Software Production Engineering

CI/CD Pipeline on Scientific Calculator Program

Akshay Mahesh Gudiyawar

MT2020137

Github repo link: https://github.com/akshaymg99/DevOps_Calculator

Docker Hub link: https://hub.docker.com/repository/docker/akshaymg99/calculator

Table of Contents

1. Introduction

Here we try to automate the development, testing and deployment of a scientific calculator program by creating a CI/CD pipeline. The calculator program developed here is a Spring Boot project hosted on a Tomcat web server running in a Docker container. This scientific calculator supports following operations: Square root function, factorial function, natural logarithm and power function.

The CI/CD pipeline for this project 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

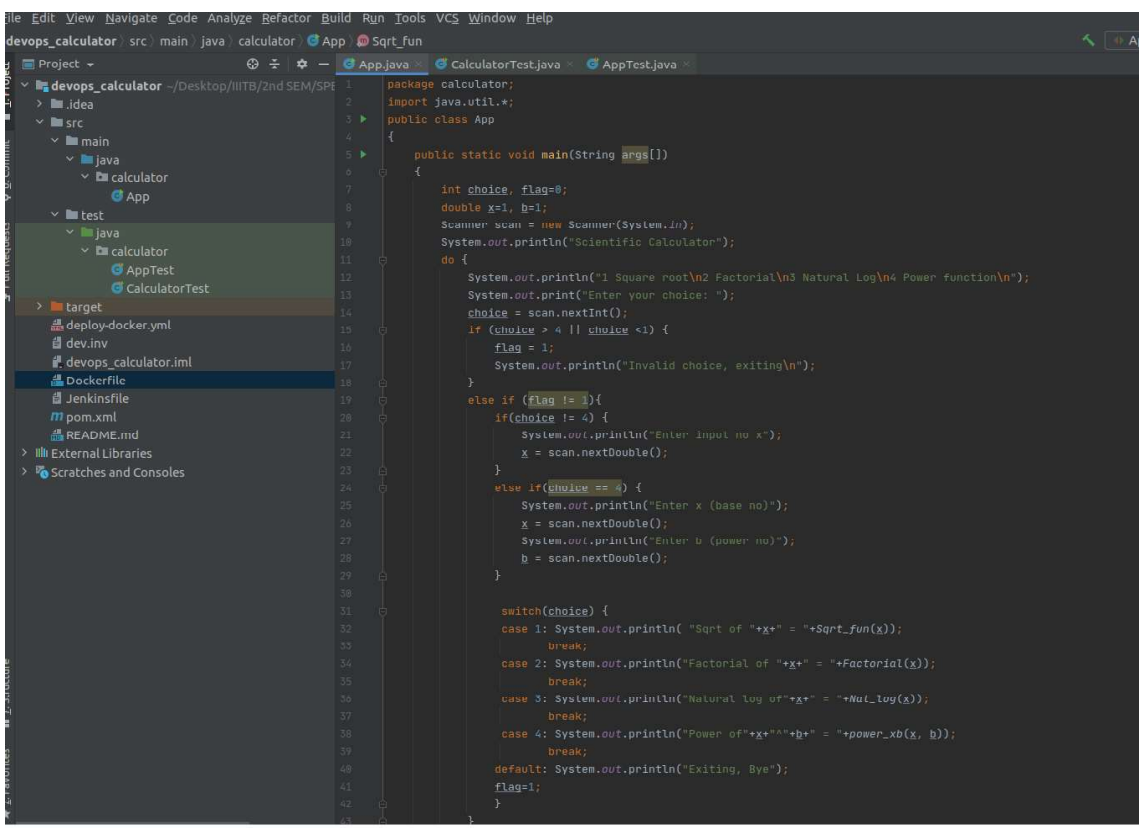
5. Deployment: Docker, Rundeck, Jenkins

6. Monitoring: Elastic Stack

The source code link: https://github.com/akshaymg99/DevOps_Calculator.git

2. Source Code

The Calculator App implementing mentioned operations is implemented as below:



The Calculator Test App program written with Junit for making test cases is done as follows (Unit test cases): Junit is a testing framework for Java programs with which we can write test cases to check if our program is behaving as expected.

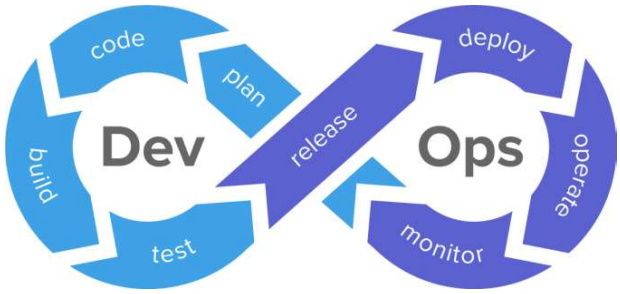
```java
            }
            System.out.println("\n");
            }while(flag==0);
    }

    static double Sqrt_fun(double x) {
        if (x <= 0) {
            throw new IllegalArgumentException("Square root of zero and negative doesn't exits in Real nu
        }
        else
            return Math.sqrt(x);
    }
    static double Factorial(double x) {
        if (x < 0) {
            throw new IllegalArgumentException("Factorial doesn't exist for negative numbers");
        }
        else {
            double fact = 1;
            for(int i=2; i<=x; i++) {
                fact = fact * i;
            }
            return fact;
        }
    }
    static double Nat_log(double x) {
        if(x <= 0) {
            throw new IllegalArgumentException("Number cannot be zero or negative");
        }
        else
            return Math.log(x);
    }
    static double power_xb(double x, double b) {
        if (x == 0) {
            throw new IllegalArgumentException("Base cannot be zero");
        }
        else
            return Math.pow(x, b);
    }
}
```

Rigorous test of Calculator App is done as follows:

```java
package calculator;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class AppTest
{
    @Test
    public void shouldAnswerWithTrue() { assertTrue( condition: true ); }
}
```

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)

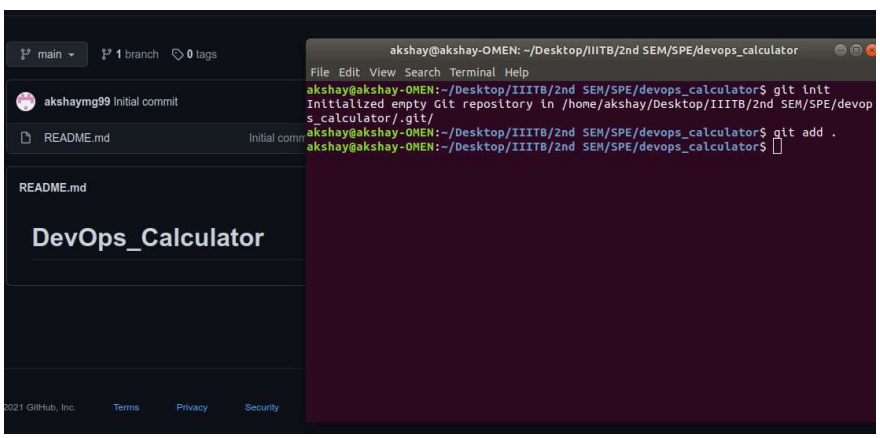
Source control refers to the practice of tracking and managing changes to code. Source control management (SCM) systems provide a running history of code development and help to resolve conflicts when merging contributions from multiple sources. I use Git as the SCM here. Github is an online repository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

The development of this program has been done incrementally. The commits done can be seen at https://github.com/akshaymg99/DevOps_Calculator/commits/master

After developing code in an IDE, to post it to SCM (Git), we have to execute the following commands in our project working directory:

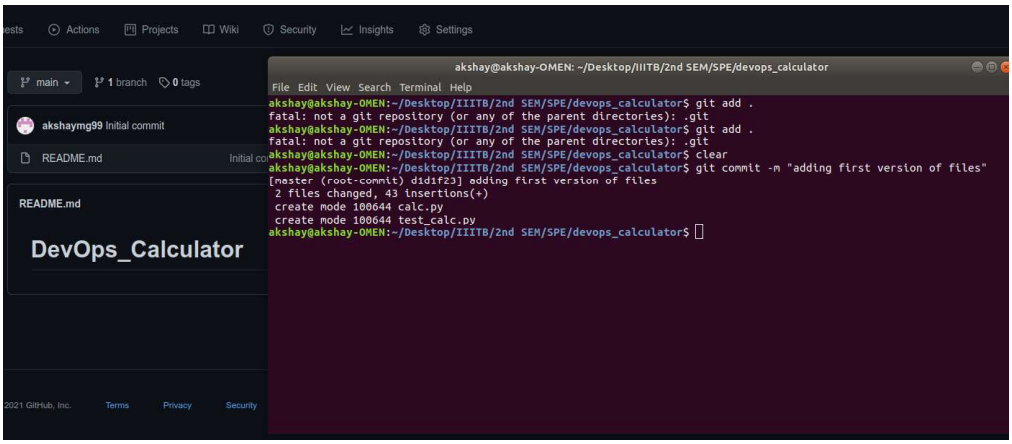
`git remote add origin <remote repo url>` -> create connection to remote repo

`git init` → for initializing local system directory as git directory to perform further actions



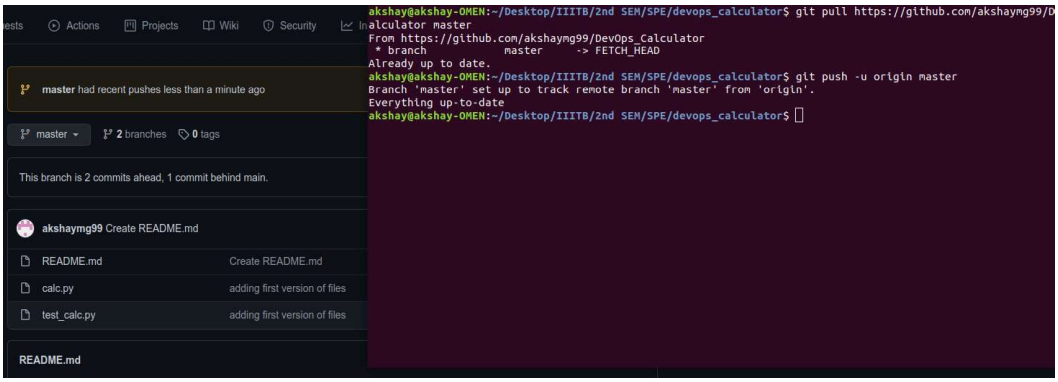
`git add . or git add /files` -> for adding entire directory or files to staging area

`git commit –m "message"` -> to take a snapshot of the current state of directory



`git pull <remote repo url>` -> to synchronize remote repository with local repo

`git push -u origin master` -> to push the local repo files to remote repository



Other git commands:

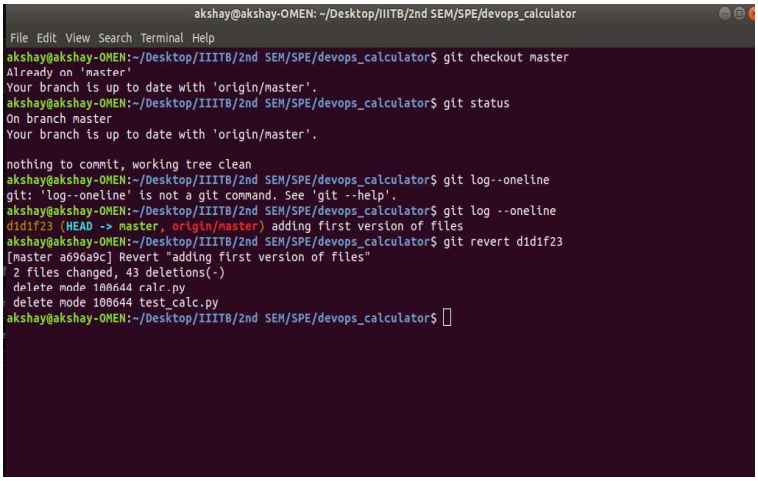
`git revert <commit>` -> undoes all changes made in <commit>

`git status` -> list which files are staged, unstaged & untracked

`git log` -> display commit history

`git checkout -b <branch>` -> create & checkout a new branch

`git merge <branch>` -> merge <branch> into current branch



3.2 Build and Test

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.

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 scientific calculator project is shown here.

```
App.java   CalculatorTest.java   AppTest.java
import org.junit.*;

public class CalculatorTest {

        private App calculator;

        @Before
        public void setUp() { calculator = new App(); }

        @Test
        public void testSqrt_fun() {
            double x = 25;
            double expectedResult = 5.0;
            double result = calculator.Sqrt_fun(x);
            Assert.assertEquals(expectedResult, result,  delta: 0);
        }

        @Test
        public void testFactorial() {
            double x = 5;
            double expectedResult = 120;
            double result = calculator.Factorial(x);
            Assert.assertEquals(expectedResult, result,  delta: 0);
        }

        @Test
        public void testNat_log() {
            double x = 100;
            double expectedResult = 4.6;
            double result = calculator.Nat_log(x);
            Assert.assertEquals(expectedResult, result,  delta: 01);
        }

        @Test
        public void testpower_xb() {
            double x = 9.0;
            double b = 2.0;
            double expectedResult = 81.0;
            double result = calculator.power_xb(x, b);
            Assert.assertEquals(expectedResult, result,  delta: 0.01);
        }

}
```

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 process 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. 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

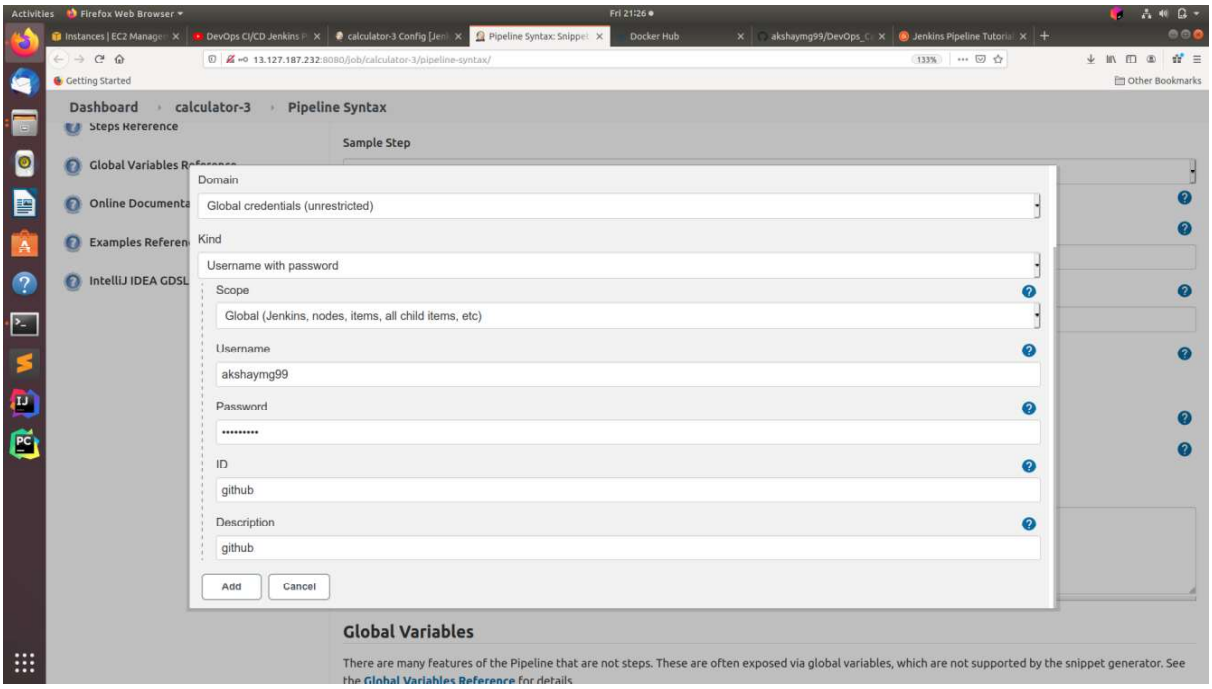
   > sudo apt-get install jenkins –y

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.



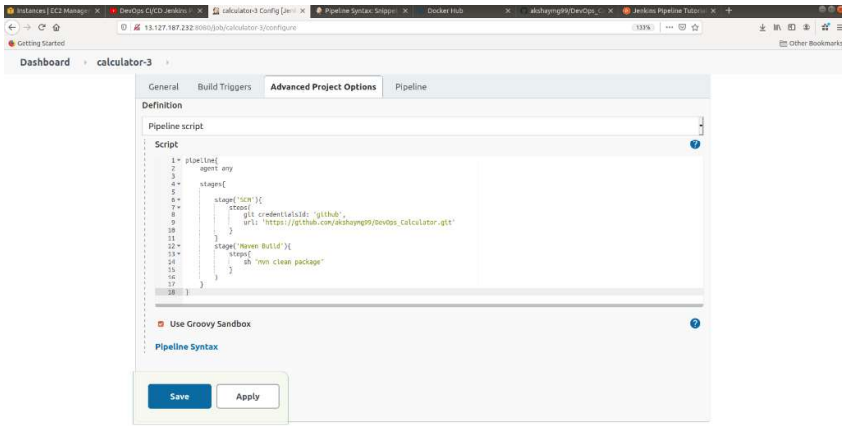
3.3.2 Jenkins Pipeline

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.

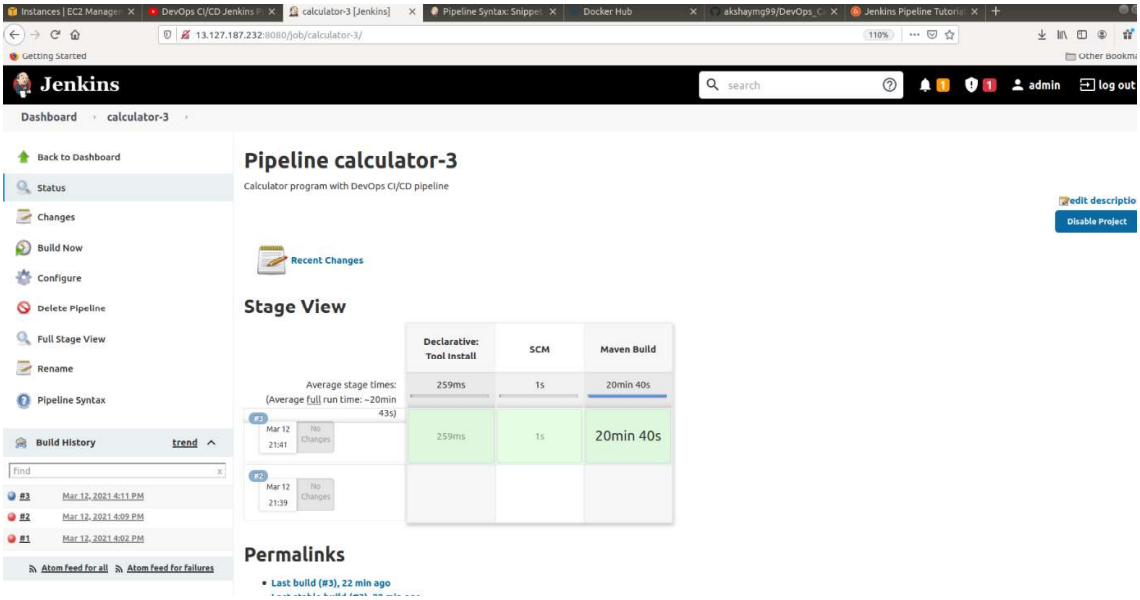
```
App.java × CalculatorTest.java × AppTest.java × Jenkinsfile ×
pipeline{
    agent any
    tools {
      maven 'maven3'
    }

    environment {
      DOCKER_TAG = getVersion()
    }


    stages{
        stage('SCM'){
            steps{
                git credentialsId: 'github',
                url: 'https://github.com/akshaymg99/DevOps_Calculator.git'
            }
        }
        stage('Maven Build'){
            steps{
                sh "mvn clean package"
            }
        }
        stage('Docker Build'){
            steps{
                sh "docker build . -t akshaymg99/calculator:latest "
            }
        }
        stage('DockerHub Push'){
            steps{
                withCredentials([string(credentialsId: 'docker-hub', variable: 'dockerHubPwd')]) {
                    sh "docker login -u akshaymg99 -p ${dockerHubPwd}"
                }
                sh "docker push akshaymg99/calculator:latest "
            }

        }


    }

}
```

Jenkins task: Maven Build pipeline

This Jenkins project is responsible for polling the Github repository 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.



Jenkins also shows a graphical view of pipeline execution after triggering a build



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 on Ubuntu, follow the 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 akshaymg99

> sudo usermod –aG docker Jenkins

3.4.2 Building and Publishing Images

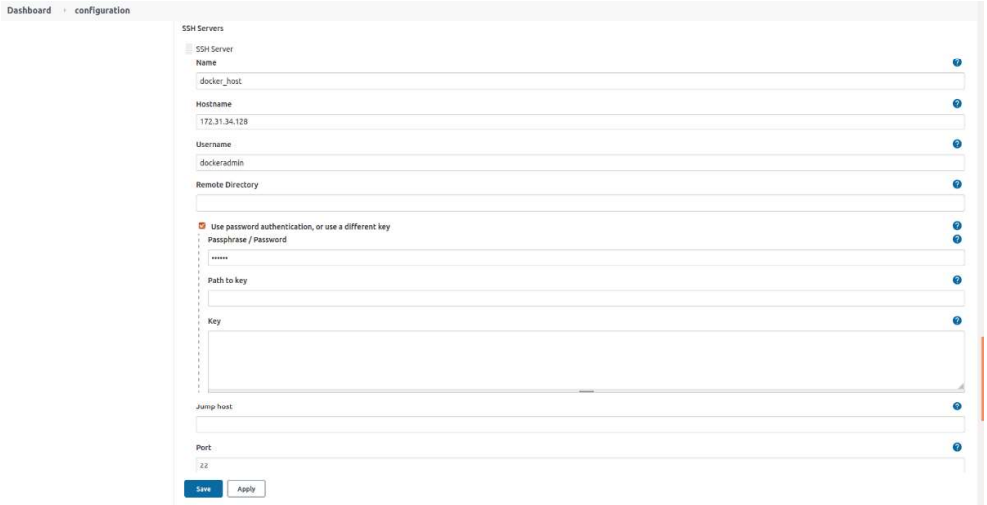
Maven build outputs a jar file consisting of the compiled classes, their dependencies and also the embedded Tomcat server. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.

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

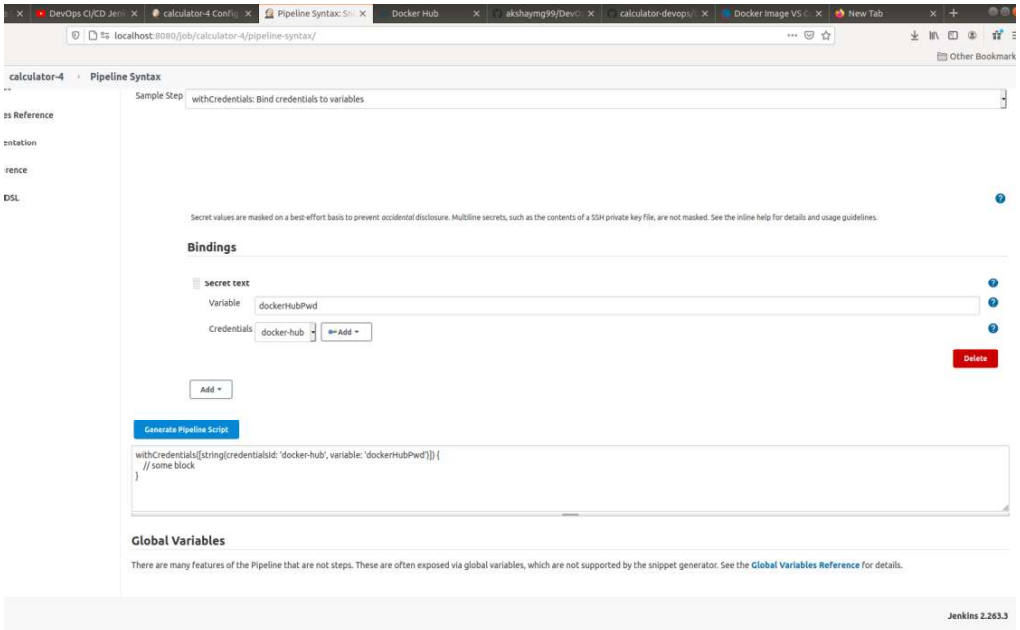
3.4.3 Jenkins Pipeline Task 2: Publish docker image to Docker Hub

We now use this Jenkins project to build the Docker image and publish it to Dockerhub.

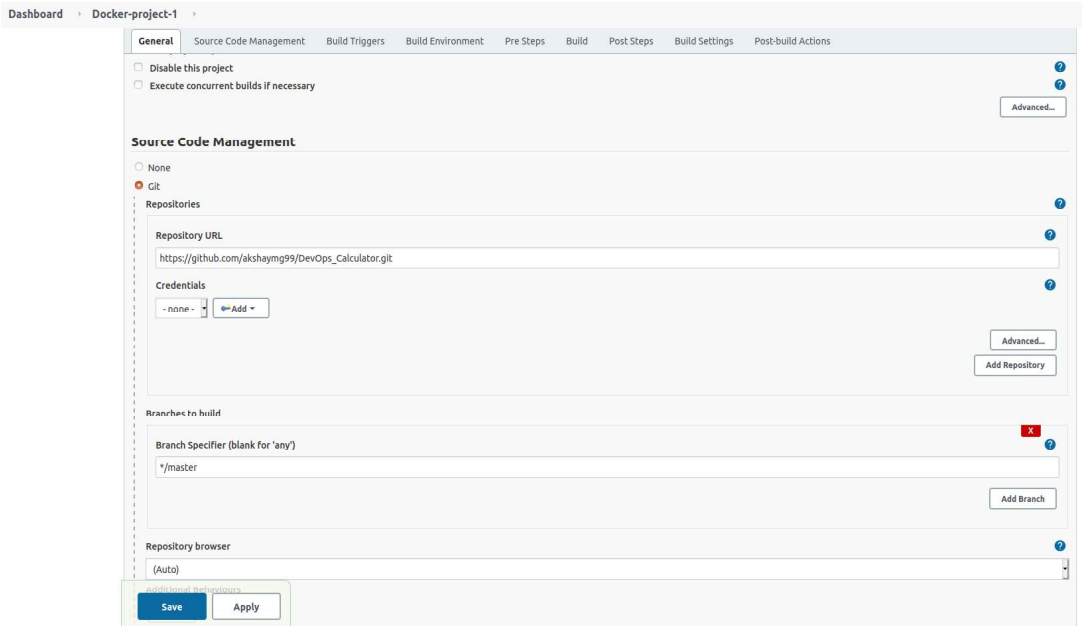
We first add docker-hub credentials to Jenkins so that it can access it while executing the pipeline.



To hide the credentials in script, we generate pipeline script with for docker password



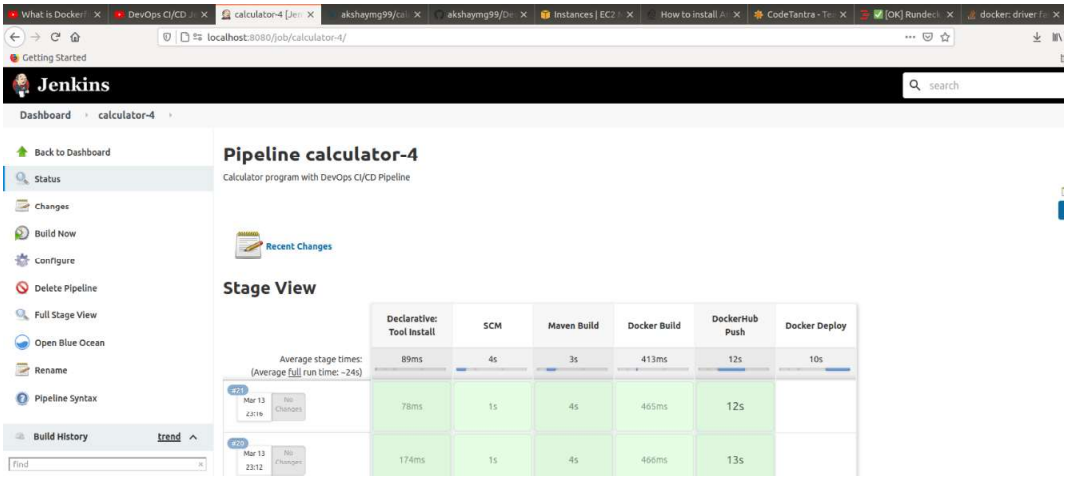
We also add git repo link to pull the latest code from repository



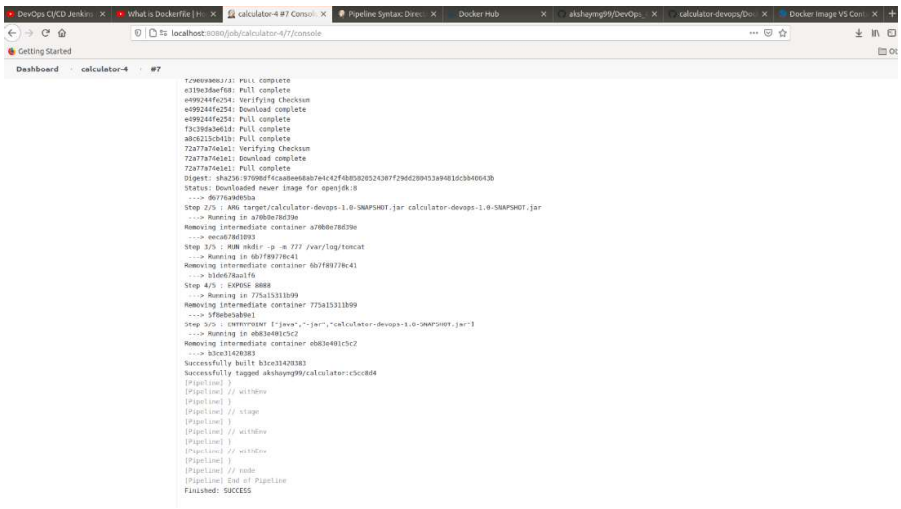
Jenkins pipeline Script to build and push image to docker hub

```
App.java × CalculatorTest.java × AppTest.java × Jenkinsfile ×
pipeline{
    agent any
    tools {
      maven 'maven3'
    }

    environment {
      DOCKER_TAG = getVersion()
    }


    stages{
        stage('SCM'){
            steps{
                git credentialsId: 'github',
                url: 'https://github.com/akshaymg99/DevOps_Calculator.git'
            }
        }
        stage('Maven Build'){
            steps{
                sh "mvn clean package"
            }
        }
        stage('Docker Build'){
            steps{
                sh "docker build . -t akshaymg99/calculator:latest "
            }
        }
        stage('DockerHub Push'){
            steps{
                withCredentials([string(credentialsId: 'docker-hub', variable: 'dockerHubPwd')]) {
                    sh "docker login -u akshaymg99 -p ${dockerHubPwd}"
                }
                sh "docker push akshaymg99/calculator:latest "
            }

        }


    }

}
```

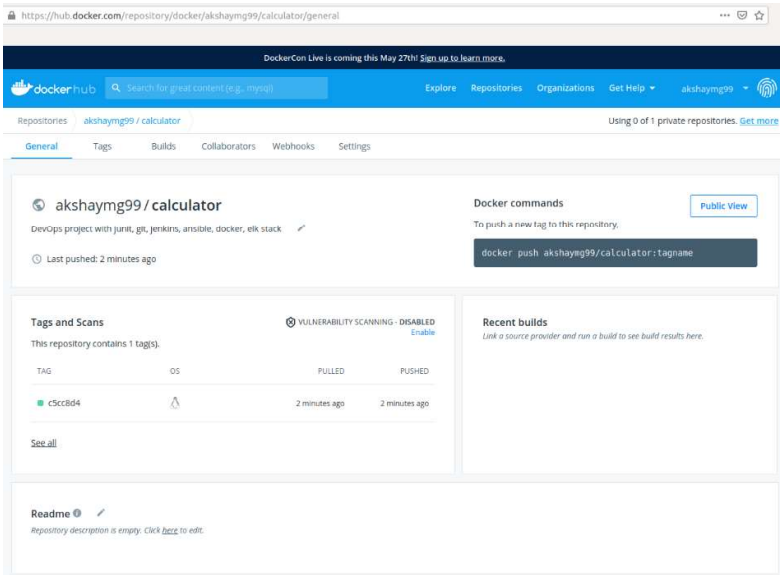
Pipeline visualization for docker build and push stages



Console output In Jenkins



Docker Hub image



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 our case) is created and published to Dockerhub, we 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 <deb package>

4. Start Rundeck

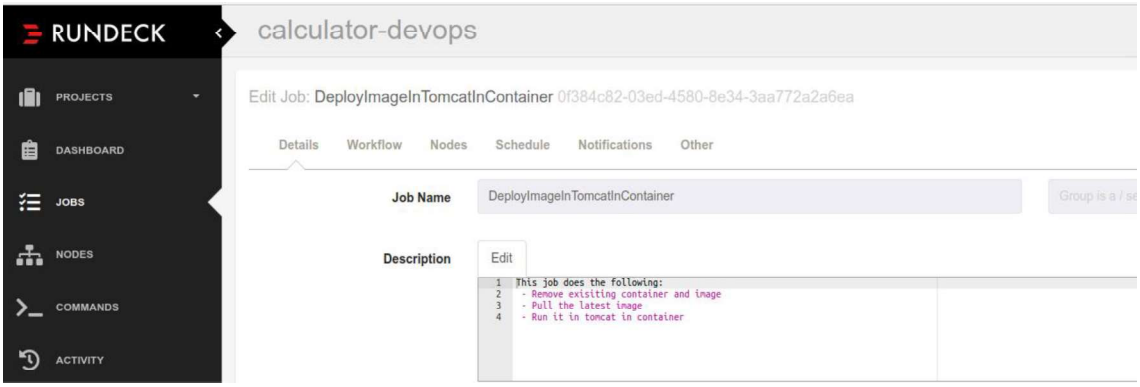
   > sudo service rundeckd start

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

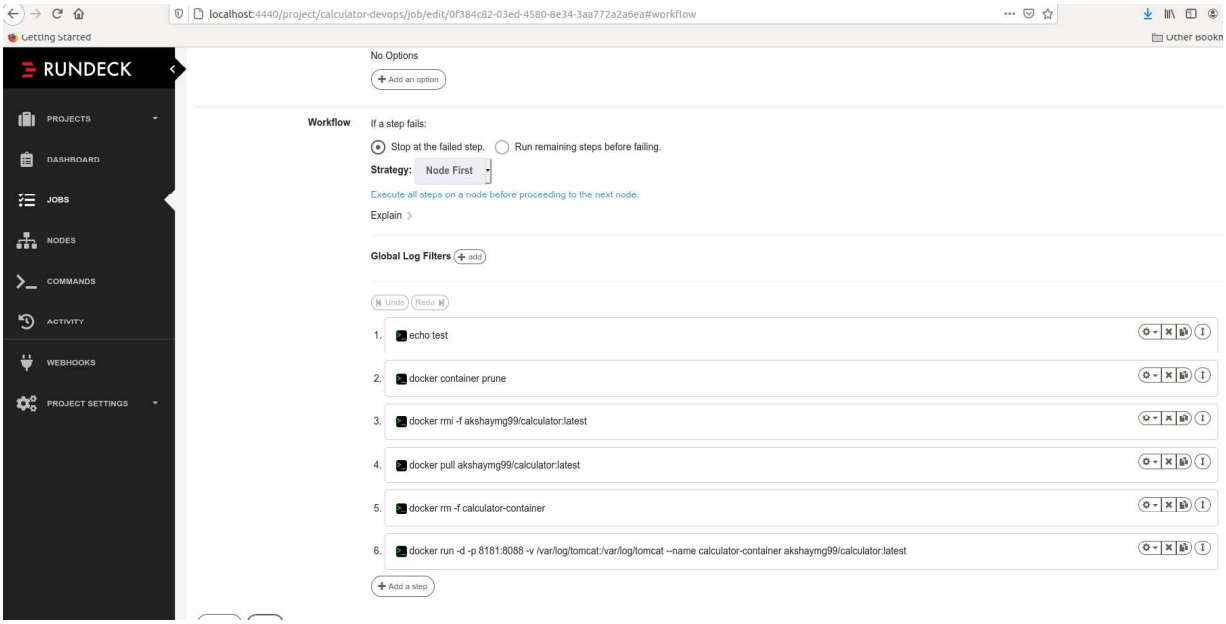
3.5.2 Creating Rundeck Project and Job

We create a Rundeck project and Rundeck job that deploys the generated Docker image locally. To do so, we follow these steps:

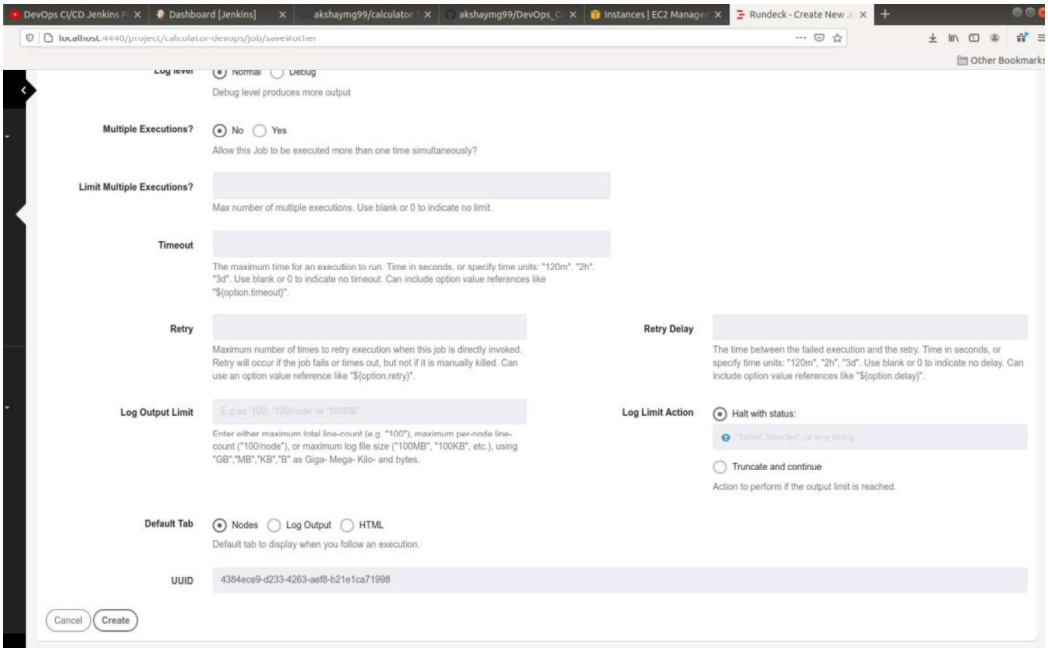
a) Create a new Rundeck job



b) Workflow description



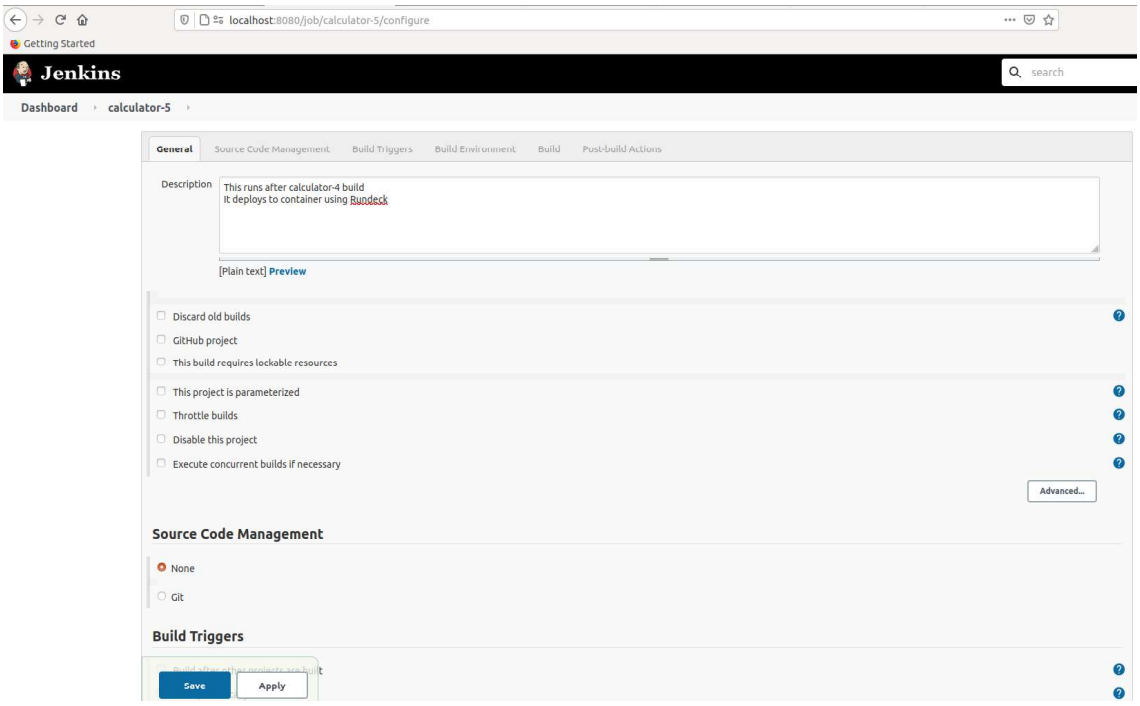
c) Configure Node to execute it locally
d) Note down the UUID of the job to put it in Jenkins



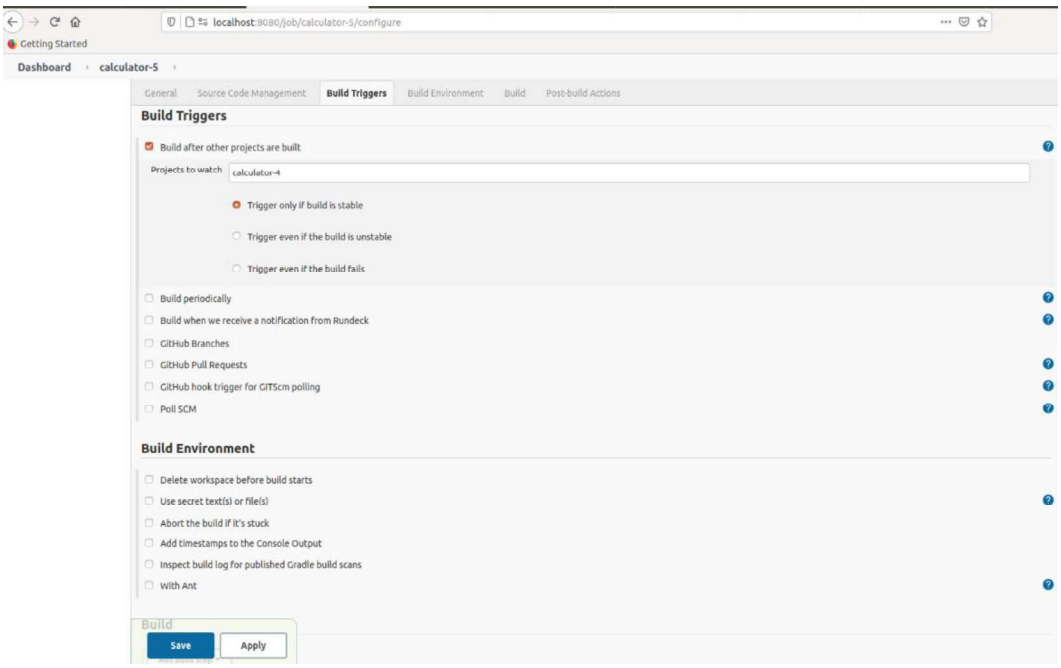
3.5.3 Jenkins Pipeline Job : calculator-deploy-image

We will use this Jenkins job to invoke the Rundeck job that deploys the image after the calculator-docker-push job (here named calculator-4) executes. To do this:

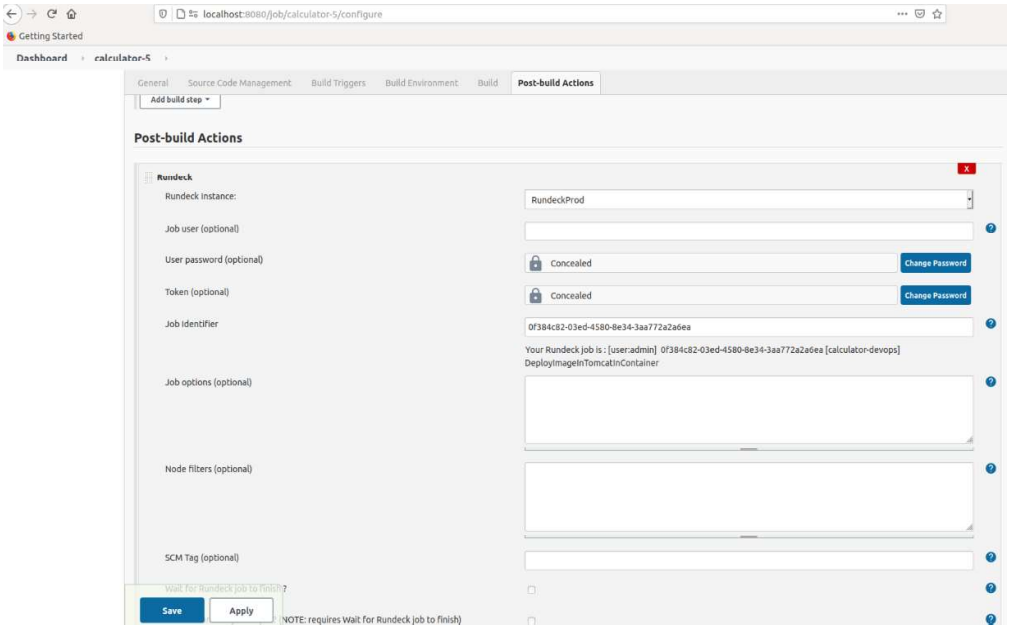
Create a freestyle project in Jenkins with following settings:



Set the build trigger to run after calculator-4 job (which pushes the docker image to hub)



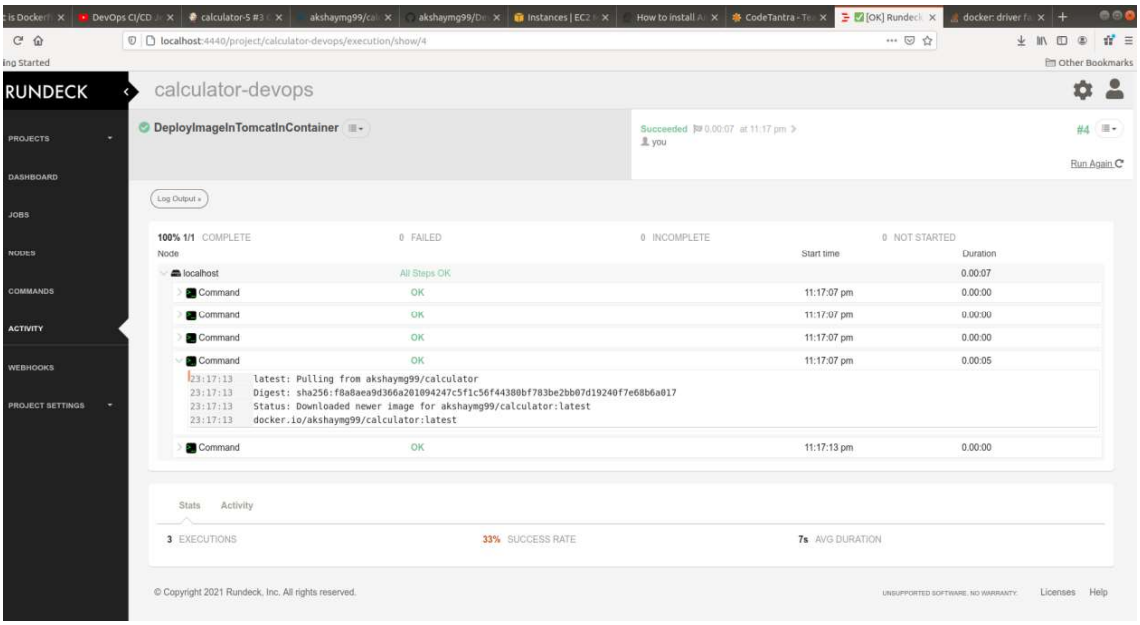
Put the UUID of Rundeck job in Post Build options



Console output

```
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Triggering a new build of calculator-5 #3
Finished: SUCCESS
```

Rundeck dashboard confirming the job to be executed



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 ELK 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.

Here we use the Elastic Stack to perform log analysis in the Calculator project. I use Filebeat to collect logs generated by Jenkins/Tomcat which are then given to Elasticsearch for analysis. Kibana is used to charts to visualize the data diagrammatically.

3.6.1 Installing the Elastic Stack

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
```

```
File  Edit  View  Search  Terminal  Help
● elasticsearch.service - Elasticsearch
   Loaded: loaded (/usr/lib/systemd/system/elasticsearch.service; disabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-03-14 16:15:45 IST; 56s ago
     Docs: https://www.elastic.co
 Main PID: 19490 (java)
    Tasks: 87 (limit: 4915)
   CGroup: /system.slice/elasticsearch.service
           ├─19490 /usr/share/elasticsearch/jdk/bin/java -Xshare:auto -Des.networkaddress.cache.ttl=60 -Des.networkadd
           └─19788 /usr/share/elasticsearch/modules/x-pack-ml/platform/linux-x86_64/bin/controller

Mar 14 16:15:28 akshay-OMEN systemd[1]: Starting Elasticsearch...
Mar 14 16:15:45 akshay-OMEN systemd[1]: Started Elasticsearch.
~
~
~
~
~
```

3. Now, install Kibana using the following commands.

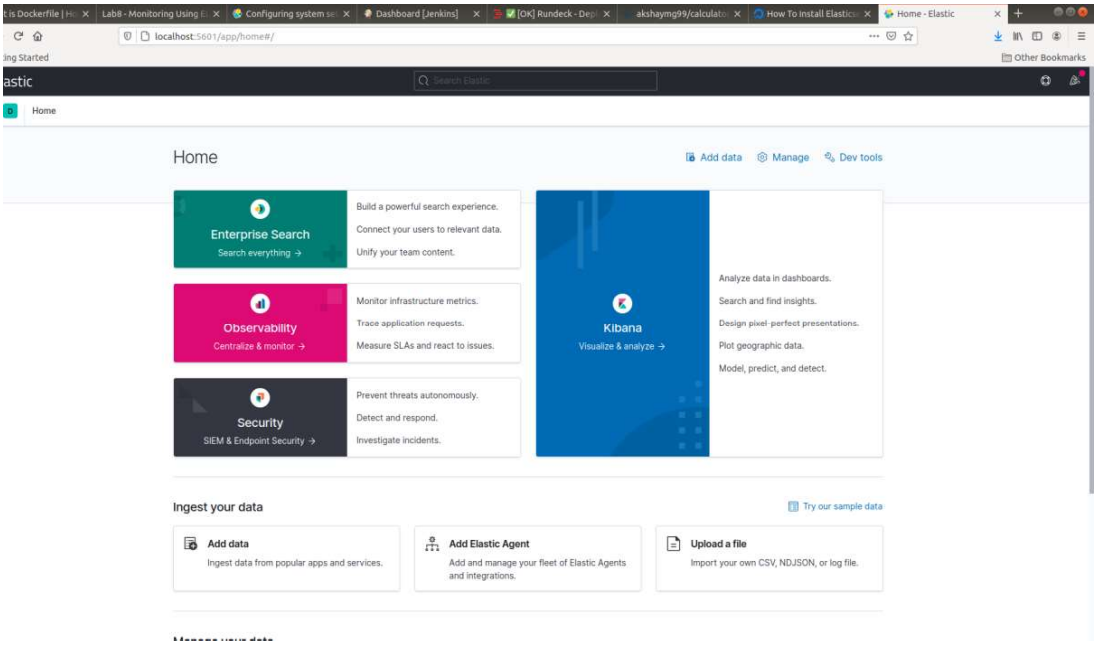
   > sudo apt-get install kibana

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

   server.port: 5601

   elasticsearch.url: http://localhost:9200

4. Kibana runs at http://localhost:5601.



5. Now, install filebeat using the following commands.

   > sudo apt-get install Filebeat

3.6.2 Configuring Filebeat

The path of configuration file for any Beats in Linux is /etc/filebeat/filebeat.yml. To configue Filebeat, edit the yml file

```yaml
# ============================== Filebeat inputs ===============================

filebeat.inputs:

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input specific configurations.

- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /opt/tomcat/logs/access_log.*.log
    - /var/log/jenkins/jenkins.log
    #- c:\programdata\elasticsearch\logs\*

  # Exclude lines. A list of regular expressions to match. It drops the lines that are
  # matching any regular expression from the list.
  #exclude_lines: ['^DBG']
```

```yaml
# ============================== 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 ===================================

# The name of the shipper that publishes the network data. It can be used to group
```

YAML ▾ Tab Width: 8 ▾ Ln 103, Col 23 ▾ INS

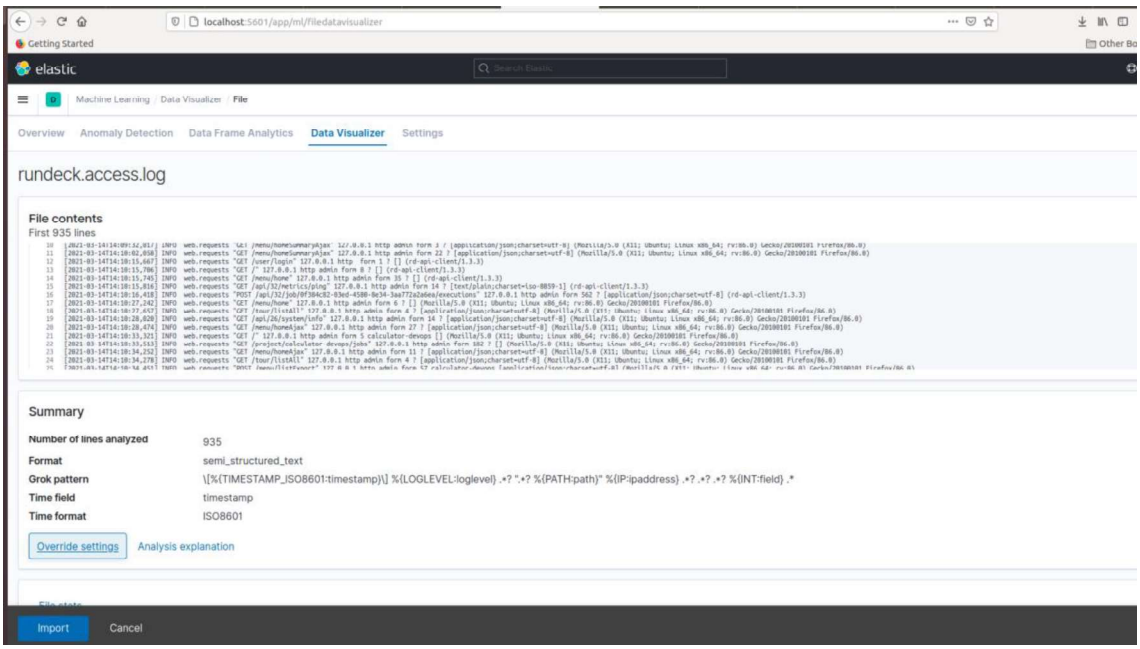
Output configuration

```
Open ▾                    *filebeat.yml                    Save
                          /etc/filebeat

# ============================== 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.
#cloud.id:

# The cloud.auth setting overwrites the `output.elasticsearch.username` and
# `output.elasticsearch.password` settings. The format is `<user>:<pass>`.
#cloud.auth:

# ================================= Outputs =================================

# Configure what output to use when sending the data collected by the beat.

# ---------------------------- Elasticsearch Output ----------------------------
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]
  index: "filebeat-%{+yyyy.MM.dd}"
setup.template:
  name:  'filebeat'
  pattern: 'filebeat-*'
  enabled: false
  # Protocol - either `http` (default) or `https`.
  #protocol: "https"

  # Authentication credentials - either API key or username/password.
  #api_key: "id:api_key"
  #username: "elastic"
  #password: "changeme"

# ------------------------------ Logstash Output -------------------------------
#output.logstash:
  # The Logstash hosts
  #hosts: ["http://localhost:5044"]

  # Optional SSL. By default is off.
  # List of root certificates for HTTPS server verifications
  #ssl.certificate_authorities: ["/etc/pki/root/ca.pem"]

  # Certificate for SSL client authentication
  #ssl.certificate: "/etc/pki/client/cert.pem"

  # Client Certificate Key
  #ssl.key: "/etc/pki/client/cert.key"

# ================================= Processors =================================
processors:
YAML ▾   Tab Width: 8 ▾   Ln 212, Col 40 ▾   INS
```

Restart Filebeat after editing the configuration file.

> sudo service filebeat restart

Visualization of Rundeck logs in Kibana by uploading its log file



4. Conclusion

In this project, we automated the entire SDLC using DevOps toolchain. This makes the development team and operations team work productively 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 companies to quickly build, test and deploy new versions of their products.

5. References

1) https://www.jenkins.io/doc/

2) https://docs.docker.com/

3) https://junit.org/junit5/docs/current/user-guide/

4) https://git-scm.com/docs/git

5) https://docs.rundeck.com/docs/

6) https://maven.apache.org/guides/index.html