Calculator (Mini-Project)

Seethamraju Purvaj

March 17, 2021

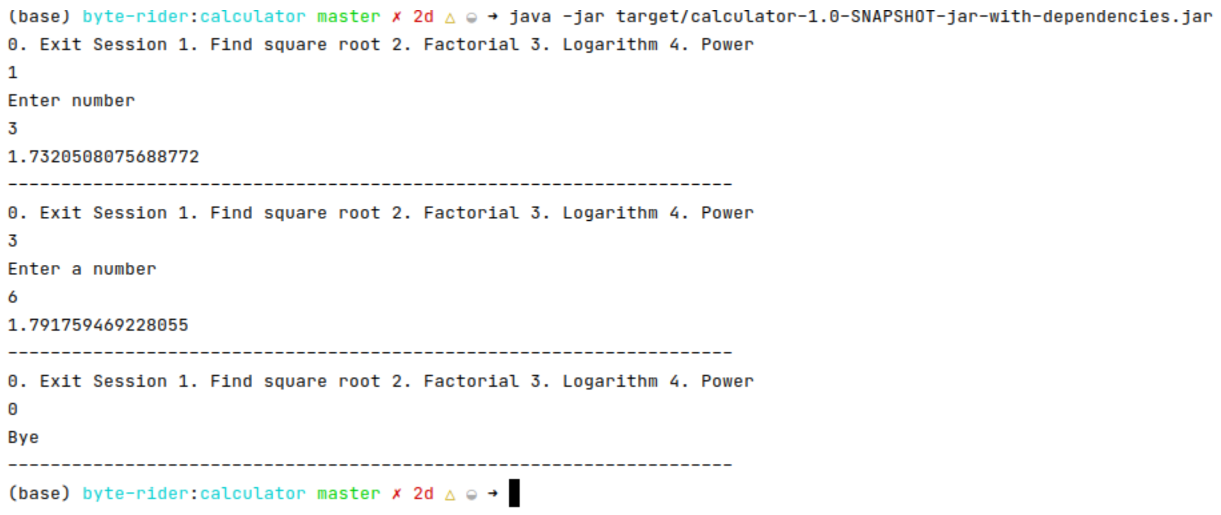
Contents

1 Introduction

This is a simple calculator application written in Java. The speciality of this project is the various DevOps concepts used in the deployment of this application. This application has 4 functionalities as given in the document

- Power function
- Factorial
- Logarithm
- Square root function

2 Application Code



```
(base) byte-rider:calculator master ✗ 2d △ ◒ ➜ java -jar target/calculator-1.0-SNAPSHOT-jar-with-dependencies.jar
0. Exit Session 1. Find square root 2. Factorial 3. Logarithm 4. Power
1
Enter number
3
1.7320508075688772
----------------------------------------------------------------------
0. Exit Session 1. Find square root 2. Factorial 3. Logarithm 4. Power
3
Enter a number
6
1.791759469228055
----------------------------------------------------------------------
0. Exit Session 1. Find square root 2. Factorial 3. Logarithm 4. Power
0
Bye
----------------------------------------------------------------------
(base) byte-rider:calculator master ✗ 2d △ ◒ ➜ █
```

The application code consist of a class *Calculator* and a main class *Driver* that uses calculator object to make an interactive terminal script. There are a set of test suites for each function written in Junit.

Repo Link http://github.com/seethamraju/calculator
Docker Repo https://hub.docker.com/r/byterider/cal
Docker Repo web https://hub.docker.com/r/byterider/calweb

2.1 How to run the Application

To run the application you must have docker installed in your machine. Please follow the steps in this link according to your environment. Docker Installation

Once done run the following command to pull the docker image into your local machine

```
$ docker pull byterider/cal:<tag-number>
```

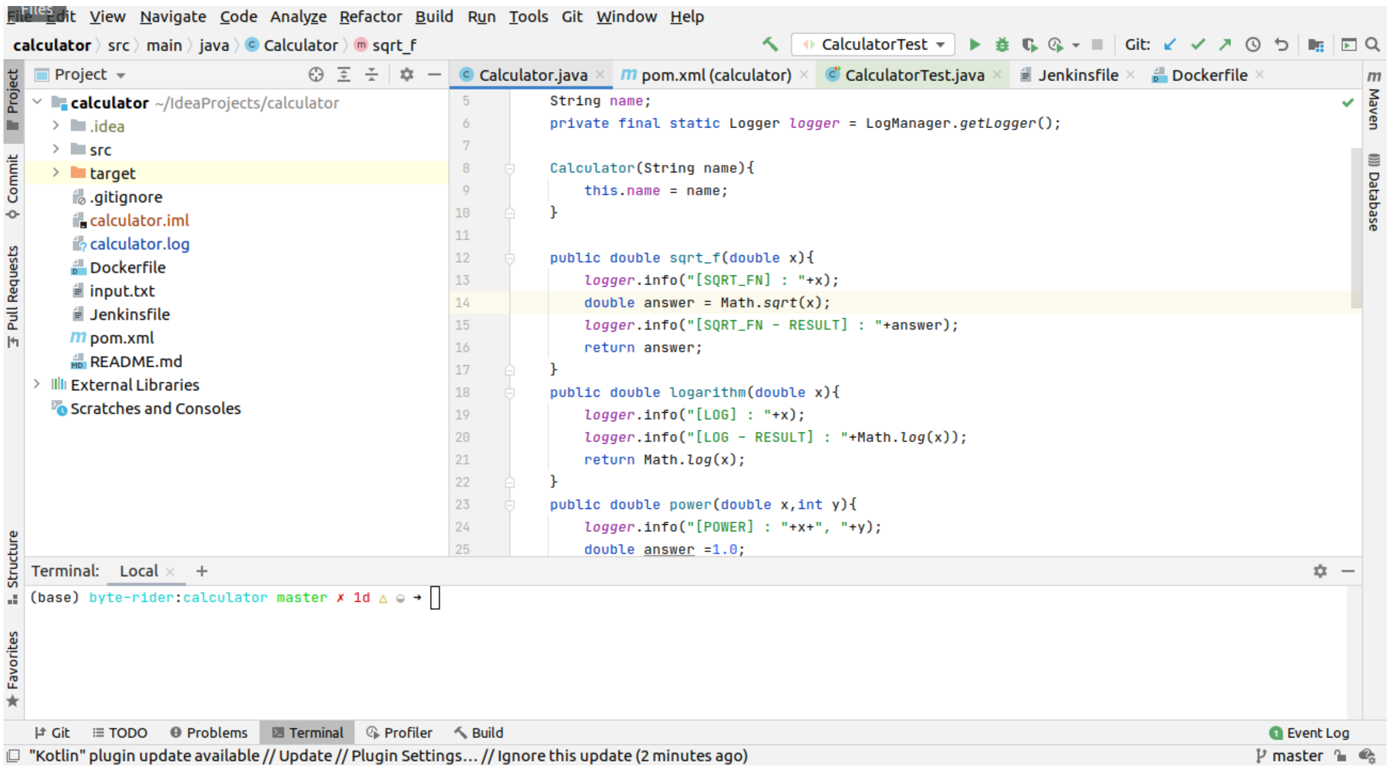
For using it pull the latest tag.

Once pulled run it using the following command

```
$ docker run -it byterider/cal:latest
```

3 Development Environment

The development environment chosen is IntelliJ which is a very well known IDE for Java development. All the setup discussed below is also according to the IntelliJ environment. We had to configure the git credentials in intelliJ.



```java
    String name;
    private final static Logger logger = LogManager.getLogger();

    Calculator(String name){
        this.name = name;
    }

    public double sqrt_f(double x){
        logger.info("[SQRT_FN] : "+x);
        double answer = Math.sqrt(x);
        logger.info("[SQRT_FN - RESULT] : "+answer);
        return answer;
    }
    public double logarithm(double x){
        logger.info("[LOG] : "+x);
        logger.info("[LOG - RESULT] : "+Math.log(x));
        return Math.log(x);
    }
    public double power(double x,int y){
        logger.info("[POWER] : "+x+", "+y);
        double answer =1.0;
```

```
(base) byte-rider:calculator master ✗ 1d △ ⊖ →
```

"Kotlin" plugin update available // Update // Plugin Settings... // Ignore this update (2 minutes ago)

Figure 1: A Sample Picture of IntellIJ workspace

4 Git

The VCS(version control system) used for the project is Git.

4.1 What is Git?

Git is a distributed version control system. It is helpful when multiple people are working on a single project's features. It helps each of the person to stay updated with the same code. Github is a hosting service based on git that enables a person to create repositories and follow them using git features.

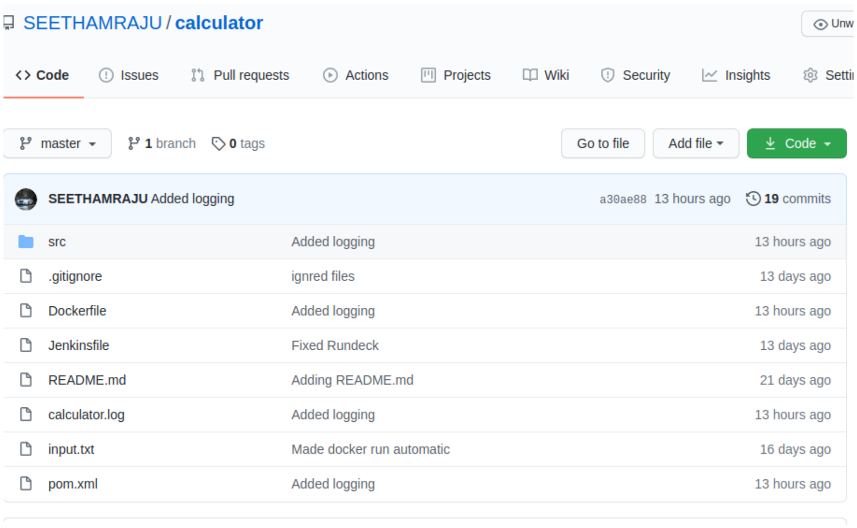
4.2 Setup

A repository is created in Github called **SEETHAMRAJU/calculator** which will from now on contain the application code. The IntelliJ has builtin integration for git hence few sequence of buttons pressed will eventually push the code to github. Most of the work is done in the ***master*** branch. In the local system we have to configure the git credentials using commands

```
$ git config —global user.password <password>
$ git config —global user.username <username>
```

Few other common commands for git operations are as below

```
$ git init # initializae git repo
$ git add . # add the existing files to git repo
$ git commit –m "" # commit the changes wiht the message
$ git push # Push the changed to the remote repo
```


SEETHAMRAJU / calculator

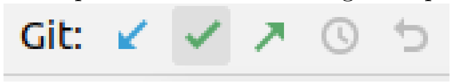
Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Setti

master · 1 branch · 0 tags · Go to file · Add file · Code

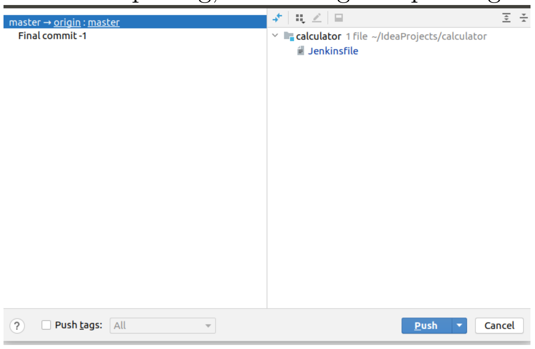
SEETHAMRAJU Added logging · a30ae88 13 hours ago · 19 commits

src	Added logging	13 hours ago
.gitignore	ignred files	13 days ago
Dockerfile	Added logging	13 hours ago
Jenkinsfile	Fixed Rundeck	13 days ago
README.md	Adding README.md	21 days ago
calculator.log	Added logging	13 hours ago
input.txt	Made docker run automatic	16 days ago
pom.xml	Added logging	13 hours ago

After creating the github repo , it is integrated with the intelliJ to commit and push all the code using GUI. Some pictures of the IntelliJ github plugin where all the git actions are done using simple GUI.



This is for pulling, commiting and pushing respectively.


master → origin : master
Final commit -1

calculator 1 file ~/IdeaProjects/calculator
Jenkinsfile

This picture is the popup window for pushing to the speicfic branch.

5 Maven

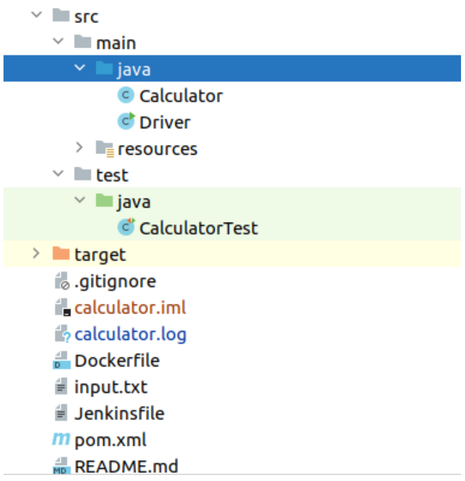
Inorder to make the process of building and testing and installing the java based application Maven was used.

5.1 What is Maven?

Maven is a software project management tool. For example, for a given java project ew might need a lot of test suites, dependencies and commands to set it up. However when this is done using maven adding simple lines to a pom.xml file can actually install the dependencies. This maven project contains a configuration file (*pom.xml*). This file is the basic file that maven uses to get all information. It is then used to 'build' projects and then test them using simple commands. It has many stages int the build life cycle some of which include verify, compile etc. The main parts that we will be using are build, test, install and occasionally clean.

5.2 Setup

For the maven we just needed to install a plugin in IntelliJ and also install maven in the local system. Once this was done we could create a new Maven project that contained the directory structure as in the image.



```
src
  main
    java
      Calculator
      Driver
    resources
  test
    java
      CalculatorTest
target
.gitignore
calculator.iml
calculator.log
Dockerfile
input.txt
Jenkinsfile
pom.xml
README.md
```

Here the **src/** folder contains the actual java code and the resources that the code uses(i.e. any config files). The **test** folder has the test cases written in JUnit.

The pom.xml file is the main file that takes care of the dependencies. This is the command for installing maven using the following command.

```
$ sudo apt install maven
```

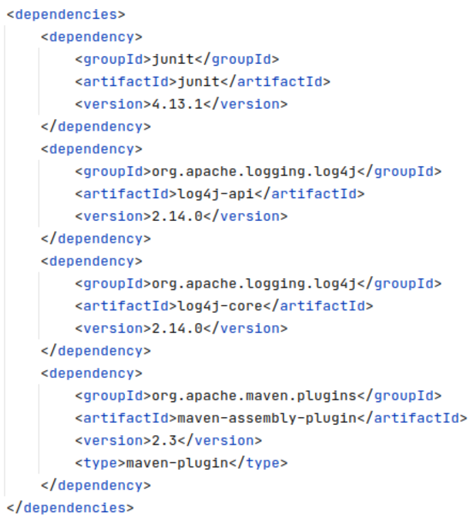
Since my machine had installed Java8 there wasn't much configuration needed. After this once we created the project we use the commands as below to install and clean.

```
$ mvn clean
$ mvn build
$ mvn test
$ mvn install
```

Here each command cleans the workspace, builds the application , runs the tests associated with application respectively. the **install** command can be used to perform the 2 and 3rd task collectively.

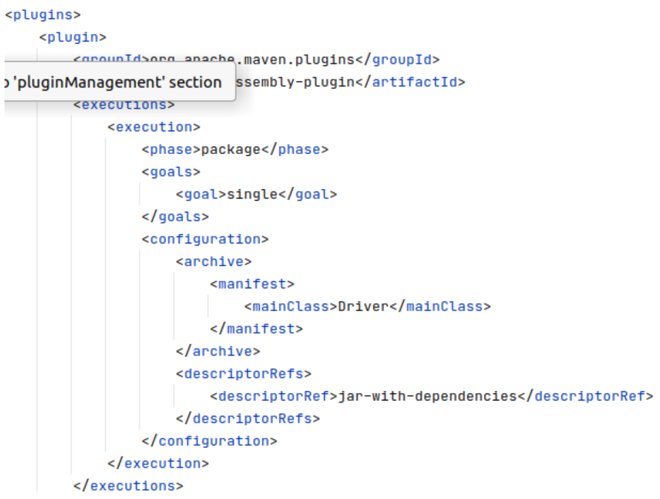
5.3 A walk through of the pom.xml

Let us first go through the pom.xml file of this project.



```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.14.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.14.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>2.3</version>
        <type>maven-plugin</type>
    </dependency>
</dependencies>
```

(a) Dependencies



```
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
'pluginManagement' section   ssembly-plugin</artifactId>
        <executions>
            <execution>
                <phase>package</phase>
                <goals>
                    <goal>single</goal>
                </goals>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>Driver</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </execution>
        </executions>
```

(b) Build Features

Here the dependencies include Junit, log4J and few assembly plugins. For log4j to be compatible with the jar file we had to change the build configuration to build with dependencies. We can also set the build settings like we did. This enables us to make the build according to our requirement.

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</groupId>
```

```
    <artifactId>calculator</artifactId>
    <version>1.0-SNAPSHOT</version>
```

These are the basic properties of the java project like version, artifactId.

6 J Unit

For the unit tests Junit was used.

6.1 What is Junit?

JUnit is a unit testing framework for the Java programming language. Especially for Java and and applications which require robust unit testing Junit can be very helpful.

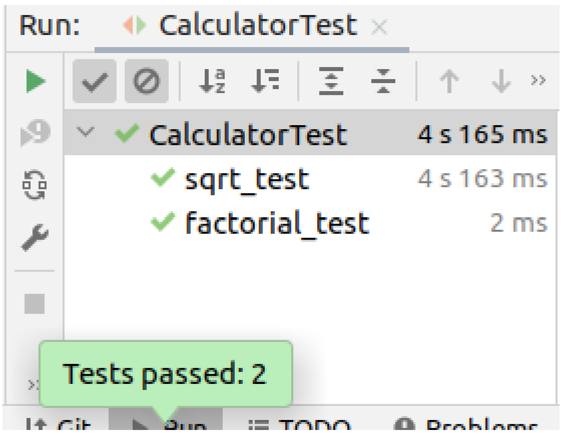
6.2 Setup

Since we are using maven we can directly add the junit dependency in the ***pom.xml*** file to install and manage the junit resources. Once done, we create a Junit test case for every function and run each of them to see how they work. In Junit a test suite is created for each functionality. For this we follow as below
right click class ⟶ create Junit testcase ⟶ Write the test cases

```
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.1</version>
    </dependency>
```

In each test case we also specify the expected value of the testcase and then wrap an assert statement around it. In our case we also have double data type and hence we also use the concept of delta where delta refers to the max absolute difference between the expected value and returned value.

```
assertEquals(expectedValue, returnedValue)
assertEquals(expectedValue, returnedValue, delta)
```



```
Run: CalculatorTest
CalculatorTest    4 s 165 ms
  sqrt_test       4 s 163 ms
  factorial_test  2 ms
Tests passed: 2
```

6.3 Test Suites

For each function most of the boundary conditions were checked in the test cases.

True Positive: Case when the code is perfect and the test cases pass.
True Negative: Case when the code is perfect but still some cases don't pass.
False Positive: Case there are errors which the test cases can't find and they pass.
False Negative: When the code is broken and the tests don't pass.

Since the app we developed has simple functionality we don't have most of the above cases. However each functioanlity has a set of tests in *src/test/java/*.

7 Dockerization

7.1 What is Docker and why?

Docker is a set of platform as a service products that deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files. It generally helps to ship applications in a free manner without worrying about their dependencies.

7.2 Initial Setup

We had to install a docker in local machine, and also create a docker registry and configure the credentials with the loca machine. Any project must containt a **Dockerfile** which contains the 'recipe' to build a docker image. The command to run to build a docker image is

```
$ docker build -t <name> .
$ docker run <imagename>:<tagname>
```

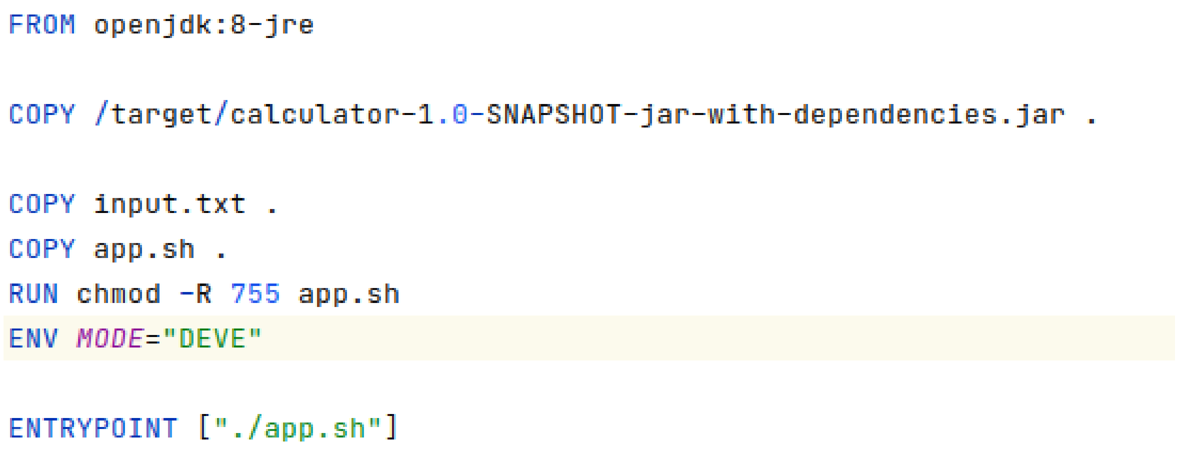
Here the first command builds the docker image. It searches for the file named **Dockerfile** in the current directory, and then build the image according to the **Dockerfile**. The next command it to actually run an image where the arugments. To pull an image and push an image to the dockerhub registry we just need to do the following commands,

```
$ docker push <imagename>:<tag>
$ docker pull <imagename>:<tag>
```

7.3 Project Dockerization

For the current project there were two ways we could create the docker image. First, by just running the *.jar* file inside the docker. Another way is to compile the *src* foledr in a docker container to get the *.jar* file and then put run it. The latter takes a lot of memory and the image is heavy where as the former is a light weight image. Hence we followed a simple procedure of copying the JAR file and running it.

However one problem is in order to automate the docker running in rundeck we must somehow bypass the inputs that the application requests for. TO tackle this we used the concept of **ENV** varaiables in dockers.



```
FROM openjdk:8-jre


COPY /target/calculator-1.0-SNAPSHOT-jar-with-dependencies.jar .


COPY input.txt .
COPY app.sh .
RUN chmod -R 755 app.sh
ENV MODE="DEVE"

ENTRYPOINT ["./app.sh"]
```

In this we copy the *jar* file,*input.txt* which is the file that contains input to the app and then *app.sh* which basically runs the jar file based on the environment. The app.sh is as below, based on the enviroment variable **MODE** it can start the instances.

```
  if [ $MODE = "DEVE" ]
 then
     java -jar calculator-1.0-SNAPSHOT-jar-with-dependencies.jar < input.txt
else
     java -jar calculator-1.0-SNAPSHOT-jar-with-dependencies.jar
fi
```

This app.sh is actually the **Entrypoint** of the Docker container which means that the docker container is defined with a particular executable. (*app.sh* in this case). The app.sh as seen above checks the value of $MODE which can be "DEVE" or "PROD". based on the MODE the specific commands is run. The variable MODE is called **environment variable**. The below are the procedures in which we give environemnt variable during **run time** and additional parametrs.

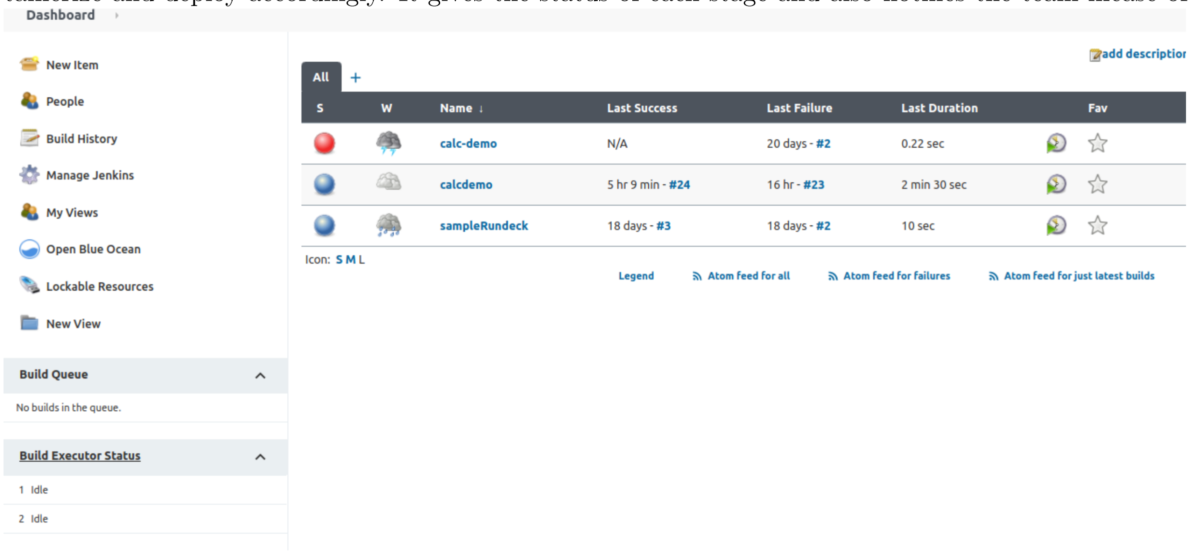
```
$ docker run —e MODE="DEVE" byterider/cal
#or
$ docker run —e MODE="PROD" —it byterider/cal
```

8 Jenkins

This is used for CI. It helps in managing the testing, dockerizing and deploying of the Java application.

8.1 What is Jenkins?

Jenkins is a CI/ CD pipeline that automates the workflow of a project thus helping the developers and their teams to stress on quality. In simple words it helps in automating the stages of a project, build, test, containerize and deploy accordingly. It gives the status of each stage and also notifies the team incase of failures.


Dashboard

New Item
People
Build History
Manage Jenkins
My Views
Open Blue Ocean
Lockable Resources
New View

add description

All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
🔴		calc-demo	N/A	20 days - #2	0.22 sec	☆
🔵		calcdemo	5 hr 9 min - #24	16 hr - #23	2 min 30 sec	☆
🔵		sampleRundeck	18 days - #3	18 days - #2	10 sec	☆

Icon: S M L

Legend · Atom feed for all · Atom feed for failures · Atom feed for just latest builds

Build Queue

No builds in the queue.

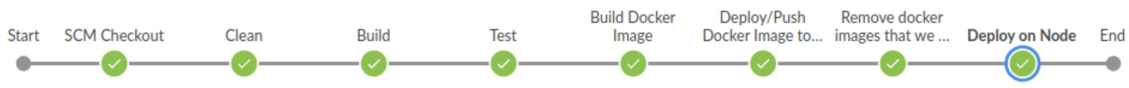
Build Executor Status

1 Idle
2 Idle

8.2 Setup

Setting up jenkins was a series that had to be followed properly. We can install directly using the *apt-get* in ubuntu. After that we create a user in jenkins and then open a new project. For the current project we used a pipeline project to setup the pipeline of CI and CD. WE have used various SCM and Docker registries and hence there was a process to setup the credentials. Setting up Docker hub registry. This involves configuring the docker account so that all "steps" requiring dockerhub credentials will use them. Setting up rundeck credentials. This involves configuring the rundeck credentials to run the jobs in the pipeline. Each part of the pipeline will be explained later.

Few **plugins** were also added along with Jenkins to help handle the maven dependent project. We installed Maven pluing and ***Blue Ocean*** as well. **Blue Ocean** is a GUI based tool which gives a rich experience compared to Jenkins.These plugins can be installed through the Jenkins GUI. *Manage Jenkins* $\longrightarrow$ *Manage Plugins* $\longrightarrow$ Available



The main part of the Jenkins is a **Jenkinsfile** which contains all the pipeline steps and the required tasks to be done.

```
$ wget —q —O — http://pkg.jenkins—ci.org/debian/jenkins—ci.org.key | sudo apt—key add —
$ sudo sh —c 'echo␣deb␣https://pkg.jenkins.io/debian␣binary/␣>␣/etc/apt/sources.list.d/jen
$  sudo apt—get install jenkins
```

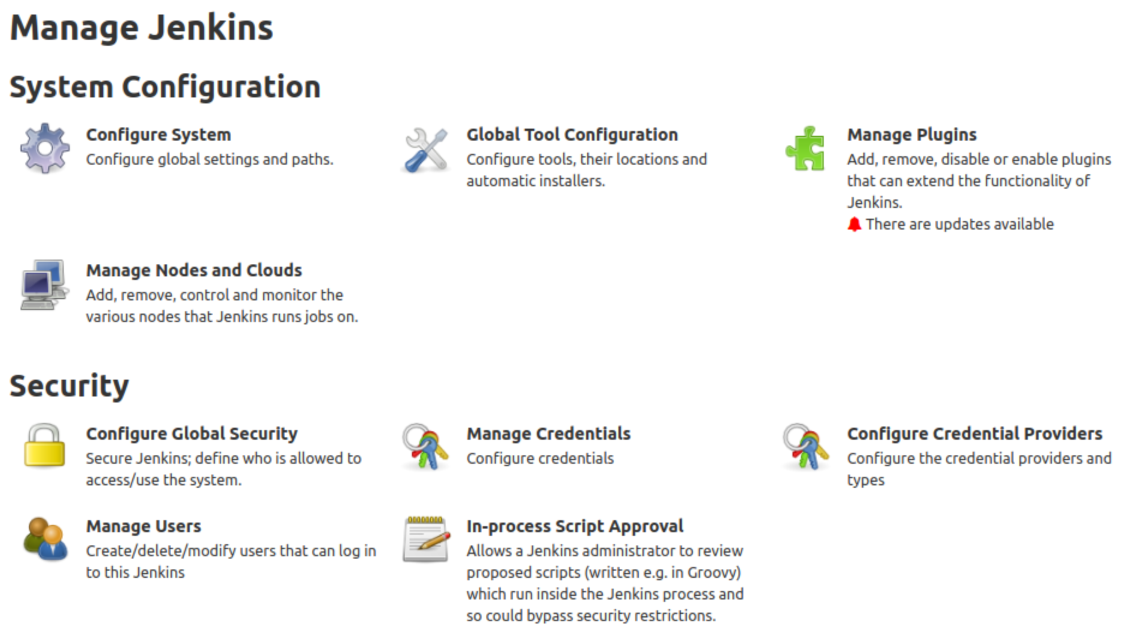
Above command is used for installing Jenkins.

8.3 Configuration

For the jenkins pipeline to run smoothly we had to configure the Jenkins Tool with multiple accounts. We first go and create a new pipeline.

8.3.1 Github Configuration

Each project had to be configured to a SCM repo (Git in our case). We could set the jenkins project to keep polling the SCM to check for new commits. So in simple words, the Jenkins checks the github repo for new commmits and then incase there are any new changes it triggers the build. more info about how its done is given below.



Manage Jenkins

System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Manage Credentials
Configure credentials

Configure Credential Providers
Configure the credential providers and types

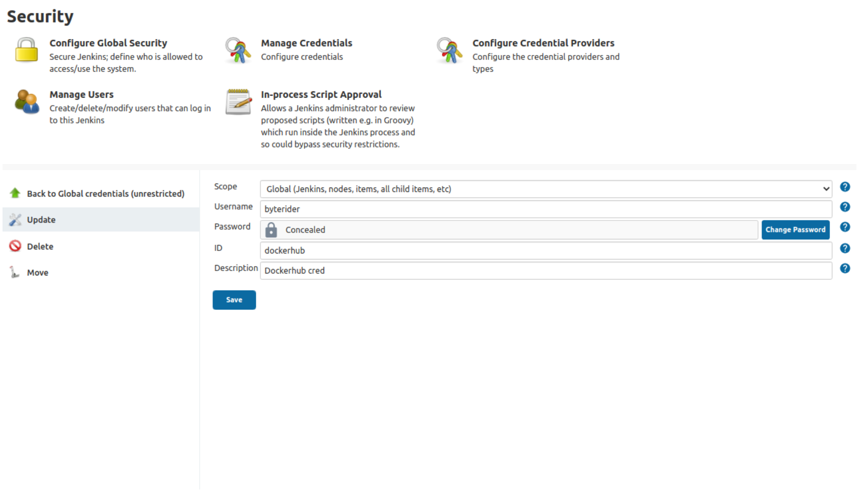
Manage Users
Create/delete/modify users that can log in to this Jenkins

In-process Script Approval
Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.

The SS is about the Jenkins Configuration management.

8.3.2 Docker Configuration

We had to save dockerhub registry credentials so that the Jenkins pipeline could directly push a docker image to the docker registry.



Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Manage Credentials
Configure credentials

Configure Credential Providers
Configure the credential providers and types

Manage Users
Create/delete/modify users that can log in to this Jenkins

In-process Script Approval
Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.

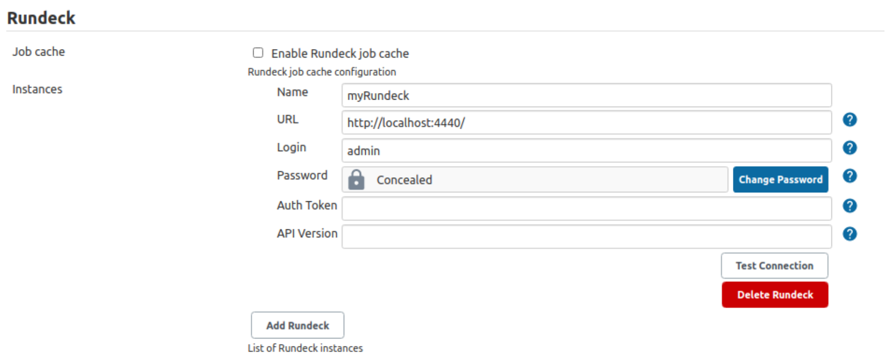
Back to Global credentials (unrestricted)
Update
Delete
Move

Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	byterider
Password	Concealed
ID	dockerhub
Description	Dockerhub cred

Here the dockerhub credentials were stored and the variable that stores these and which helps us refer these is **dockerhub**. You can see this in *Jenkins* line 4.

8.3.3 Rundeck Configuration

To trigger a rundeck Job we had to configure the rundeck tool with jenkins and then specify the required *Job ID* inside the Jenkinsfile. This would be sufficient to run the rundeck job. However while running it's important for Rundeck tool to be running.



Rundeck

Job cache: Enable Rundeck job cache

Rundeck job cache configuration

Instances

Name	myRundeck
URL	http://localhost:4440/
Login	admin
Password	Concealed
Auth Token	
API Version	

Here we configured the rundeck tool and specified the location where it is running and the Rundeck credentials and the name which will be used in the Pipeline file (*line 58*).

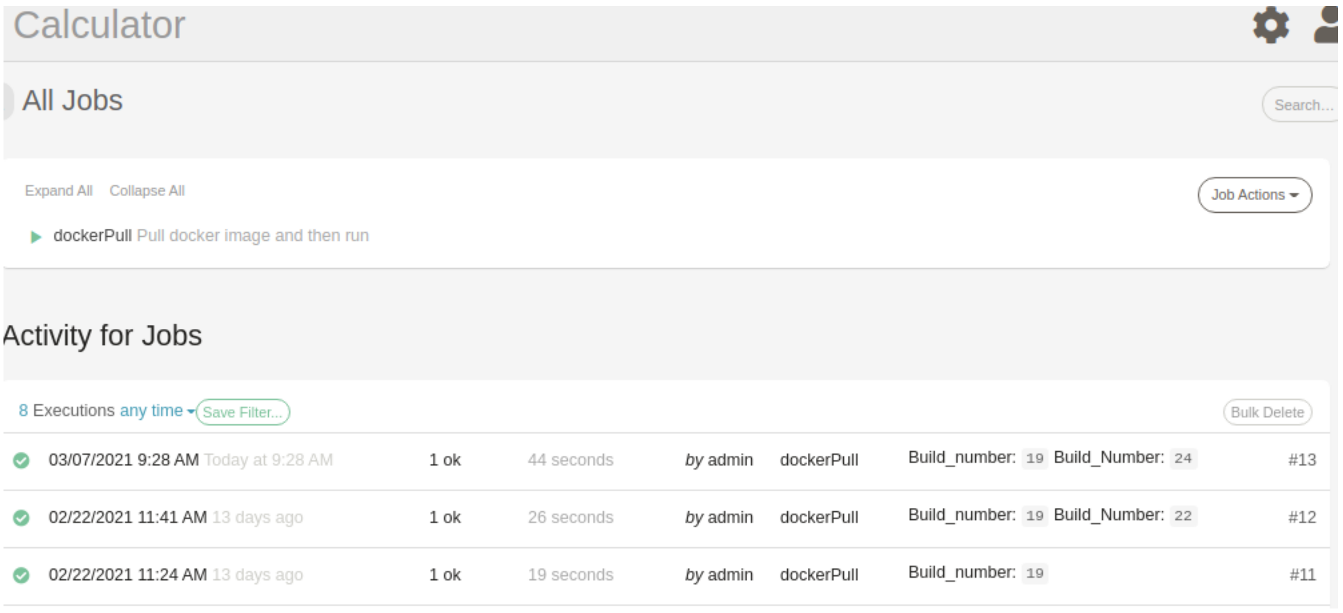
9 Rundeck

Rundeck is a tool used for deployment.

9.1 What is Rundeck?

Rundeck is an opensource tool for automating and addressing complexities in deployment. This generally helps in continuous deployment. . It provides web console, CLI tools, and a Web API.

9.2 Setup



Calculator

All Jobs

dockerPull Pull docker image and then run

Activity for Jobs

8 Executions any time

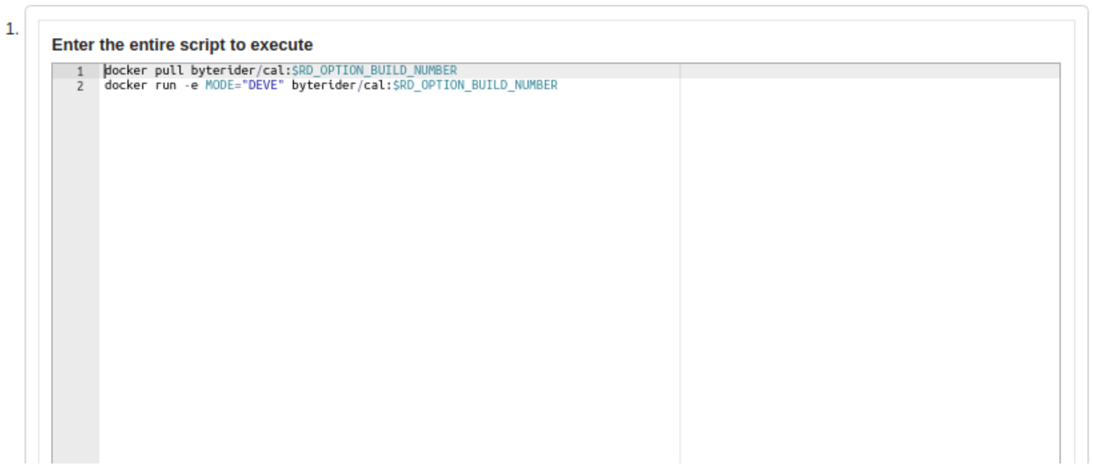
03/07/2021 9:28 AM Today at 9:28 AM	1 ok	44 seconds	by admin	dockerPull	Build_number: 19 Build_Number: 24	#13
02/22/2021 11:41 AM 13 days ago	1 ok	26 seconds	by admin	dockerPull	Build_number: 19 Build_Number: 22	#12
02/22/2021 11:24 AM 13 days ago	1 ok	19 seconds	by admin	dockerPull	Build_number: 19	#11

Rundeck in general was installed in normal debian style. Once done we had to loign using the credentials ***admin*** which is both password and username. Then we created a Job which basically has a lot of options to create workflow i.e. which are basically tasks that needs to be performed (Generally bash commands). We can also select a set of "nodes" i.e. systems to run the commands. This helps in automatic deployment.

9.3 Project Job

For this project we created a simple job that pulls a docker image and then just runs it. This job was then triggered by the Jenkins pipeline. This Job had the following script and the following options.

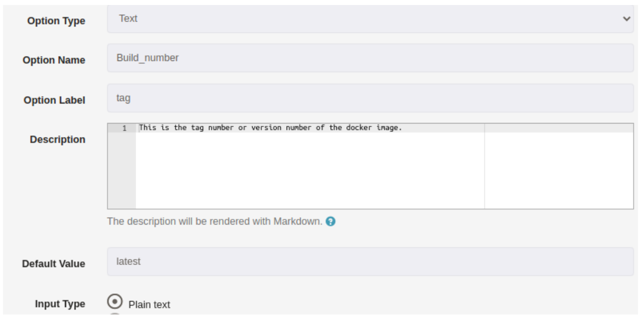
Options are basically the arguments that we can dynamically give to a rundeck Job. This was used to provide the tag for the docker image as each time a docker image was pushed to the registry with a new tag (which was the **BUILD NUMBER** of jenkins.) This was given as an input in the Jenkins. Incase we don't specify the option then we also set the default value of the argument. **Options** help in changing few parameters of the job dynamically.


1.

Enter the entire script to execute

```
docker pull byterider/cal:$RD_OPTION_BUILD_NUMBER
docker run -e MODE="DEVE" byterider/cal:$RD_OPTION_BUILD_NUMBER
```

Figure 3: SS of the Job for this project



Option Type	Text
Option Name	Build_number
Option Label	tag
Description	This is the tag number or version number of the docker image.
	The description will be rendered with Markdown.
Default Value	latest
Input Type	[✓ Plain text]

Figure 4: SS of the options for this project

10 Continuous Logging and Monitoring

This is basically using the ELK stack to analyse the logs data from the application. We add the **log4j** library to add logging feature to the application. After adding we get a *log* file after running. Now this log file can be actually fed to the ELK stack so that it can help analyze the data.

10.1 Setup

In this project ***Elasticsearch cloud*** was used due to the constraints on CPU usage. Steps to configure the Elasticsearch cloud is given below.

1. Create a new ELK cloud account
2. Create a deployment and select required options like stack, Memory etc.
3. Now copy the **Cloud ID** , **Username** and **password** into the logstash.yml file near the place where we have to configure the cloud.
4. next update the *.conf* file accordingly and run the logstash executable. Once done we can go to the elk cloud website and go to kibana and then use the logs for analyizing by giving the index name.

One important aspect of a logging mechanism is the logging syntax and the filters for logstash config file.

10.2 Configuration Files

logstash.yml This is the file where we have to use the Cloud credentials to let the logstash program know that we have to send the logs to cloud.
Logstash Config File This is specific to the program that we are using. This will have different plugins which help the logs to be filtered and transferrd to teh ELK stack.
There are three parts to the file , *input, filter, output.*
input :

```
  input {
file{
        path => "/home/byte-rider/IdeaProjects/calculator/calculator.log"
        start_position => "beginning"
 }
}
```

The above snippet specifies the input source for the logstash application. Here it is the log file placed in the given location.

```
filter{
        grok{
             match => [
                  "message", "%{HTTPDATE:timestamp_string} \[%{GREEDYDATA:thread}\] \[%{LOGL
             ]
        }
        date {
                  match => [
                            "timestamp_string", "dd/MMM/YYYY:HH:mm:ss SSS"
                  ]
        }
        mutate {
                  remove_field => [timestamp_string]
        }

}
```

Here the code represents the way to filtering the logs so that they can be used in the Kibana.

```
output {
  elasticsearch {
    hosts => ["<link>"]
    index => "calculator"
    user => "elastic"
    password => "<password>"
  }
}
```

This snippet is basically tells, where the processed logs must be sent to. Here we are given the cloudID, hosts, passwords since we are using the cloud ELK deployment.

10.3 Log4j

Log4j is an interface for logging in Java. It provides simple API calls for logging into a log file. Installing Log4j is just making few changes to the *pom.xml*. We also need to specify few properties for the library to understand how it should log.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <File name="FileAppender" fileName="calculator.log" immediateFlush="false" append="true">
          <PatternLayout pattern="%d{dd/MMM/yyy:HH:mm:ss SSS} [%F] [%level] %logger{36} %msg%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="FileAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

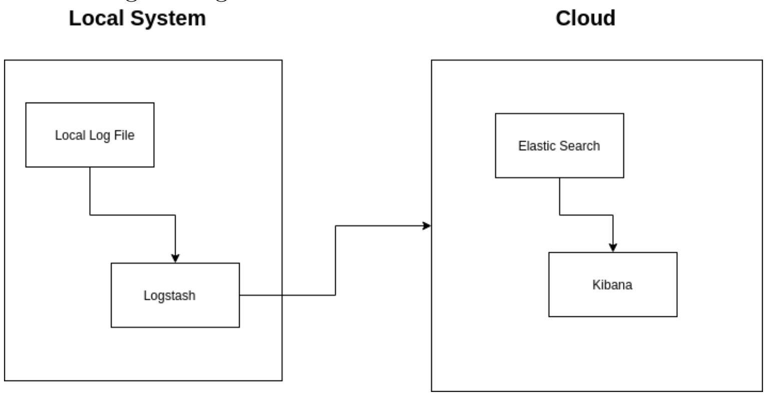
Here we are specifying the pattern in which logs must be written and the mode (i.e. File) and location of the file. We are also specifying that the logs must be appended to a file. We can also tweak the code to get logs on different sources as well. the option **immediateFlush** decides if the logs have to be flushed after the execution of the project is complete or they have to be written as and when they come. **Append** ensures that the logs are accumulated each time rather than being overwritten for every execution. Few examples of logs are:

```
08/Mar/2021:23:05:16 842 [Calculator.java] [INFO] Calculator [LOG] : 4.0
08/Mar/2021:23:05:16 844 [Calculator.java] [INFO] Calculator [LOG - RESULT] : 1.3862943611198906
```

The above statement gives the following information : timestamp, thread, level of logging, class name, action, and finally the numerical result. Once we get these logs these are then fed to the ELK stack via the logstash tool.

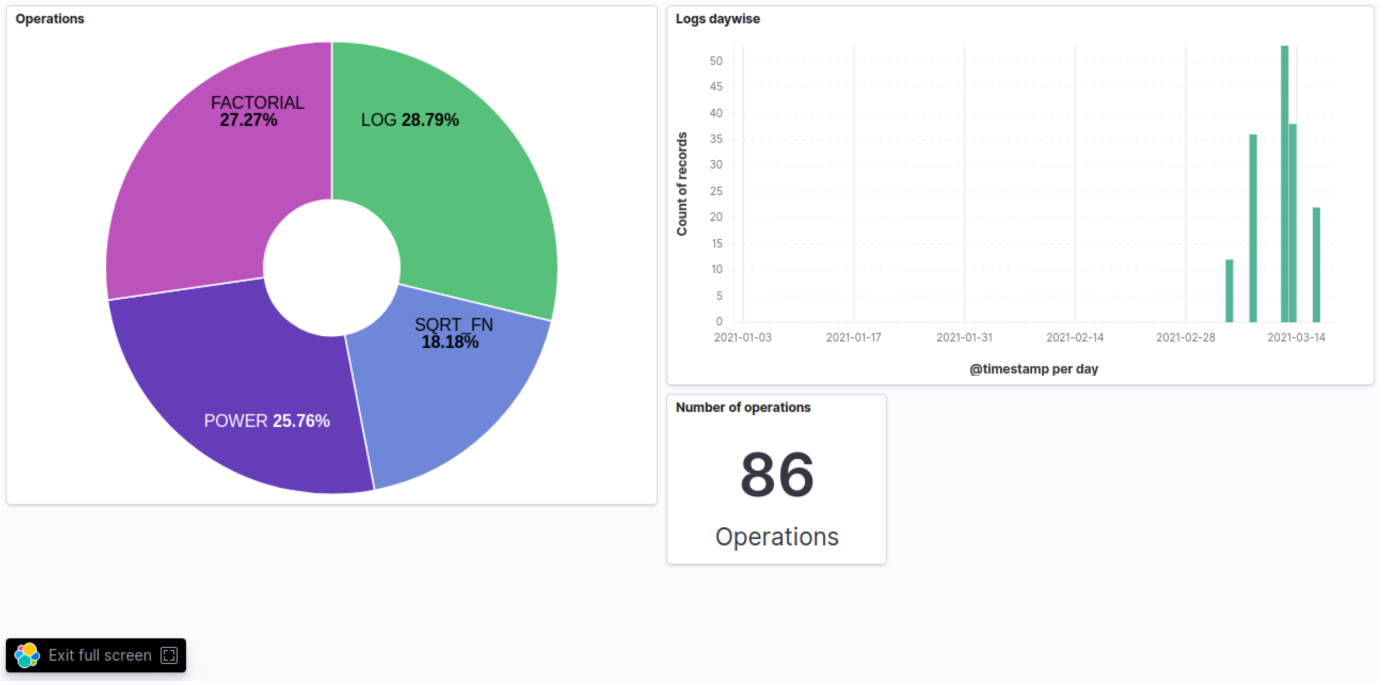
10.4 ELK Stack

Logstash tool uses the logs in the system to send to the ELK cloud. In the cloud Elastic Search parses and processes the given logs and then sends to Kibana to visualize.

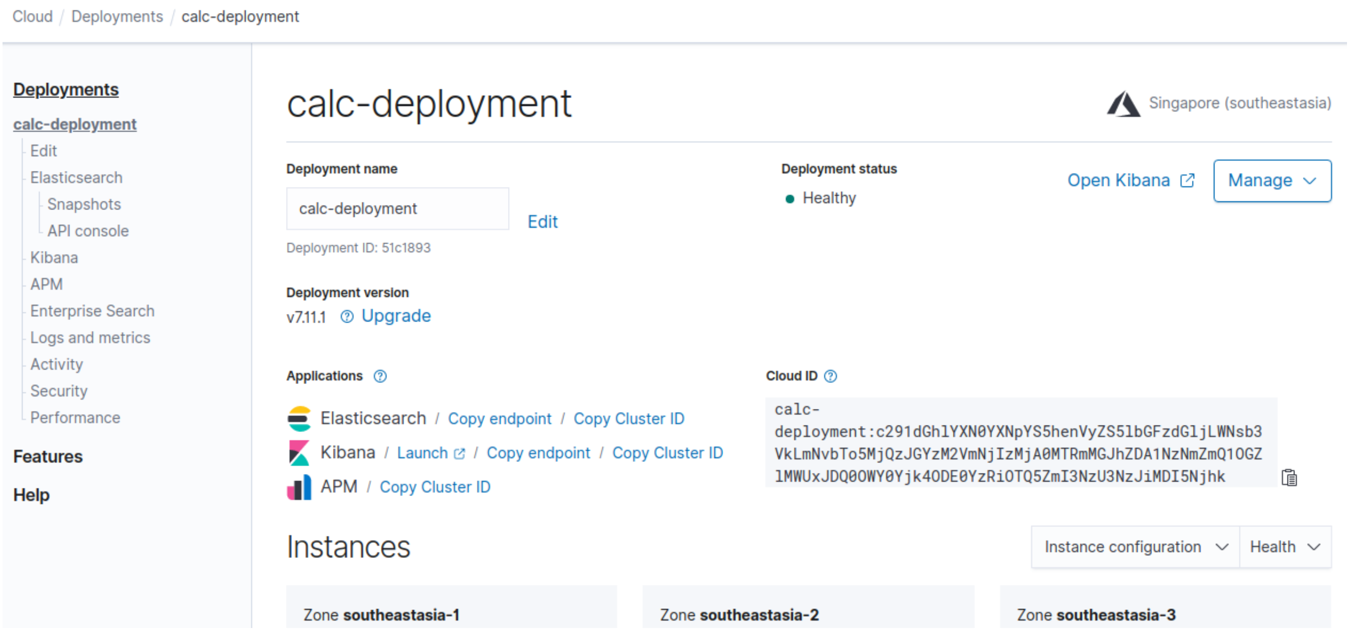


10.5 Dashboard

A simple dashboard built by using **Lens** of ELK cloud. It shows the split of the operations, count of them and also the operations per day.



The below is a picture of ELK in cloud.



Cloud / Deployments / calc-deployment

Deployments
calc-deployment
Edit
Elasticsearch
Snapshots
API console
Kibana
APM
Enterprise Search
Logs and metrics
Activity
Security
Performance
Features
Help

calc-deployment

Singapore (southeastasia)

Deployment name: calc-deployment Edit
Deployment ID: 51c1893

Deployment status: ● Healthy

Open Kibana Manage

Deployment version: v7.11.1 Upgrade

Applications
Elasticsearch / Copy endpoint / Copy Cluster ID
Kibana / Launch / Copy endpoint / Copy Cluster ID
APM / Copy Cluster ID

Cloud ID

```
calc-
deployment:c291dGhlYXN0YXNpYS5henVyZS5lbGFzdGljLWNsb3
VkLmNvbTo5MjQzJGYzM2VmNjIzMjA0MTRmMGJhZDA1NzNmZmQ1OGZ
lMWUxJDQ0OWY0Yjk4ODE0YzRiOTQ5ZmI3NzU3NzJiMDI5Njhk
```

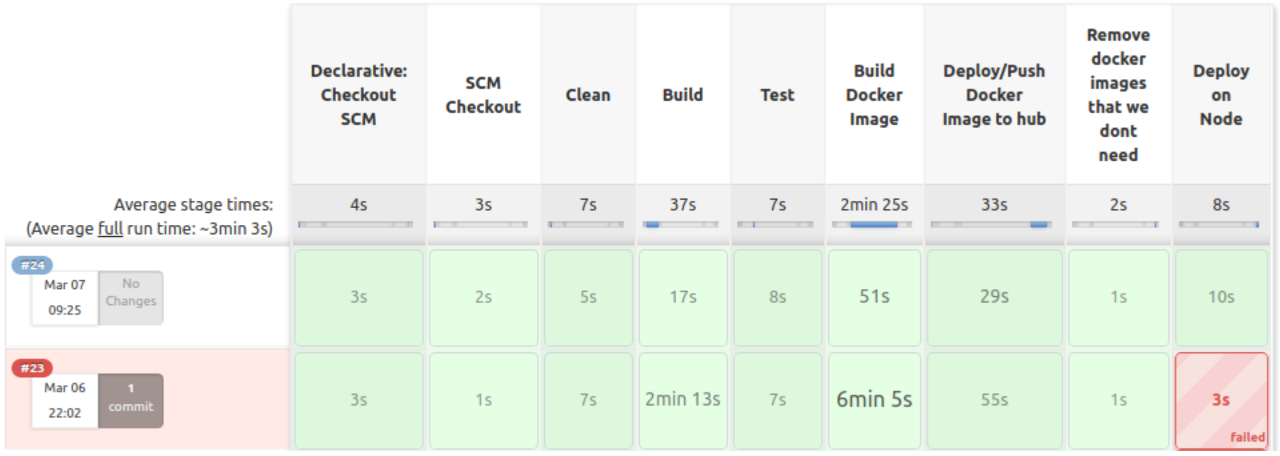
Instances

Instance configuration Health

Zone **southeastasia-1** Zone **southeastasia-2** Zone **southeastasia-3**

11 Pipeline

In this section we will discuss in depth about our specific project and the configurations done and files.



	Declarative: Checkout SCM	SCM Checkout	Clean	Build	Test	Build Docker Image	Deploy/Push Docker Image to hub	Remove docker images that we dont need	Deploy on Node
Average stage times: (Average full run time: ~3min 3s)	4s	3s	7s	37s	7s	2min 25s	33s	2s	8s
#24 Mar 07 09:25 No Changes	3s	2s	5s	17s	8s	51s	29s	1s	10s
#23 Mar 06 22:02 1 commit	3s	1s	7s	2min 13s	7s	6min 5s	55s	1s	3s failed

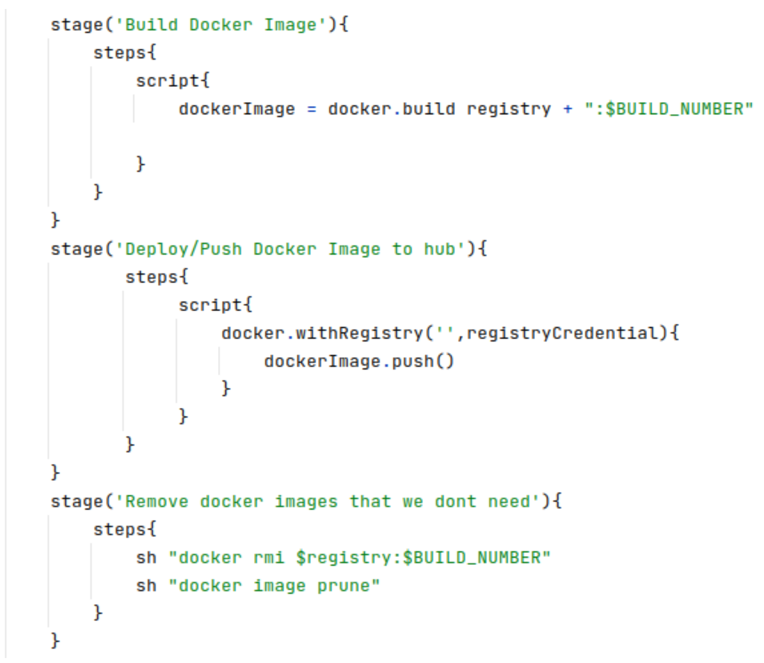
The basic idea is the CI tool i.e. Jenkins keeps polling the SCM i.e. Git repo to see if there are any changes. Incase there are any changes then the pipeline starts. The polling is done every 15minutes and more details are given below.

11.1 Pipeline

It polls the SCM (https://github.com/seethamraju/calculator) for commits. Then it **clones** the repo and then **cleans**, **builds** and **tests** using the maven commands. Once this is done we will have a jar/war file with us. Now a simple **docker image** is built. Once latest image is build it is then pushed to the dockerhub using the tag as the build number. <**tag name**> = **build number** After pushing it to docker hub it then runs the Rundeck Job which is to pull the docker image on a **node** and run it once.

```
environment{
    registry = "byterider/cal"
    registryCredential = 'dockerhub'
    dockerImage = ''
}
agent any
```

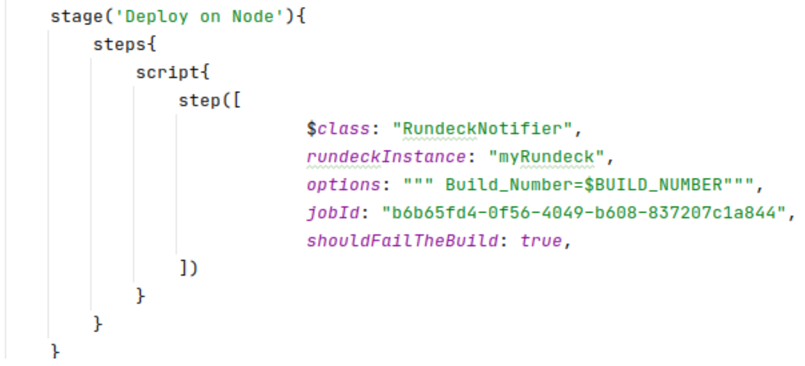
Here are the environment variables we will be using in the script file. *registry* , *registryCredential* refer to the



```
stage('Build Docker Image'){
    steps{
        script{
            dockerImage = docker.build registry + ":$BUILD_NUMBER"

        }
    }
}
stage('Deploy/Push Docker Image to hub'){
       steps{
            script{
                docker.withRegistry('',registryCredential){
                    dockerImage.push()
                }
            }
       }
}
stage('Remove docker images that we dont need'){
    steps{
        sh "docker rmi $registry:$BUILD_NUMBER"
        sh "docker image prune"
    }
}
```

Here we are basically building the docker image using the dokcer plugin in Jenkins. The image is stored in **dockerImage**. This is then pushed to the dockerhub registry with the given credentials that are represented by **registryCredential**. The image is pushed with the tag as the **BUILD_NUMBER** which comes from the Jenkins Environment. By default the image is pushed to docker hub hence the link is set as **'byterider/cal'**. After that we clean the local workspace by removing all the images inside it.



```
stage('Deploy on Node'){
    steps{
        script{
            step([
                        $class: "RundeckNotifier",
                        rundeckInstance: "myRundeck",
                        options: """ Build_Number=$BUILD_NUMBER""",
                        jobId: "b6b65fd4-0f56-4049-b608-837207c1a844",
                        shouldFailTheBuild: true,
            ])
        }
    }
}
```

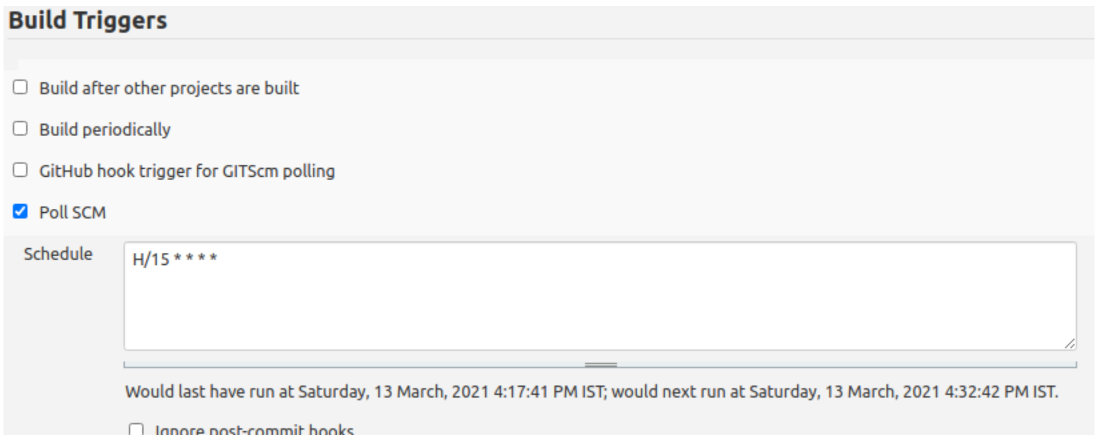
At last we run the required rundeck job which basically pulls the pushed docker image and then runs it. Here we specify the rundeck instance that needs to be configured, the Job-Id and also the **options** i.e. the arguments for the Job. Here the argument is the **BUILD_NUMBER** which the Rundeck job uses to pull the docker image with the given tag(BUILD NUMBER).

11.2 SCM Polling

The Jenkins tool Polls the SCM every few minutes to see if there are any new changes. Incase it finds any changes then it starts the processes in the pipeline. The duration of the time between two Polls is specified using the **Cron** format.

* * * * *

1st = minute
2nd = hour
3rd = day
4th = month
5th = day of the week


Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [x] Poll SCM

Schedule: H/15 * * * *

Would last have run at Saturday, 13 March, 2021 4:17:41 PM IST; would next run at Saturday, 13 March, 2021 4:32:42 PM IST.

- [ ] Ignore post-commit hooks

Here the Polling is done every 15minutes. **H/15** is the one that indicates that it shoudl poll for every 15min every hour, day.

11.3 Building and Packaging

These were discussed in the Section 5

11.4 Dockerization

These were discussed in Section 7

11.5 Integration

This was discussed in Section 8

11.6 Deployment

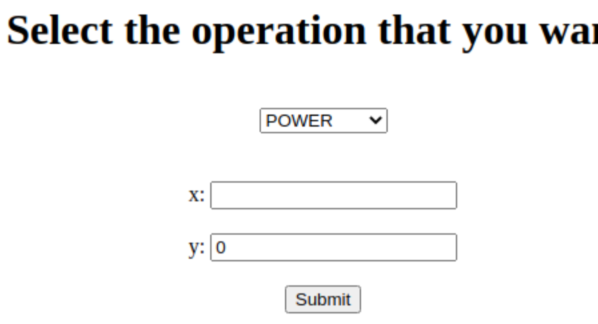
These were discussed in Section 9

11.7 Logging Management

This was discussed in Section 10

12 The WebAPP variant (Optional)

The webapp uses Java servlet concept. The code can be found on the same repo on the branch named webapp.


Select the operation that you wa

POWER

x:

y: 0

Submit

This has a *.war* file that can be deployed on the tomcat server.
The app is temporarily hosted in this link. (However incase the free subscription ends it might not exist :))
http://purvaj.southindia.azurecontainer.io:8080/calculatorweb-1.0-SNAPSHOT/

12.1 Docker

The following is the docker file

```
FROM tomcat:9.0
ADD target/calculatorweb-1.0-SNAPSHOT.war /usr/local/tomcat/webapps/
EXPOSE 8080
CMD ["catalina.sh", "run"]
```

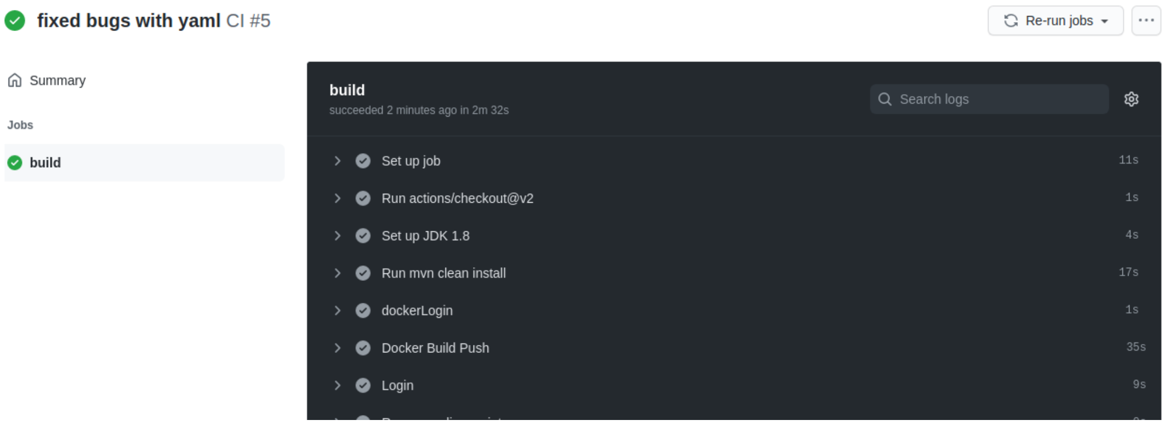
The ADD command is used when we are transferring files like war which are in compressed format. The dockerimage corresponding to it is **byterider/calweb**.

12.2 Deployment

This app was deployed on **Azure container Instance**. We basically create a container instance in azure. PFB the ss for the process. We specify the properties of the container and also the options incase of failure along with the required ports.

12.3 Continuous Deployment using Github Actions

For deploying the webapp into the Azure container Instance we use **Github Actions** which is basically github's own CI/CD workflow integration. The github actions requires a *.yaml* file inside its **.github/workflow/** directory. A snippet of the yaml file is given below. Few other screenshots are about the github actions UI and the status of the builds.

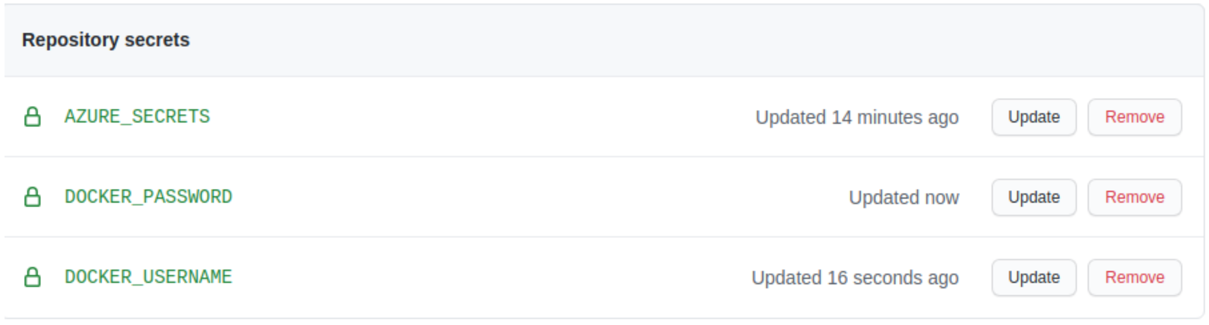


fixed bugs with yaml CI #5

build
succeeded 2 minutes ago in 2m 32s

```
Set up job                    11s
Run actions/checkout@v2        1s
Set up JDK 1.8                 4s
Run mvn clean install         17s
dockerLogin                    1s
Docker Build Push             35s
Login                          9s
```

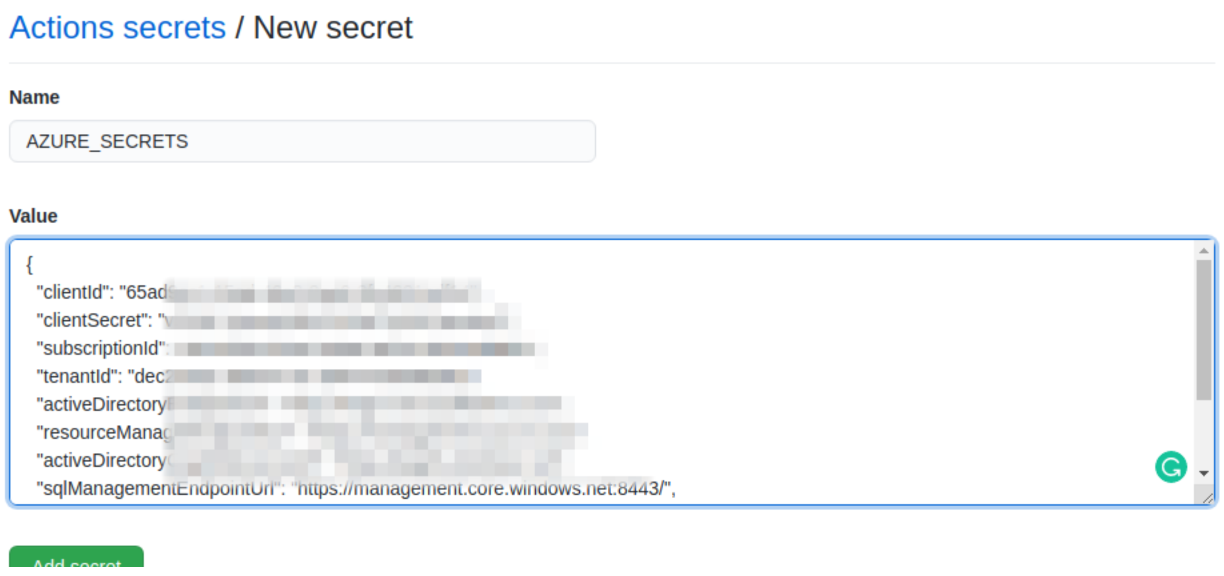
We have to configure the Azure AD service principles, which basically are the login credentials of the Azure account and also the credentials for docker hub just like we did in Jenkins. For that we use the concept of **secrets** in Github.



Repository secrets

AZURE_SECRETS	Updated 14 minutes ago	Update	Remove
DOCKER_PASSWORD	Updated now	Update	Remove
DOCKER_USERNAME	Updated 16 seconds ago	Update	Remove

List of Secrets.



Actions secrets / New secret

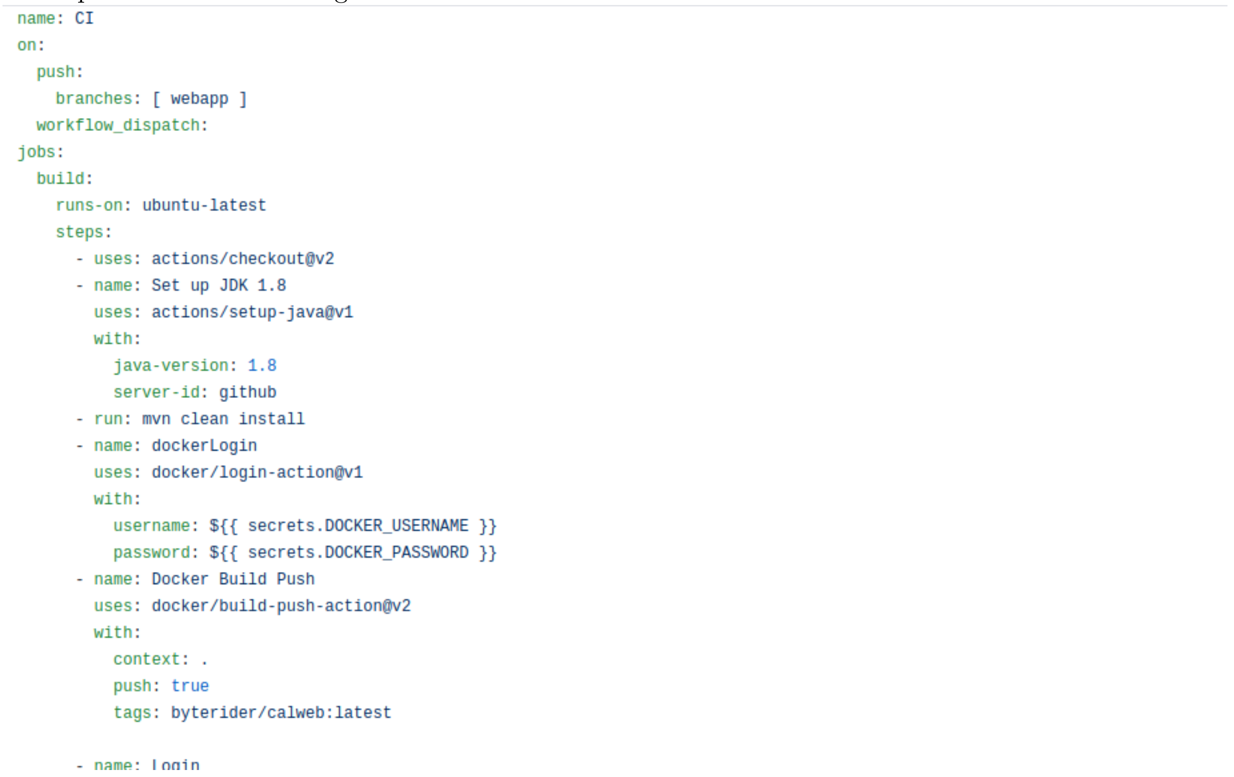
Name

AZURE_SECRETS

Value

```
{
  "clientId": "65ad[redacted]
  "clientSecret": "[redacted]
  "subscriptionId": [redacted]
  "tenantId": "dec[redacted]
  "activeDirectory[redacted]
  "resourceManag[redacted]
  "activeDirectory[redacted]
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
```

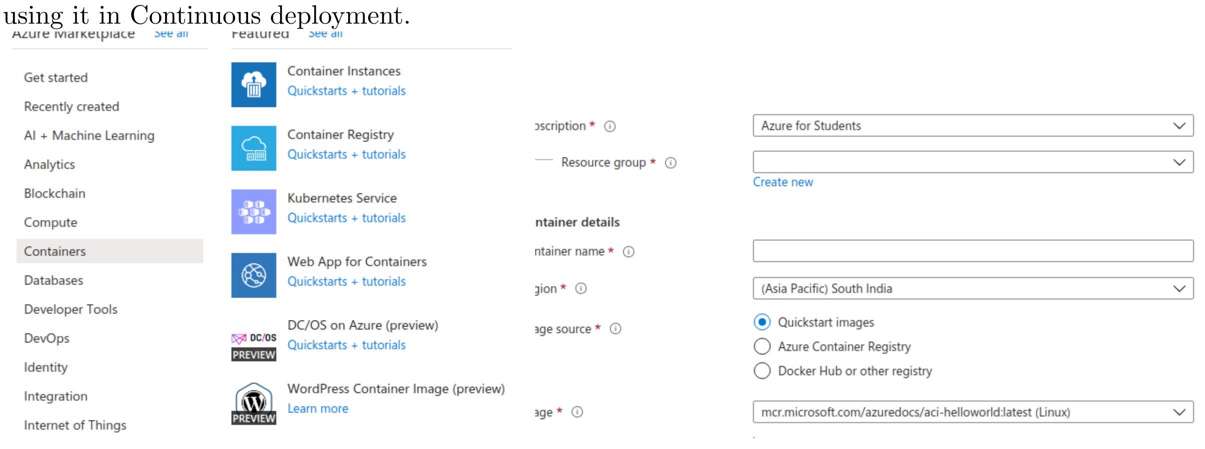
A sample secret for Azure login.



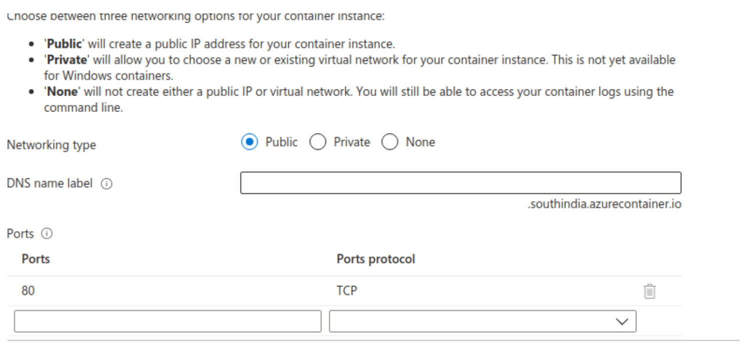
```yaml
name: CI
on:
  push:
    branches: [ webapp ]
  workflow_dispatch:
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up JDK 1.8
        uses: actions/setup-java@v1
        with:
          java-version: 1.8
          server-id: github
      - run: mvn clean install
      - name: dockerLogin
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}
      - name: Docker Build Push
        uses: docker/build-push-action@v2
        with:
          context: .
          push: true
          tags: byterider/calweb:latest

      - name: Login
```

A snippet of the *yaml* file which is used by the github actions. The syntax is not discussed as it is out of the scope of this report. Below are the screenshots for creating a container instance in Azure, however we won't be using it in Continuous deployment.



ɔscription *	Azure for Students
Resource group *	
ntainer details	
ntainer name *	
ʒion *	(Asia Pacific) South India
ɑge source *	[✓ Quickstart images] Azure Container Registry Docker Hub or other registry
ɑge *	mcr.microsoft.com/azuredocs/aci-helloworld:latest (Linux)


Choose between three networking options for your container instance:

- '**Public**' will create a public IP address for your container instance.
- '**Private**' will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- '**None**' will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type: [✓ Public] Private None

DNS name label: .southindia.azurecontainer.io

Ports

Ports	Ports protocol
80	TCP

13 Challenges faced

1. For a Jenkins tool or a Rundeck tool to use docker we had to add them to the docker user group using the following command.

   ```
   $ sudo usermod -aG docker <servicename>
   $ sudo usermod -aG docker jenkins # example
   ```

2. There is another problem for pushing dockerhub due to the required credentials. This can be dealt installing the packages. This arises as the jenkins and rundeck cannot access the dockerhub directly.

   ```
   $ sudo apt install gnupg2 pass
   ```

3. Since we are actually using continuous Integration , delivery and deployment, the terminal based application could not be tested with manual intervention. Hence a set of inputs were flushed into the app from ***input.txt*** while building the docker container.

References

1. https://stackoverflow.com/
2. https://www.jenkins.io
3. https://docs.microsoft.com/en-us/azure/
4. https://docs.github.com/en/actions