DevOps Toolchain

Calculator.

Nomula Rohith Yogi

IMT2016072

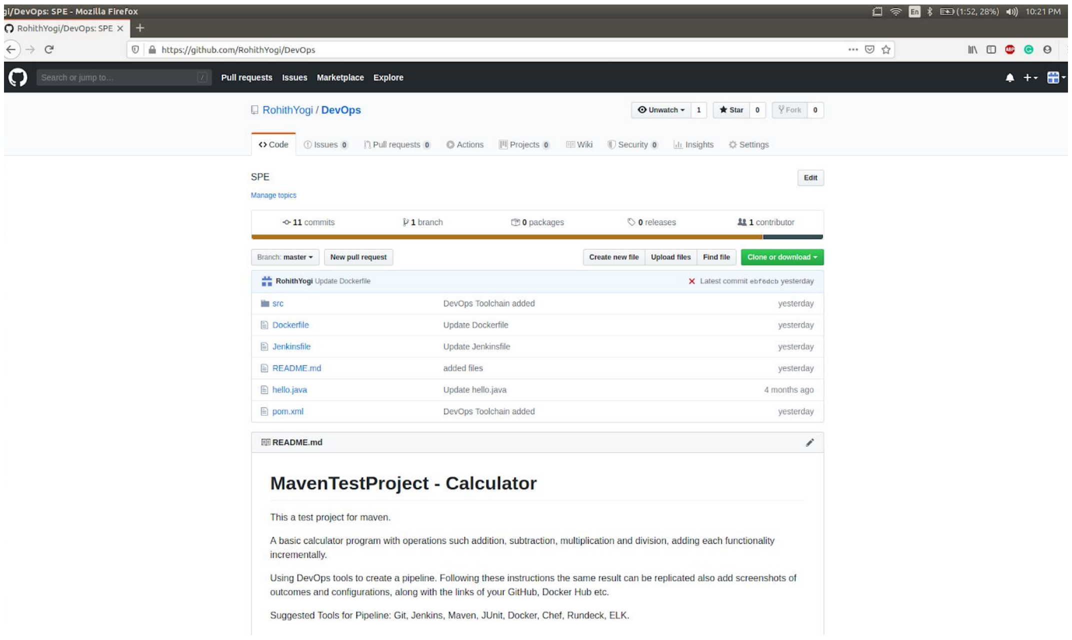
Introduction

This project sets out to create a basic calculator program with operations such as addition, subtraction, multiplication and division. In this project we explore the use cases of DevOps tools as we aim to add these functionalities incrementally. The project follows these stages which are a common sight in a typical software development life cycle.

1. Coding - GitHub
2. Continuous Integration - Maven, Jenkins
3. Testing - JUnit
4. Delivery - Docker
5. Deployment - Rundeck
6. References

Coding - **GitHub.**

Firstly, we create a github repository for our calculator project. This repository helps us to maintain our code in a version control environment. I have created just that in my GitHub account. https://github.com/RohithYogi .



SPE

RohithYogi Update Dockerfile — Latest commit ebfedcb yesterday

src	DevOps Toolchain added	yesterday
Dockerfile	Update Dockerfile	yesterday
Jenkinsfile	Update Jenkinsfile	yesterday
README.md	added files	yesterday
hello.java	Update hello.java	4 months ago
pom.xml	DevOps Toolchain added	yesterday

README.md

MavenTestProject - Calculator

This a test project for maven.

A basic calculator program with operations such addition, subtraction, multiplication and division, adding each functionality incrementally.

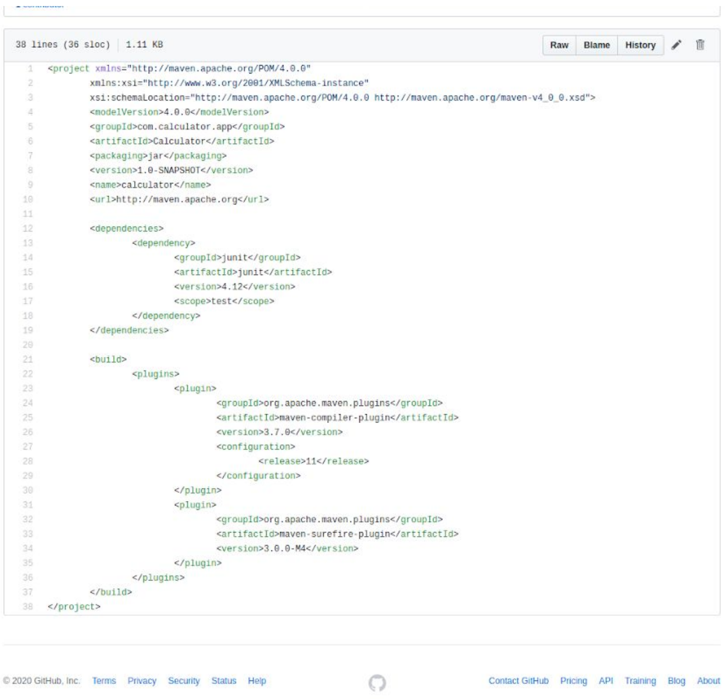
Using DevOps tools to create a pipeline. Following these instructions the same result can be replicated also add screenshots of outcomes and configurations, along with the links of your GitHub, Docker Hub etc.

Suggested Tools for Pipeline: Git, Jenkins, Maven, JUnit, Docker, Chef, Rundeck, ELK.

In the src folder you will find all the code related to this project. This git environment enables us to maintain various versions of our code and also helps us to integrate the code with other platforms like jenkins and docker hub. Thus this is very important for the application development process. After creating a repository, clone it to your local system so that you can develop from your IDE. After we write a basic calculator application in java while following the structure of the package we check if the java file is compiling and running successfully. Then we commit the changes and push them to the GitHub repository. While this is the process followed by most of the developers we can also find it useful if we do this process locally instead of entering into the github repository.

Continuous Integration - **Maven.**

This stage we create a POM (project object model) file. This file is essential and powerful for the build process. This tool helps in building java based projects with all the dependencies and helps us in managing the process requirements. This pom.xml file contains all the dependencies and the plugins associated with the java project and helps us bring all under one roof for building the project. It looks like this :

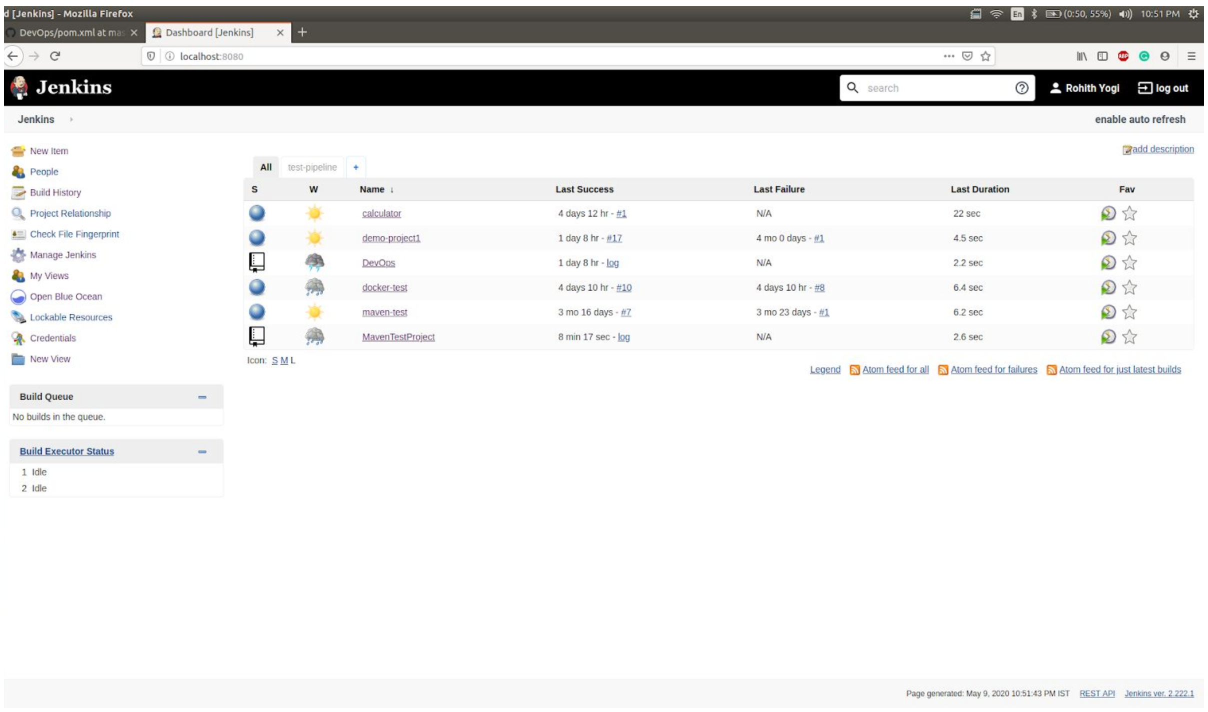


```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.calculator.app</groupId>
        <artifactId>Calculator</artifactId>
        <packaging>jar</packaging>
        <version>1.0-SNAPSHOT</version>
        <name>calculator</name>
        <url>http://maven.apache.org</url>

        <dependencies>
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.12</version>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.7.0</version>
                                <configuration>
                                        <release>11</release>
                                </configuration>
                        </plugin>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-surefire-plugin</artifactId>
                                <version>3.0.0-M4</version>
                        </plugin>
                </plugins>
        </build>
</project>
```

This file contains dependencies like junit for testing the java project and maven compiler and surefire plugins for build step. Now we have the code and we know how to build the code. We now need a tool to link jenkins to integrate further and create a pipeline for the development process.

Continuous Integration - **Jenkins.**

Now we enter the pipeline stage. Install jenkins and start the service. We will be able to see it running in the *localhost:8080* Then setup jenkins with a username and password and enter your credentials and login to jenkins you will have a dashboard page similar to this



S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
●	☀	calculator	4 days 12 hr - #1	N/A	22 sec	☆
●	☀	demo-project1	1 day 8 hr - #17	4 mo 0 days - #1	4.5 sec	☆
		DevOps	1 day 8 hr - log	N/A	2.2 sec	☆
●		docker-test	4 days 10 hr - #10	4 days 10 hr - #8	6.4 sec	☆
●	☀	maven-test	3 mo 16 days - #7	3 mo 23 days - #1	6.2 sec	☆
		MavenTestProject	8 min 17 sec - log	N/A	2.6 sec	☆

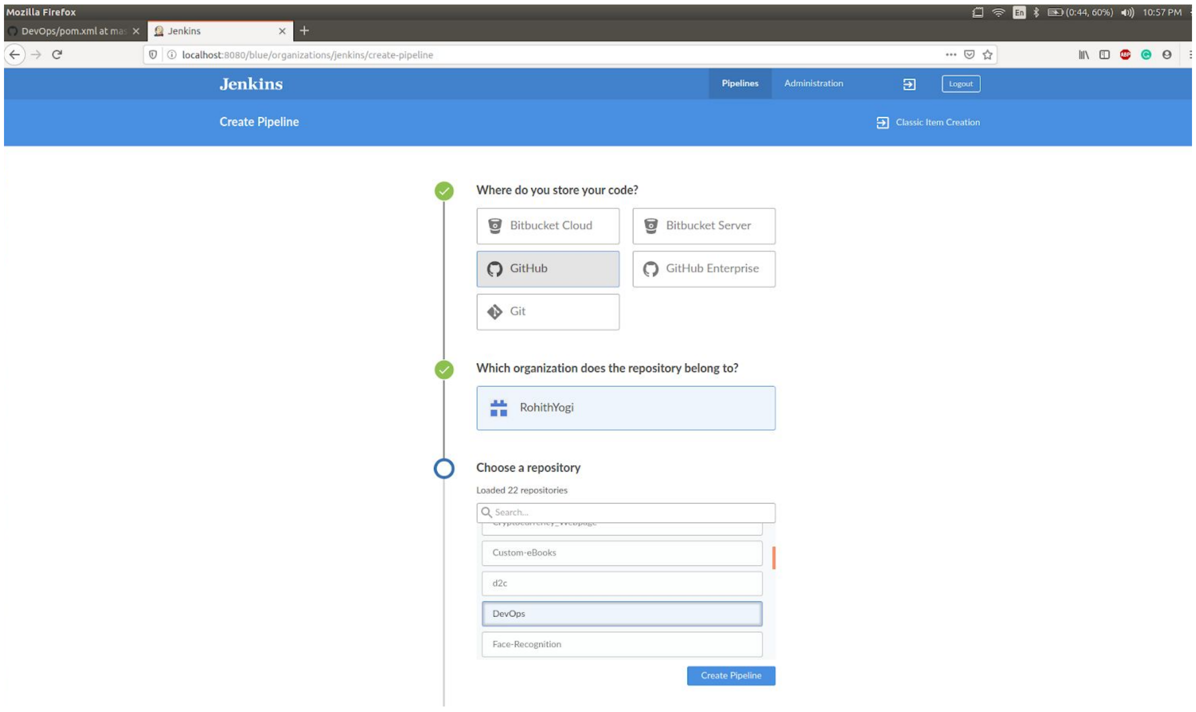
Icon: S M L

Build Queue
No builds in the queue.

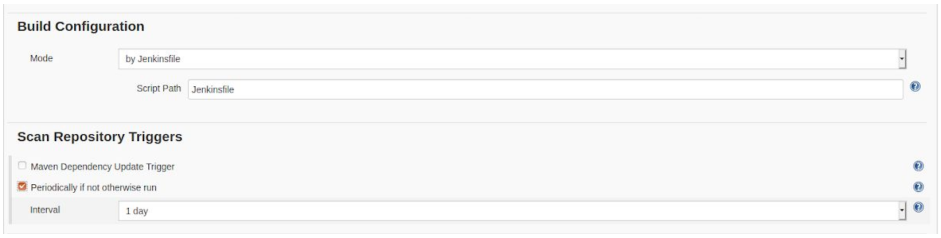
Build Executor Status
1 Idle
2 Idle

Page generated: May 9, 2020 10:51:43 PM IST REST API Jenkins ver. 2.222.1

Here we find the manage jenkins option where you install git, GitHib, docker, rundeck and blue ocean plugins required for this project. The git and GitHub plugins enable you to integrate with git repo and build your project. Blue Ocean plugin where we create the pipeline of the project. Now we already have a pom file and our code ready to go and we have it in our git repository. Open blue ocean and click on create pipeline now we see a page similar to this.



Select accordingly and you will reach the create pipeline option select it and we now enter pipeline code here. Or else we also have other options of creating an external pipeline and integrating it to the jenkins service to build. The pipeline code has all the stages in it first it will clone the repository for further use of the code then it will install dependencies of maven and build using the pom file then it will execute the testing stage using junit tool and we have thus completed build and test in the continuous integration step further sections will add on to this. But this way of creating jenkins process creates a repo directly in jenkins dashboard page as seen above where we now have to configure build triggers and establish the connection more concretely. The GitHub credentials are not required because it is already loaded and in the build triggers sections enter what duration you want to check the git repo for changes and build it. These are the build configurations that we specify. Now the jenkins file here is generated from the pipeline we create in blue ocean or it can be done locally and pushed into the repo. Blue ocean recognises this file and further takes into visualising the pipeline that was coded in it.



Build Configuration

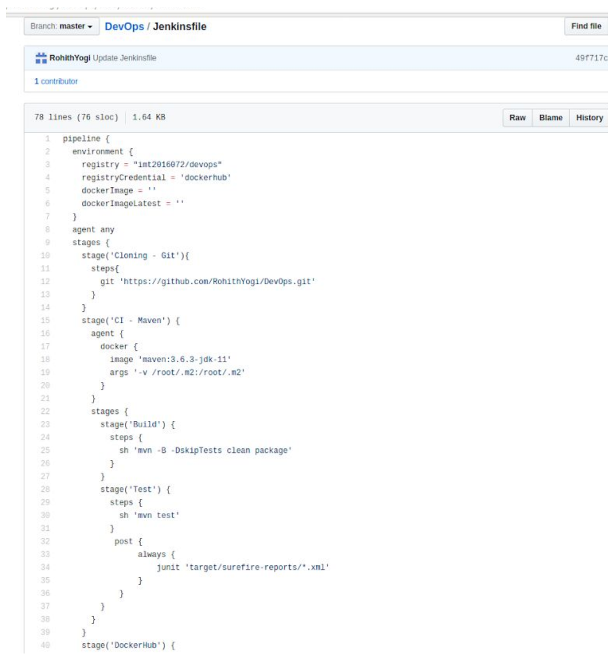
Mode	by Jenkinsfile
Script Path	Jenkinsfile

Scan Repository Triggers

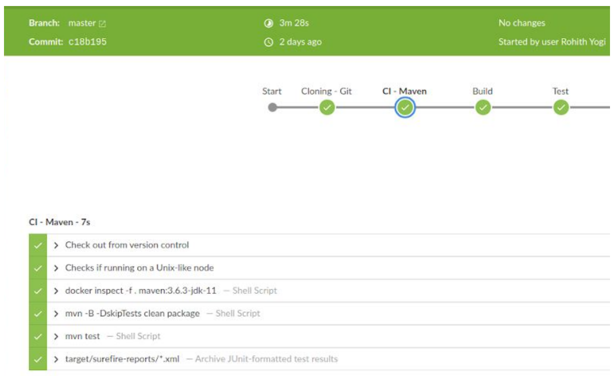
- [ ] Maven Dependency Update Trigger
- [✓] Periodically if not otherwise run

Interval	1 day

The jenkins pipeline looks like this:



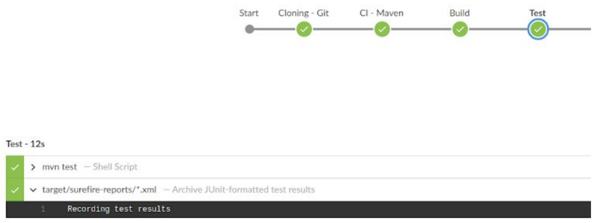
```
pipeline {
  environment {
    registry = "imt2016072/devops"
    registryCredential = 'dockerhub'
    dockerImage = ''
    dockerImageLatest = ''
  }
  agent any
  stages {
    stage('Cloning - Git'){
      steps{
        git 'https://github.com/RohithYogi/DevOps.git'
      }
    }
    stage('CI - Maven') {
      agent {
        docker {
          image 'maven:3.6.3-jdk-11'
          args '-v /root/.m2:/root/.m2'
        }
      }
      stages {
        stage('Build') {
          steps {
            sh 'mvn -B -DskipTests clean package'
          }
        }
        stage('Test') {
          steps {
            sh 'mvn test'
          }
           post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
      }
    }
    stage('DockerHub') {
```



CI - Maven - 7s

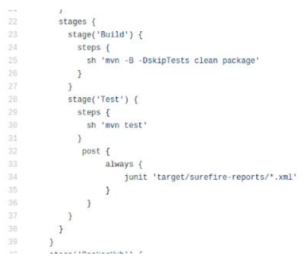
```
Check out from version control
Checks if running on a Unix-like node
docker inspect -f . maven:3.6.3-jdk-11 — Shell Script
mvn -B -DskipTests clean package — Shell Script
mvn test — Shell Script
target/surefire-reports/*.xml — Archive JUnit-formatted test results
```

Testing - **JUnit.**



Test - 12s

```
mvn test — Shell Script
target/surefire-reports/*.xml — Archive JUnit-formatted test results
1    Recording test results
```

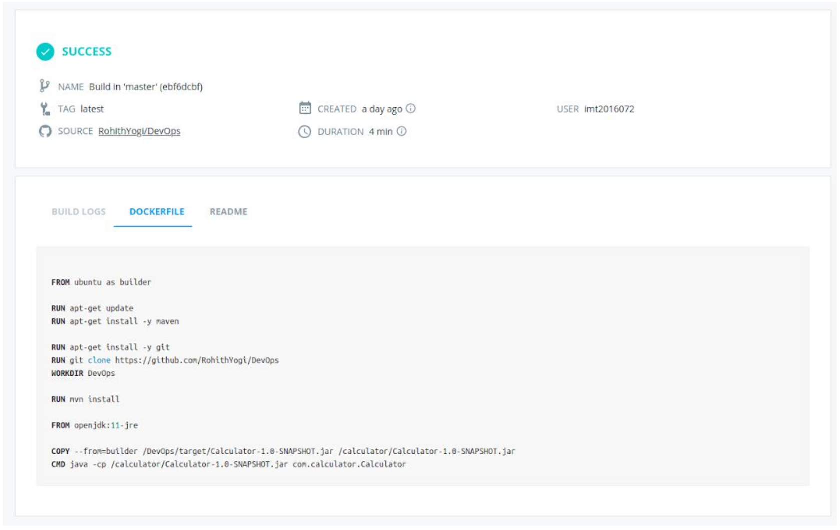


```
      stages {
        stage('Build') {
          steps {
            sh 'mvn -B -DskipTests clean package'
          }
        }
        stage('Test') {
          steps {
            sh 'mvn test'
          }
           post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
      }
    }
```

From the target surefire we test our project using the junit specified in the pom file. The pipeline is visualized like the images above. This helps us categorize our project stage wise and prioritize based on the logs received. We have just finished the cloning, Maven installation - build and test stages and we now move on to delivery using docker hub. This is an important stage where we containerize our project install dependencies and run it all using one docker file.

Delivery - **Docker.**

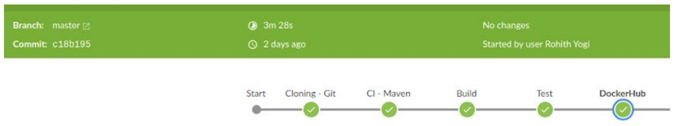
Now we create a docker hub account. Login to your docker hub account. You can find my docker hub calculator project here : https://hub.docker.com/repository/docker/imt2016072/calculator . We must essentially write a docker file for our application. Create an environment in the container where our project needs to run by using the docker file. First we create a repository and connect it to our GitHub as done earlier for jenkins then docker hub directly looks for Dockerfile in our repository and builds that. We can even check if the docker file is running correctly or not. For writing the Dockerfile we must know what is required for our program to run.



NAME Build in 'master' (ebf6dcbf)
TAG latest
SOURCE RohithYogi/DevOps
CREATED a day ago
DURATION 4 min
USER imt2016072

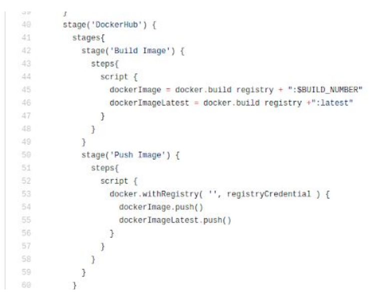
```
FROM ubuntu as builder

RUN apt-get update
RUN apt-get install -y maven

RUN apt-get install -y git
RUN git clone https://github.com/RohithYogi/DevOps
WORKDIR DevOps

RUN mvn install

FROM openjdk:11-jre

COPY --from=builder /DevOps/target/Calculator-1.0-SNAPSHOT.jar /calculator/Calculator-1.0-SNAPSHOT.jar
CMD java -cp /calculator/Calculator-1.0-SNAPSHOT.jar com.calculator.Calculator
```

First we have a container with ubuntu. Now we update the container by running apt-get update on it sudo is not required since we are the root. Then we install maven dependencies and install git after that clone our own git repository into the container for executing the program. Copy the build step and run the command for running the java file here in the Dockerfile. These commands will be executed when we run the docker container. Since the docker hub provides us a great interface to work with and manage the projects we use it to build the docker image and check the status and inform jenkins. For this we have to integrate docker hub to jenkins. We can do this by adding credentials in the jenkins configuration, name the key and use it in our pipelining code. The code we write here basically tells docker hub to build and push the latest image.

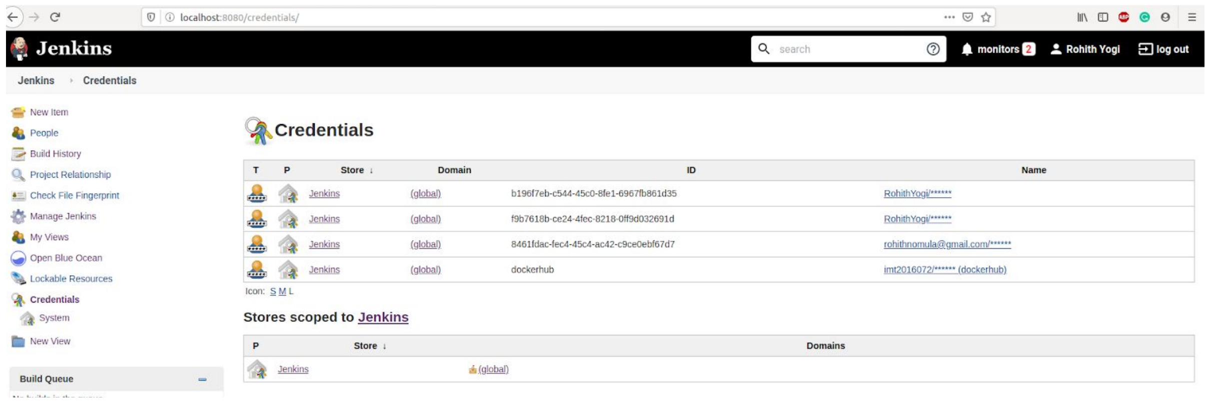

Branch: master
Commit: c18b195
3m 28s
2 days ago
No changes
Started by user Rohith Yogi

DockerHub - <1s

```
✓ > Checks if running on a Unix-like node
✓ > Shell Script
✓ > Checks if running on a Unix-like node
✓ > Shell Script
✓ > Checks if running on a Unix-like node
✓ > Shell Script
✓ > Checks if running on a Unix-like node
✓ > Shell Script
✓ > Checks if running on a Unix-like node
✓ > Shell Script
✓ > Checks if running on a Unix-like node
✓ > Shell Script
```



```
  }
  stage('DockerHub') {
    stages{
      stage('Build Image') {
        steps{
          script {
            dockerImage = docker.build registry + ":$BUILD_NUMBER"
            dockerImageLatest = docker.build registry +":latest"
          }
        }
      }
      stage('Push Image') {
        steps{
          script {
            docker.withRegistry( '', registryCredential ) {
              dockerImage.push()
              dockerImageLatest.push()
            }
          }
        }
      }
    }
  }
```



Credentials

T	P	Store ↓	Domain	ID	Name
		Jenkins	(global)	b196f7eb-c544-45c0-8fe1-6967fb861d35	RohithYogi/******
		Jenkins	(global)	f9b7618b-ce24-4fec-8218-0ff9d032691d	RohithYogi/******
		Jenkins	(global)	8461fdac-fec4-45c4-ac42-c9ce0ebf67d7	rohithnomula@gmail.com/******
		Jenkins	(global)	dockerhub	imt2016072/****** (dockerhub)

Icon: S M L

Stores scoped to Jenkins

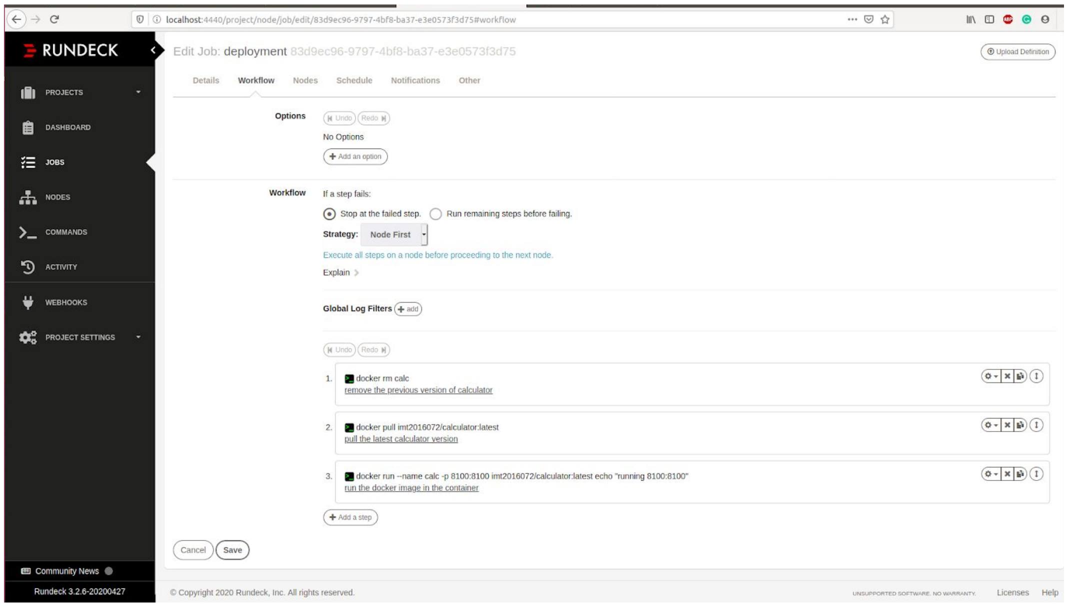
P	Store ↓	Domains
	Jenkins	(global)

Add the credentials to the docker hub here in this page and proceed to tagging and naming it. Now we use this in our jenkins pipeline file for building and pushing the image. This usually requires a docker hub password for authenticating your repository and using the Dockerfile to build the image. This completes the docker hub step of building and pushing the docker image. We have to have the plugins installed in the jenkins for docker and rundeck. This is a simple installation process that can be done by plugin Manager in jenkins. Locally we need to have git, docker, jenkins and rundeck assuming you already have java, maven etc pre installed in your system. Thus far we have completed build, test and deliver (using docker container) .

Now we enter the deployment stage.

Deployment - **Rundeck.**

Install and start rundeck service. We will have to open http://localhost:4440/ for using the rundeck dashboard. Make sure you wait for service to start completely before drawing into any conclusions. You can do this by running *tail -f /var/log/rundeck/service.log* command and check if the host is in production. Here, we aim only to deploy the program in the local machine. But we can further add on ssh service nodes for deploying it to many nodes. The ssh service is password protected but there are workarounds like having a separate docker container as root user login and install ssh in it to enable creating multiple nodes. But in this process we have to again install docker in these containers too and enable access to rundeck service by adding it into the docker group and setting permissions for it. Although I tried this complex method I am not keen to report it here as it will only add more doubts. So I have followed a normal process and made rundeck deploy only in the local machine. First we have to ensure that we have java 8 version running in the background if not it will only lead to more rundeck errors. By: *update-alternatives --config java* : we can select different versions of java to run before that application starts. I personally have faced a lot of issues if java 11 was enabled before rundeck service starts. So I recommend checking the version before starting the service. Now we enter the rundeck dashboard with the create project option so click and enter the name of project and create button. Now create a job with this workflow.



Edit Job: deployment 83d9ec96-9797-4bf8-ba37-e3e0573f3d75

Options: No Options

Workflow: If a step fails: [✓ Stop at the failed step.] Run remaining steps before failing.

Strategy: Node First

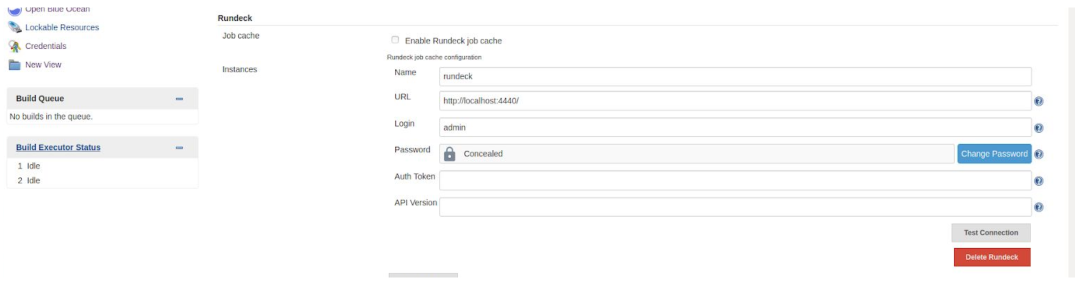
Execute all steps on a node before proceeding to the next node.

Global Log Filters

1. docker rm calc
   remove the previous version of calculator
2. docker pull imt2016072/calculator:latest
   pull the latest calculator version
3. docker run --name calc -p 8100:8100 imt2016072/calculator:latest echo "running 8100:8100"
   run the docker image in the container

© Copyright 2020 Rundeck, Inc. All rights reserved. UNSUPPORTED SOFTWARE. NO WARRANTY.

We now include the commands required for the docker image to run. Docker hub tags the last pushed docker image as :latest so we use that here. First command in the above image removes previous versions of the calculator. The next command pulls the latest calculator image and the final one runs it. I have redirected it to run in port 8100 because its a program and needs menu driven options choosing we can not directly run it. If we do run it normally we get a tty error saying no input options etc. so it's better this way we know for sure our code is working and the calculator is built on the latest pushed code. Before this we have to give rundeck access to run docker. We can do this by adding rundeck to groups and setting permissions. After this we copy the job ID and keep it with us for pipeline implementation of this process. Further we can check if the job is running here in the rundeck itself. First, we have to configure the rundeck in jenkins as show



No builds in the queue.

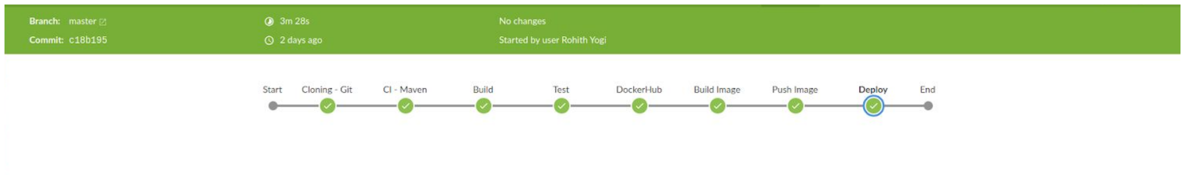
1 Idle
2 Idle

Rundeck

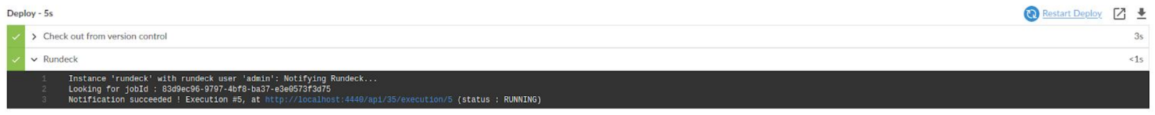
Job cache: ☐ Enable Rundeck job cache

Instances: Rundeck job cache configuration

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



Branch: master
Commit: c18b195
3m 28s
2 days ago
No changes
Started by user Rohith Yogi

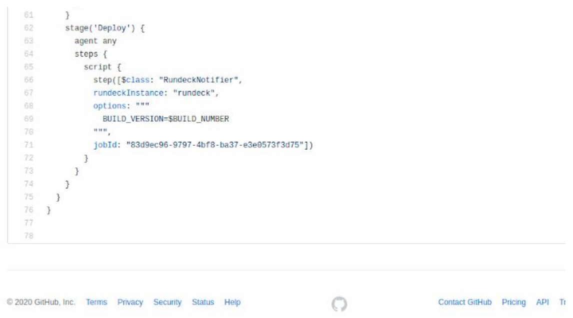


Deploy - 5s

✓ Check out from version control 3s

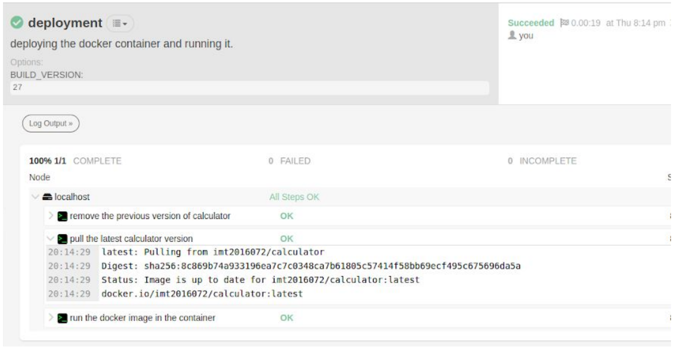
✓ Rundeck <1s

```
Instance 'rundeck' with rundeck user 'admin': Notifying Rundeck...
Looking for jobId : 83d9ec96-9797-4bf8-ba37-e3e0573f3d75
Notification succeeded ! Execution #5, at http://localhost:4440/api/35/execution/5 (status : RUNNING)
```



```
    }
    stage('Deploy') {
      agent any
      steps {
        script {
          step([$class: "RundeckNotifier",
          rundeckInstance: "rundeck",
          options: """
            BUILD_VERSION=$BUILD_NUMBER
          """,
          jobId: "83d9ec96-9797-4bf8-ba37-e3e0573f3d75"])
        }
      }
    }
  }
}
```

© 2020 GitHub, Inc. Terms Privacy Security Status Help Contact GitHub Pricing API Tr



deployment

deploying the docker container and running it.

Options:
BUILD_VERSION:
27

Succeeded 0.00:19 at Thu 8:14 pm
you

100% 1/1 COMPLETE 0 FAILED 0 INCOMPLETE

Node

localhost All Steps OK

remove the previous version of calculator OK

pull the latest calculator version OK

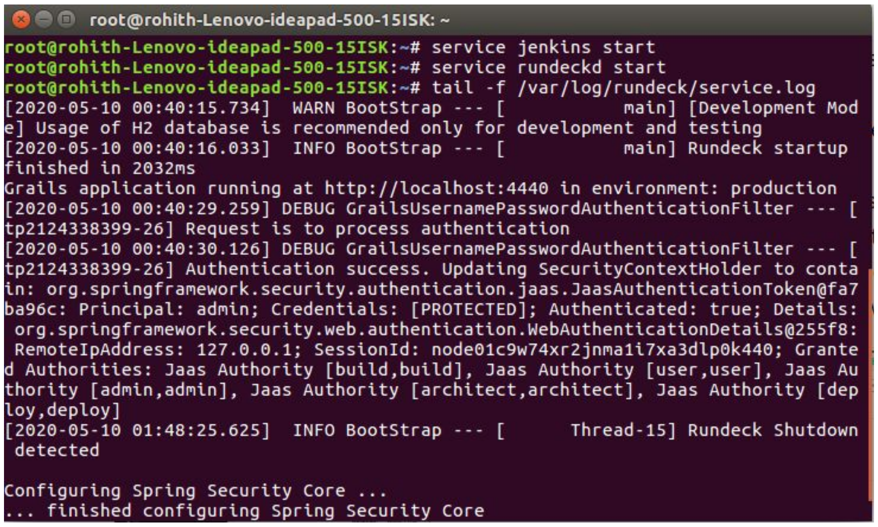
```
20:14:29 latest: Pulling from imt2016072/calculator
20:14:29 Digest: sha256:8c869b74a933196ea7c7c0348ca7b61805c57414f58bb69ecf495c675696da5a
20:14:29 Status: Image is up to date for imt2016072/calculator:latest
20:14:29 docker.io/imt2016072/calculator:latest
```

run the docker image in the container OK

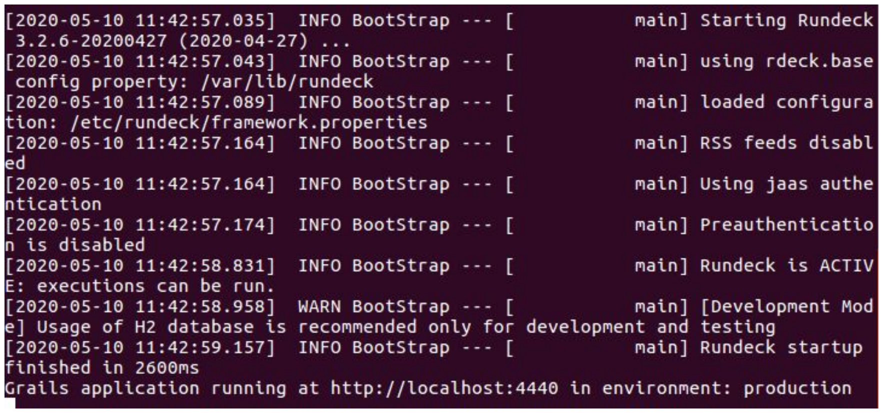
After this pipeline code is complete we build it in jenkins and can see that result in rundeck. We see that the job is successful in the other image. Further upon pushing your latest code into the git repo it gets checked by jenkins and built where this build further triggers docker hub to run the docker file, create an image and push the latest version of it into the repository. The pipeline further triggers rundeck with the help of job ID and the credentials provided thereby running the job and deploying the program. This finishes the DevOps toolchain.

Configure rundeck to use docker by following these commands.

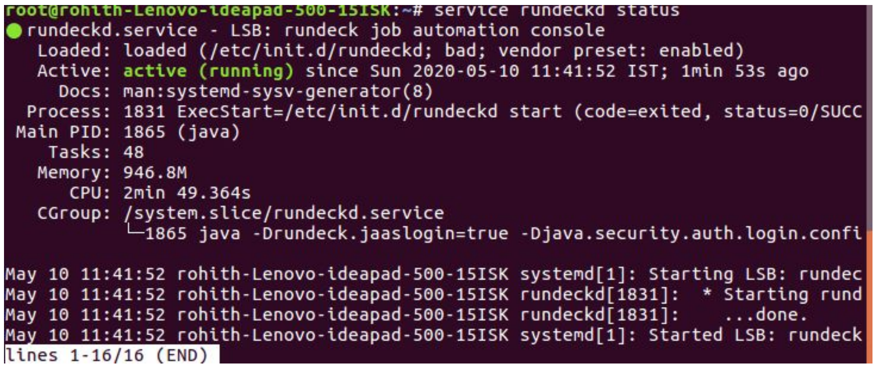
- *sudo groupadd docker*
- *sudo usermod -aG docker $USER*
- *sudo chmod 777 /var/run/docker.sock*
- *systemctl restart docker*
- *systemctl restart jenkins*



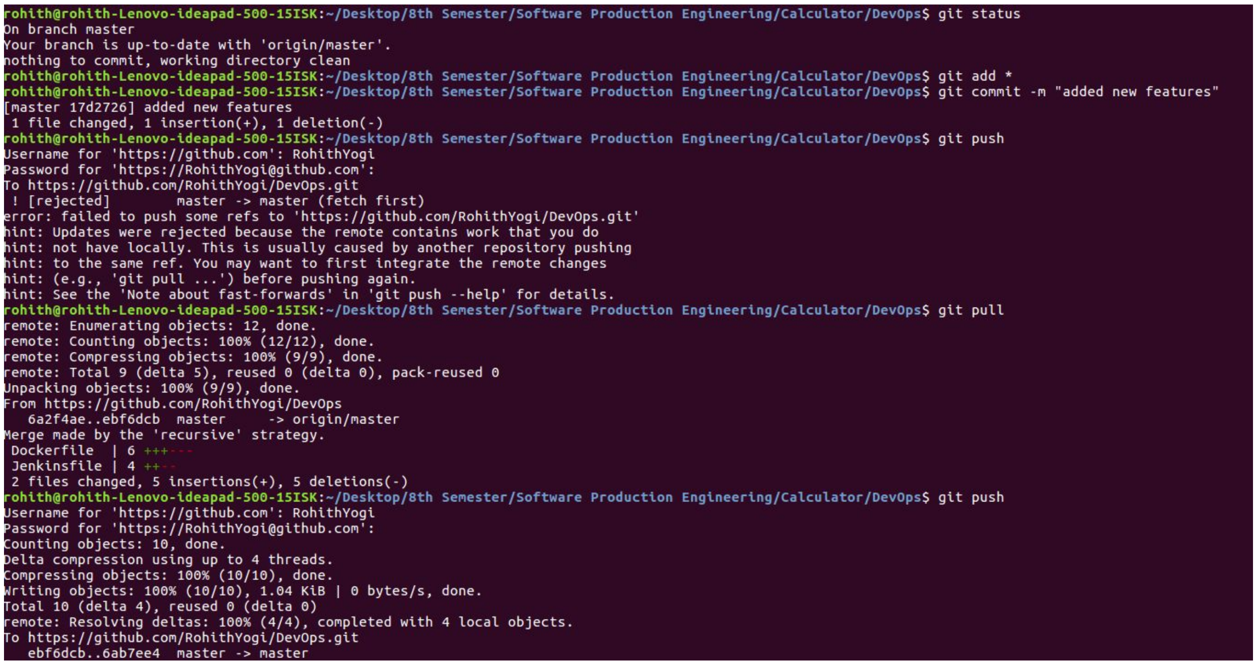
```
root@rohith-Lenovo-ideapad-500-15ISK: ~
root@rohith-Lenovo-ideapad-500-15ISK:~# service jenkins start
root@rohith-Lenovo-ideapad-500-15ISK:~# service rundeckd start
root@rohith-Lenovo-ideapad-500-15ISK:~# tail -f /var/log/rundeck/service.log
[2020-05-10 00:40:15.734]  WARN BootStrap --- [           main] [Development Mod
e] Usage of H2 database is recommended only for development and testing
[2020-05-10 00:40:16.033]  INFO BootStrap --- [           main] Rundeck startup
finished in 2032ms
Grails application running at http://localhost:4440 in environment: production
[2020-05-10 00:40:29.259] DEBUG GrailsUsernamePasswordAuthenticationFilter --- [
tp2124338399-26] Request is to process authentication
[2020-05-10 00:40:30.126] DEBUG GrailsUsernamePasswordAuthenticationFilter --- [
tp2124338399-26] Authentication success. Updating SecurityContextHolder to conta
in: org.springframework.security.authentication.jaas.JaasAuthenticationToken@fa7
ba96c: Principal: admin; Credentials: [PROTECTED]; Authenticated: true; Details:
 org.springframework.security.web.authentication.WebAuthenticationDetails@255f8:
 RemoteIpAddress: 127.0.0.1; SessionId: node01c9w74xr2jnma1i7xa3dlp0k440; Grante
d Authorities: Jaas Authority [build,build], Jaas Authority [user,user], Jaas Au
thority [admin,admin], Jaas Authority [architect,architect], Jaas Authority [dep
loy,deploy]
[2020-05-10 01:48:25.625]  INFO BootStrap --- [      Thread-15] Rundeck Shutdown
 detected

Configuring Spring Security Core ...
... finished configuring Spring Security Core
```



```
[2020-05-10 11:42:57.035]  INFO BootStrap --- [           main] Starting Rundeck
 3.2.6-20200427 (2020-04-27) ...
[2020-05-10 11:42:57.043]  INFO BootStrap --- [           main] using rdeck.base
 config property: /var/lib/rundeck
[2020-05-10 11:42:57.089]  INFO BootStrap --- [           main] loaded configura
tion: /etc/rundeck/framework.properties
[2020-05-10 11:42:57.164]  INFO BootStrap --- [           main] RSS feeds disabl
ed
[2020-05-10 11:42:57.164]  INFO BootStrap --- [           main] Using jaas authe
ntication
[2020-05-10 11:42:57.174]  INFO BootStrap --- [           main] Preauthenticatio
n is disabled
[2020-05-10 11:42:58.831]  INFO BootStrap --- [           main] Rundeck is ACTIV
E: executions can be run.
[2020-05-10 11:42:58.958]  WARN BootStrap --- [           main] [Development Mod
e] Usage of H2 database is recommended only for development and testing
[2020-05-10 11:42:59.157]  INFO BootStrap --- [           main] Rundeck startup
finished in 2600ms
Grails application running at http://localhost:4440 in environment: production
```

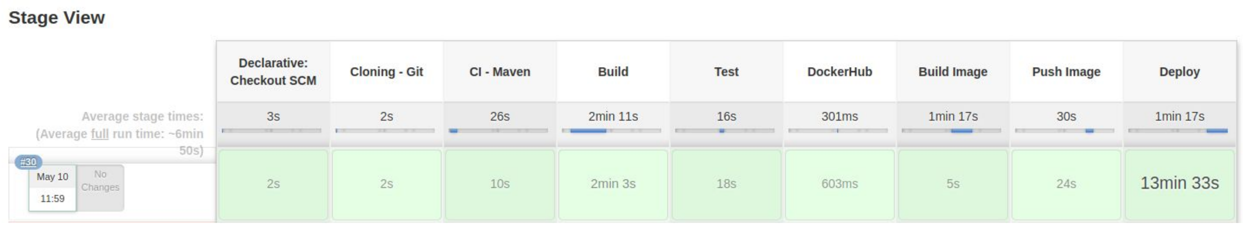


```
root@rohith-Lenovo-ideapad-500-15ISK:~# service rundeckd status
● rundeckd.service - LSB: rundeck job automation console
   Loaded: loaded (/etc/init.d/rundeckd; bad; vendor preset: enabled)
   Active: active (running) since Sun 2020-05-10 11:41:52 IST; 1min 53s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 1831 ExecStart=/etc/init.d/rundeckd start (code=exited, status=0/SUCC
 Main PID: 1865 (java)
    Tasks: 48
   Memory: 946.8M
      CPU: 2min 49.364s
   CGroup: /system.slice/rundeckd.service
           └─1865 java -Drundeck.jaaslogin=true -Djava.security.auth.login.confi

May 10 11:41:52 rohith-Lenovo-ideapad-500-15ISK systemd[1]: Starting LSB: rundec
May 10 11:41:52 rohith-Lenovo-ideapad-500-15ISK rundeckd[1831]:  * Starting rund
May 10 11:41:52 rohith-Lenovo-ideapad-500-15ISK rundeckd[1831]:    ...done.
May 10 11:41:52 rohith-Lenovo-ideapad-500-15ISK systemd[1]: Started LSB: rundeck
lines 1-16/16 (END)
```



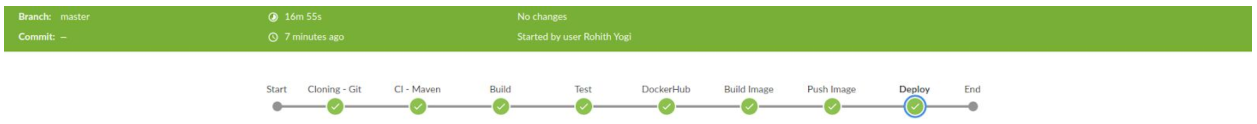
```
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git add *
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git commit -m "added new features"
[master 17d2726] added new features
 1 file changed, 1 insertion(+), 1 deletion(-)
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git push
Username for 'https://github.com': RohithYogi
Password for 'https://RohithYogi@github.com':
To https://github.com/RohithYogi/DevOps.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/RohithYogi/DevOps.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git pull
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 9 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
From https://github.com/RohithYogi/DevOps
   6a2f4ae..ebf6dcb  master     -> origin/master
Merge made by the 'recursive' strategy.
 Dockerfile  | 6 +++---
 Jenkinsfile | 4 ++--
 2 files changed, 5 insertions(+), 5 deletions(-)
rohith@rohith-Lenovo-ideapad-500-15ISK:~/Desktop/8th Semester/Software Production Engineering/Calculator/DevOps$ git push
Username for 'https://github.com': RohithYogi
Password for 'https://RohithYogi@github.com':
Counting objects: 10, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 1.04 KiB | 0 bytes/s, done.
Total 10 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/RohithYogi/DevOps.git
   ebf6dcb..6ab7ee4  master -> master
```

After pushing the additional features the jenkins is configured to check it every 1 day so it checks and builds the latest code following the pipeline created as below :

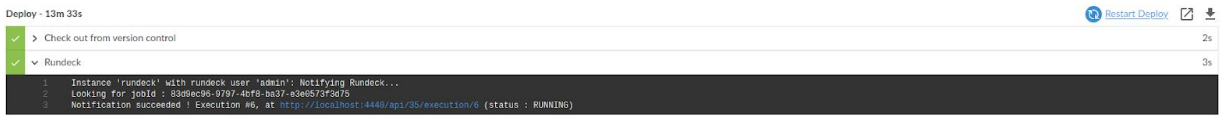


Stage View

	Declarative: Checkout SCM	Cloning - Git	CI - Maven	Build	Test	DockerHub	Build Image	Push Image	Deploy
Average stage times: (Average full run time: ~6min 50s)	3s	2s	26s	2min 11s	16s	301ms	1min 17s	30s	1min 17s
#30 May 10 11:59 No Changes	2s	2s	10s	2min 3s	18s	603ms	5s	24s	13min 33s

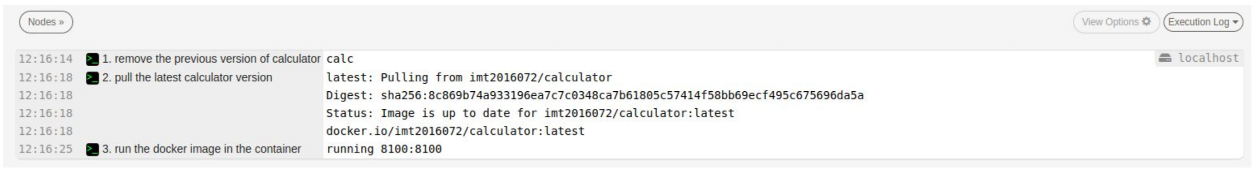


```
Branch: master          16m 55s           No changes
Commit: –               7 minutes ago     Started by user Rohith Yogi
```



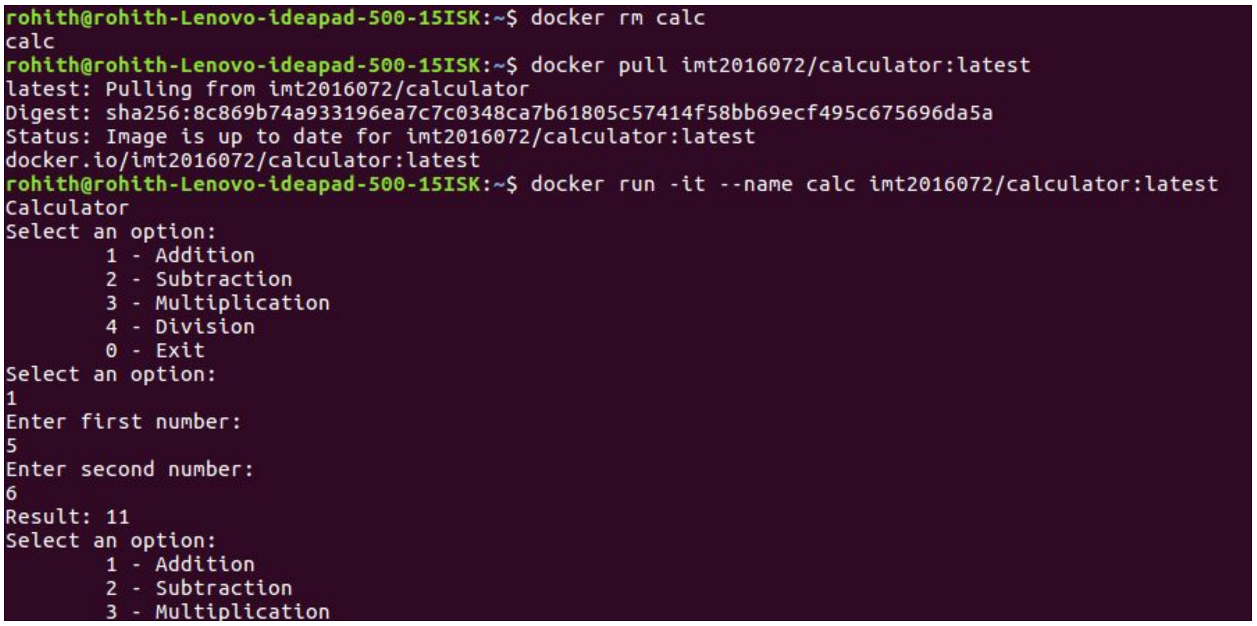
Deploy - 13m 33s

```
Check out from version control                                        2s
Rundeck                                                               3s
Instance 'rundeck' with rundeck user 'admin': Notifying Rundeck...
Looking for jobId : 83d9ec96-9797-4bf8-ba37-e3e0573f3d75
Notification succeeded ! Execution #6, at http://localhost:4440/api/35/execution/6 (status : RUNNING)
```



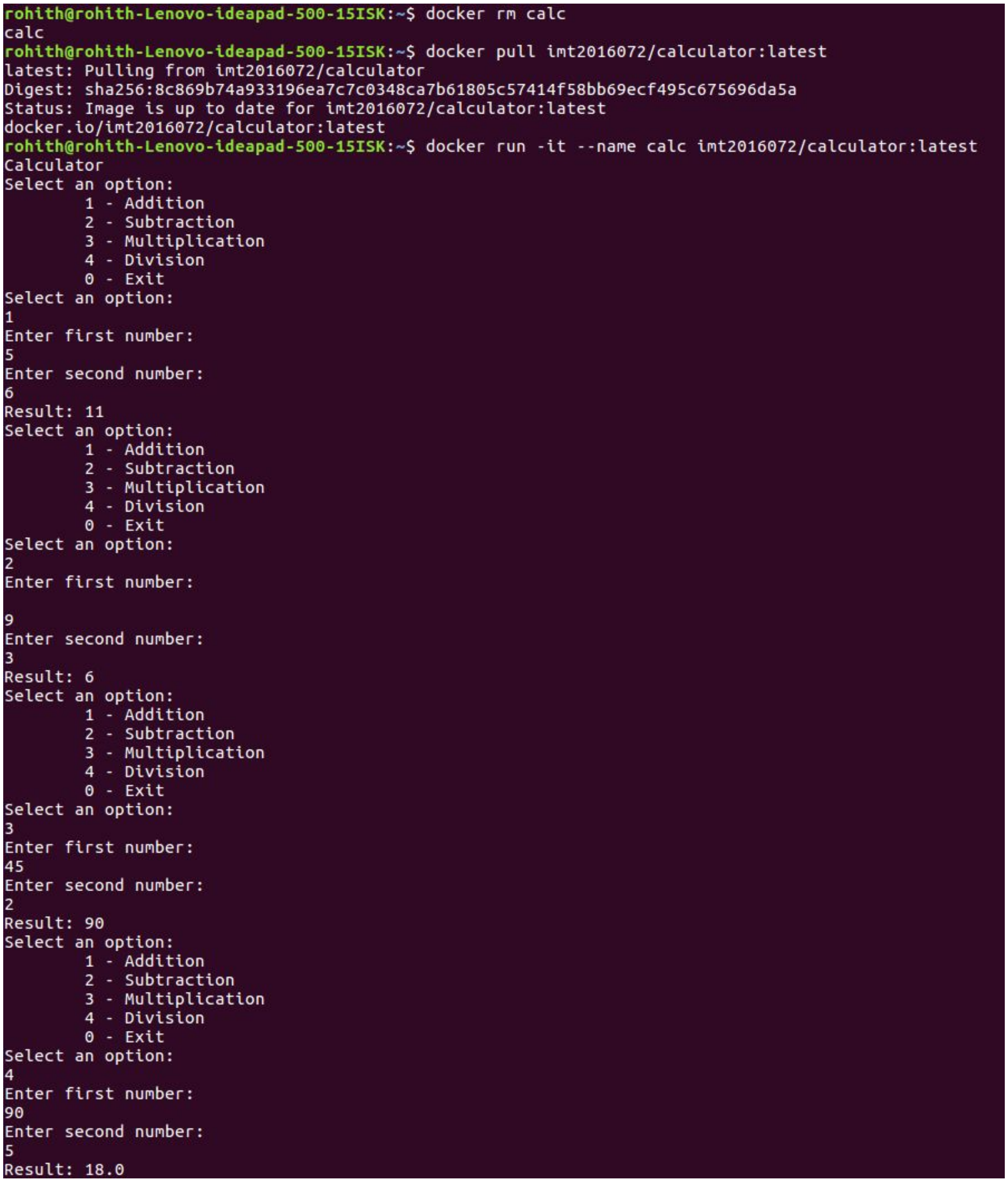
```
12:16:14  1. remove the previous version of calculator  calc                                   localhost
12:16:18  2. pull the latest calculator version         latest: Pulling from imt2016072/calculator
12:16:18                                                Digest: sha256:8c869b74a933196ea7c7c0348ca7b61805c57414f58bb69ecf495c675696da5a
12:16:18                                                Status: Image is up to date for imt2016072/calculator:latest
12:16:18                                                docker.io/imt2016072/calculator:latest
12:16:25  3. run the docker image in the container      running 8100:8100
```

Here I have redirected the running program to a port number but this is just for testing whether we deployed the image properly or not. The real check is pulling the latest image and running it. I have done that here in the below image.



```
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker rm calc
calc
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker pull imt2016072/calculator:latest
latest: Pulling from imt2016072/calculator
Digest: sha256:8c869b74a933196ea7c7c0348ca7b61805c57414f58bb69ecf495c675696da5a
Status: Image is up to date for imt2016072/calculator:latest
docker.io/imt2016072/calculator:latest
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker run -it --name calc imt2016072/calculator:latest
Calculator
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
        4 - Division
        0 - Exit
Select an option:
1
Enter first number:
5
Enter second number:
6
Result: 11
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
```

Further we can add more features to this calculator and check to see if the image is being deployed correctly. The final output looks like this:



```
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker rm calc
calc
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker pull imt2016072/calculator:latest
latest: Pulling from imt2016072/calculator
Digest: sha256:8c869b74a933196ea7c7c0348ca7b61805c57414f58bb69ecf495c675696da5a
Status: Image is up to date for imt2016072/calculator:latest
docker.io/imt2016072/calculator:latest
rohith@rohith-Lenovo-ideapad-500-15ISK:~$ docker run -it --name calc imt2016072/calculator:latest
Calculator
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
        4 - Division
        0 - Exit
Select an option:
1
Enter first number:
5
Enter second number:
6
Result: 11
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
        4 - Division
        0 - Exit
Select an option:
2
Enter first number:

9
Enter second number:
3
Result: 6
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
        4 - Division
        0 - Exit
Select an option:
3
Enter first number:
45
Enter second number:
2
Result: 90
Select an option:
        1 - Addition
        2 - Subtraction
        3 - Multiplication
        4 - Division
        0 - Exit
Select an option:
4
Enter first number:
90
Enter second number:
5
Result: 18.0
```

References

https://www.edureka.co/community/55640/jenkins-docker-docker-image-jenkins-pipeline-docker-registry

https://blog.koley.in/2018/install-and-setup-rundeck-on-ubuntu

https://plugins.jenkins.io/rundeck/

https://www.edureka.co/community/7764/trying-docker-jenkins-pipeline-facing-jenkins-pipeline-socket

https://boxboat.com/2017/05/30/jenkins-blue-ocean-pipeline/

https://cloudacademy.com/course/jenkins-cicd-advanced/blue-ocean-build-package-and-publish-docker-image-docker-hub/

https://forums.docker.com/t/error-response-from-daemon-get-https-registry-1-docker-io-v2/23741/21

https://medium.com/@khandelwal12nidhi/ci-cd-with-jenkins-and-ansible-e9163d4a6e82