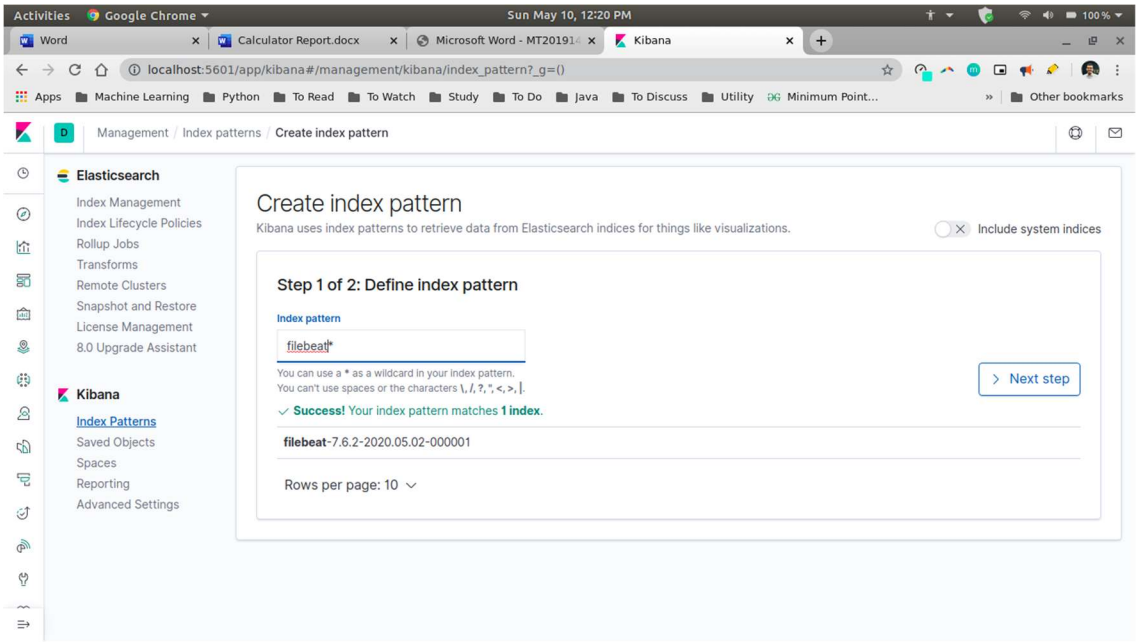
> sudo service filebeat restart

3.6.3 Visualizing Logs in Kibana

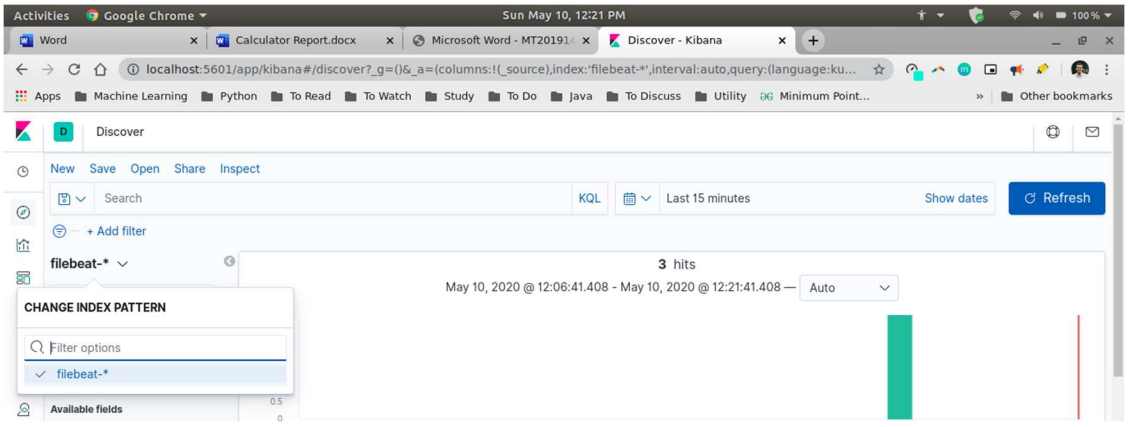
I now create visualizations to show the number of requests that our service failed to serve. I first create an index pattern which allows Kibana to fetch the logs coming from Filebeat. I then use the HTTP Response code to show the proportion of requests that were served successfully.

To do this, follow the given steps:

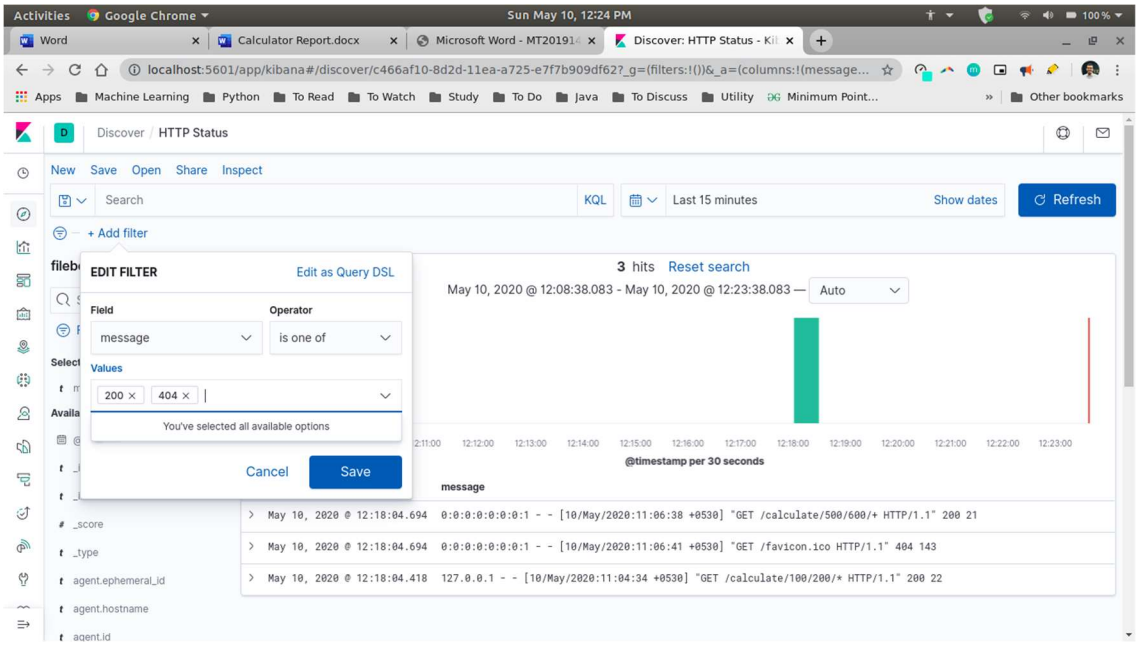
1. On the Kibana dashboard, go to Management -> Index Pattern. Create the index pattern filebeat-*



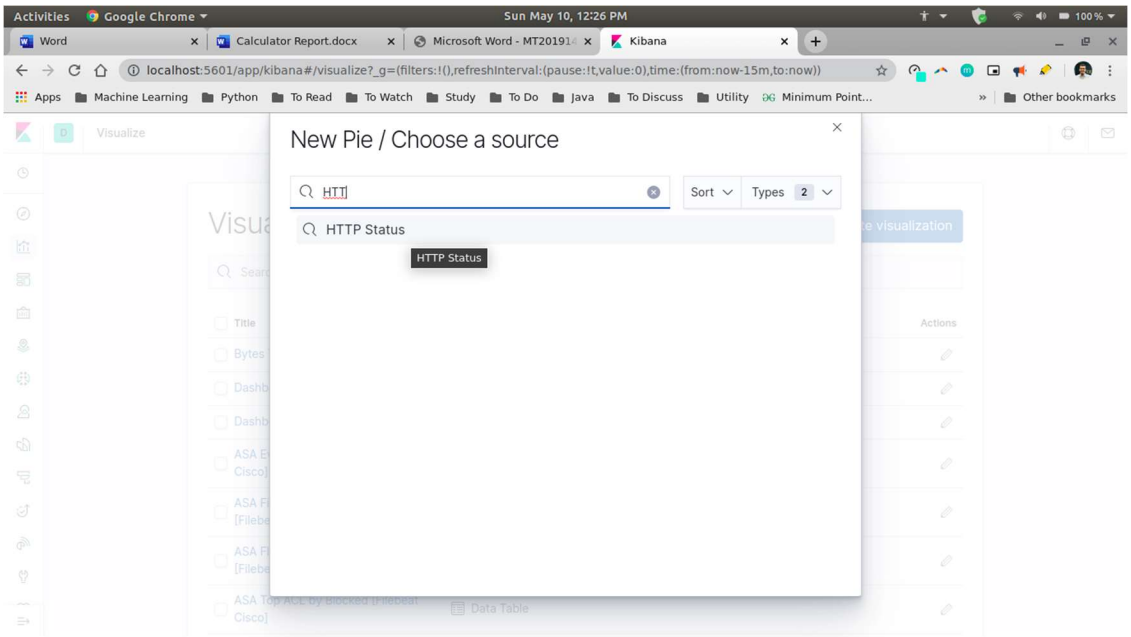
2. On the Kibana dashboard, go to Discover and change the index pattern to filebeat-* from the dropdown list.



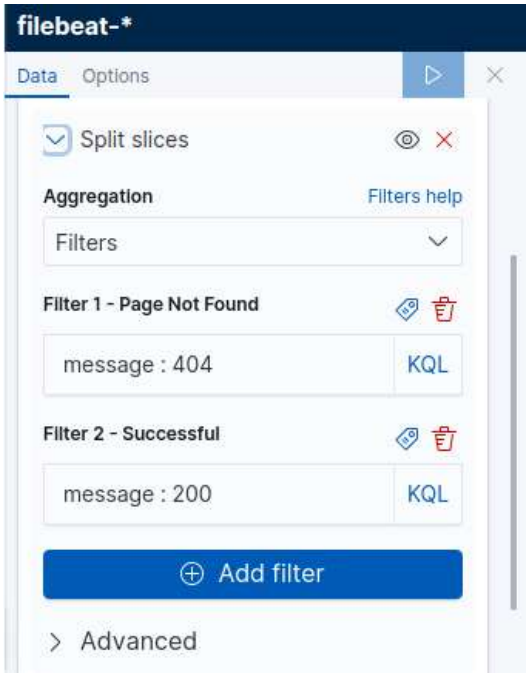
3. Add a filter as shown below.



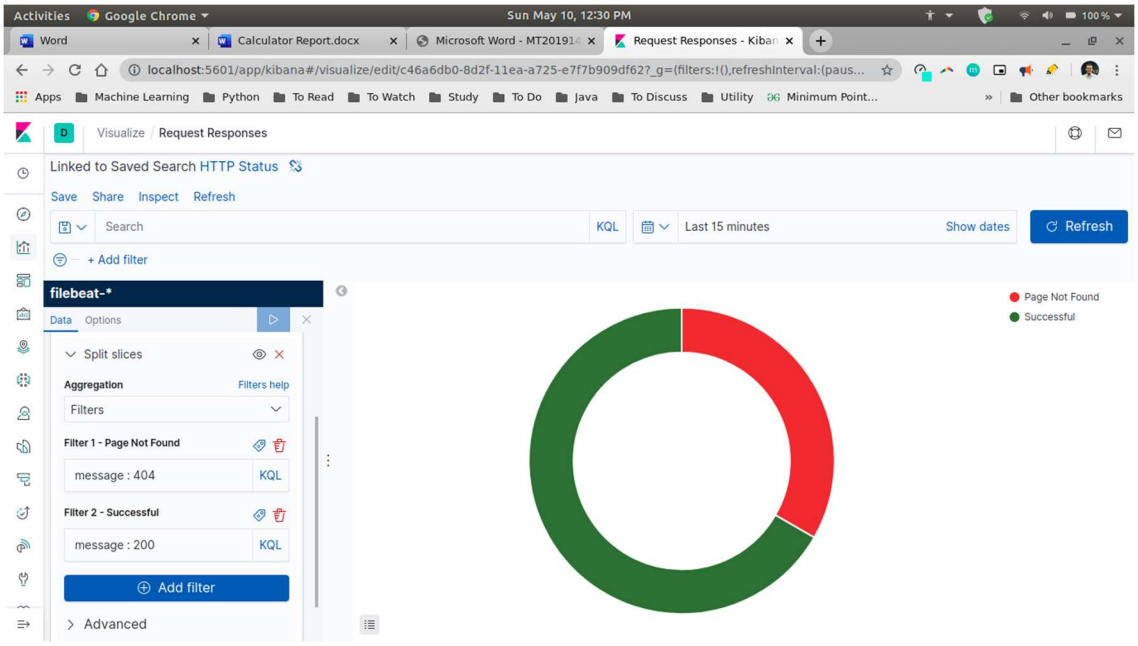
4. Save the search by first clicking the Save button and then clicking the Save link on top. This search can now be used as a data source for creating charts.
5. On the Kibana dashboard, go to Visualise and click on Create Visualisation. Choose Pie Chart there.
6. Choose the saved search in the Choose Source dialog box.



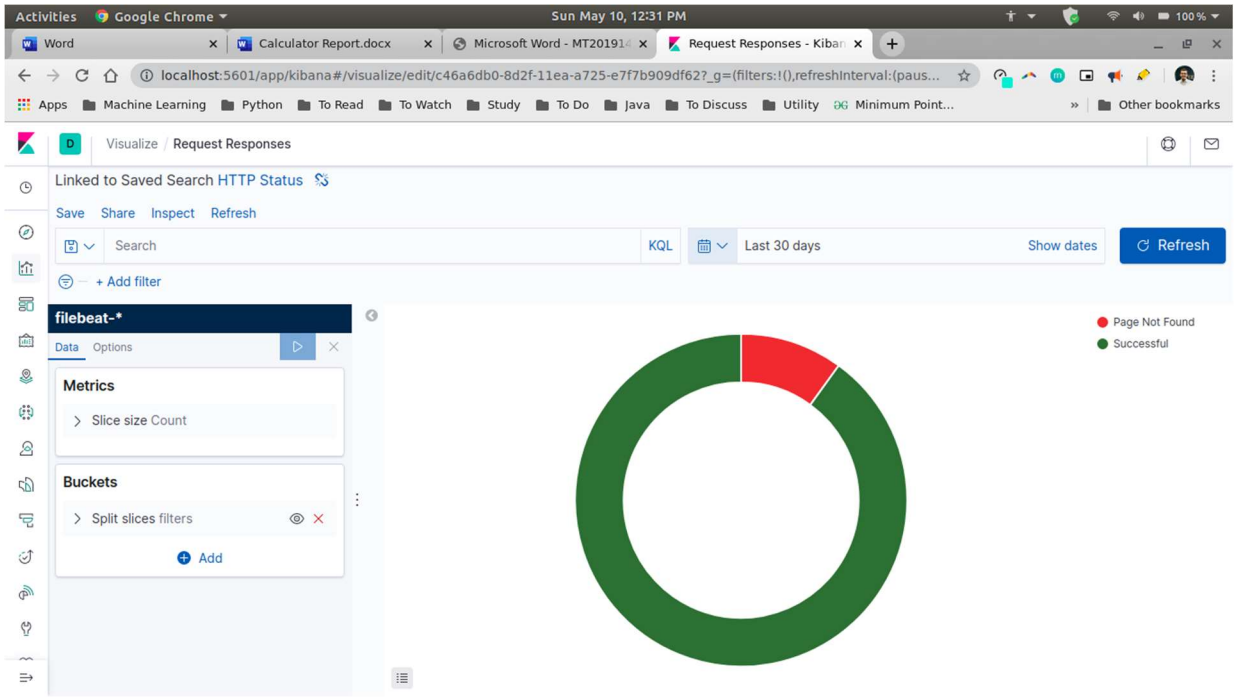
7. Add buckets on the left pane.



8. The visualization can now be seen. You can also change the colour scheme used and the labels from the legend on the top right corner of the screen.



The final output is shown below. The visualisation will get updated in real-time as the system gets more and more requests.



4. Conclusion

In this project, I automated the entire SDLC using DevOps toolchain. This makes the life of development team and operations team easy as the DevOps pipeline gives the comfort of making code changes easily and also reduces the chances of encountering errors in production. The toolchain allows software companies to quickly build, test and deploy new versions of their products. This can be very useful to companies like Amazon which release new versions many times a day.

5. References

I have referred to the following resources while developing this project:

1. https://git-scm.com/doc
2. https://maven.apache.org/guides/index.html
3. https://junit.org/junit5/docs/current/user-guide/
4. https://www.jenkins.io/doc/
5. https://docs.docker.com/
6. https://logz.io/learn/complete-guide-elk-stack/
7. https://logz.io/blog/filebeat-tutorial/
8. https://logz.io/blog/apache-tomcat-monitoring/