

International Institute of Information Technology, Bangalore



Academic Year 2019-20

Project Report on Devops Calculator Assignment

Submitted By

Shivani Shrivastava

MT2019104

GitHub : <https://github.com/ShrivastavaShivani>

DockerHub : <https://hub.docker.com/u/shrivastavashivani>

Introduction

The project aims at building DevOps Continuous Integration and Continuous Deployment Pipeline. So, this project has a very simple source code for Calculator. Following tools used to build this project:

1. IntelliJ
2. AWS EC2
3. Github
4. Webhook
5. Jenkins
6. Docker
7. Docker Hub
8. Rundeck

WorkFlow of Project

The project starts with writing source code in intelliJ. Develop a Maven project on Java Language. The code is then pushed to github from where jenkins pulls it using git plugin and builds the jar file using Maven Integration plugin. The jar file is then tested using JUnit plugin. Jenkins then uses Docker Build and Publish plugin to make docker images which are then pushed to Docker Hub. Jenkins then finally invokes Rundeck. Rundeck deletes all the previous images and containers related to the new container in which it's going to deploy new image and the it pulls docker image from Docker Hub and deploys the image on any other docker container (same in our case).

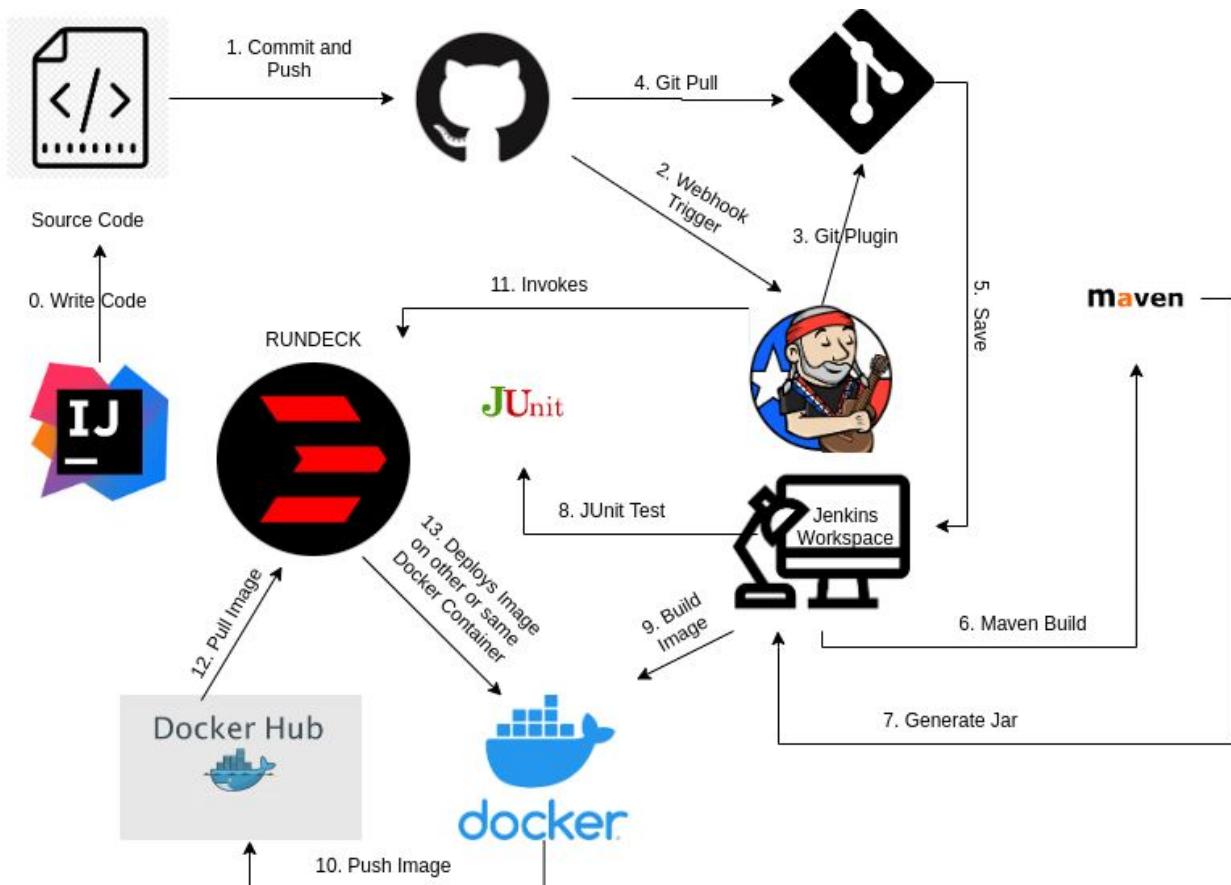


Fig 1. Workflow

Source Code

The very first step in making a devops project is coding. We used IntelliJ IDEA for coding the calculator program in java. For testing utilities we made a MAVEN project. Install IntelliJ on Ubuntu 18.04 using following commands

```
$sudo snap install intellij-idea-ultimate --classic
```

Start a new maven project and select java version 8.

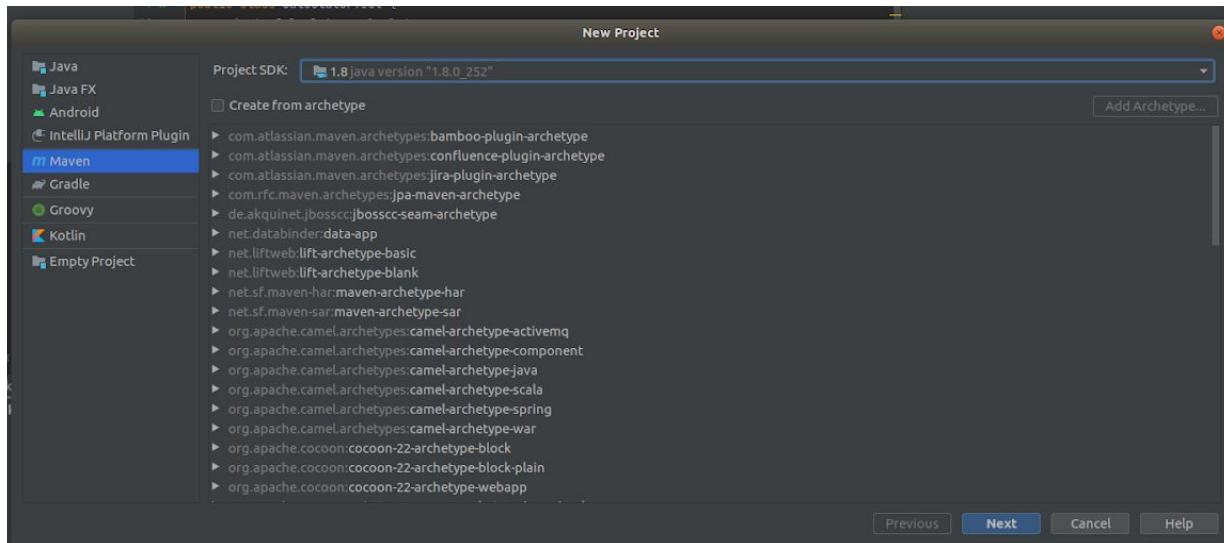


Fig 2. IntelliJ project config

Give your project a suitable name, location, group_id, artifact_id and version.

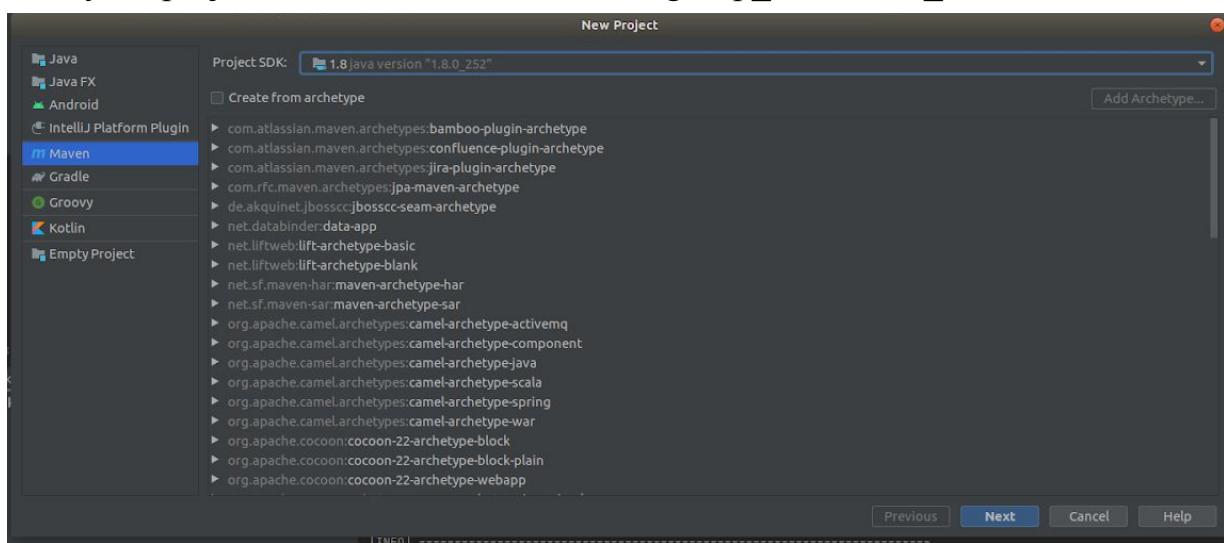
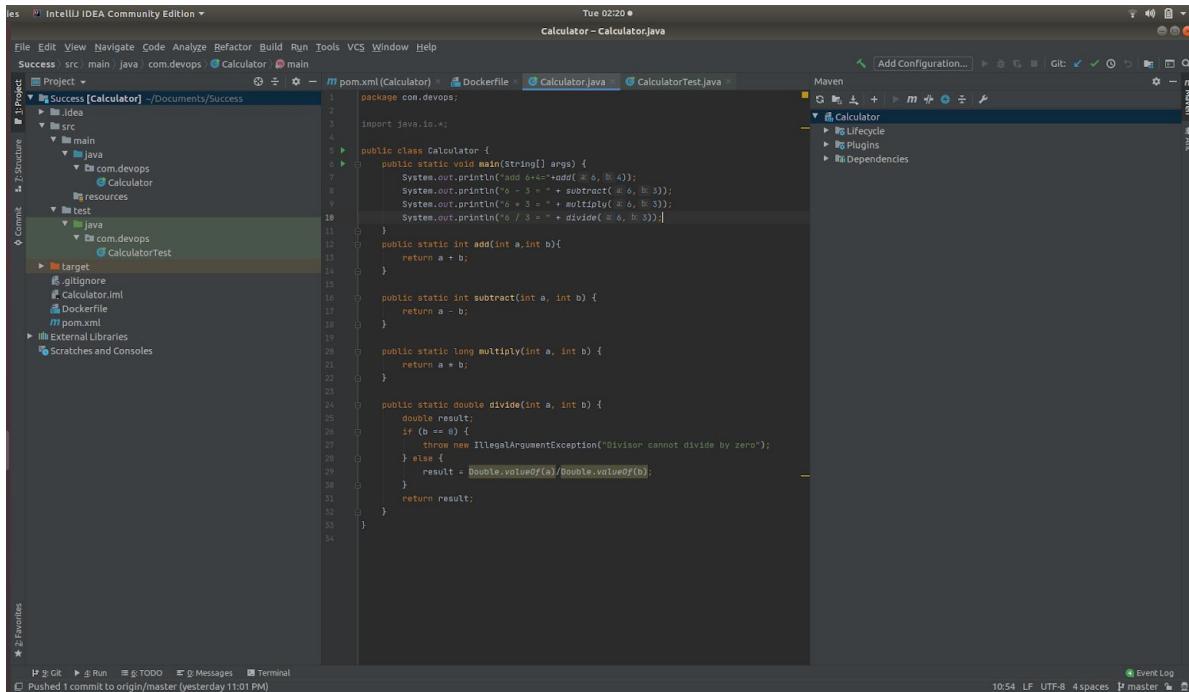


Fig 3. IntelliJ project config

Create two new java classes, Calculator.java and CalculatorTest.java and write code inside it. Remember to write code only for addition operation, as the rest of the 3 will be pushed after building the pipeline as a part of Continuous Integration. The final source code is as follows:



```

package com.devops;

import java.util.*;

public class Calculator {
    public static void main(String[] args) {
        System.out.println("add = " + add(6, 5));
        System.out.println("6 - 5 = " + subtract(6, 5));
        System.out.println("6 * 5 = " + multiply(6, 5));
        System.out.println("6 / 5 = " + divide(6, 5));
    }

    public static int add(int a, int b) {
        return a + b;
    }

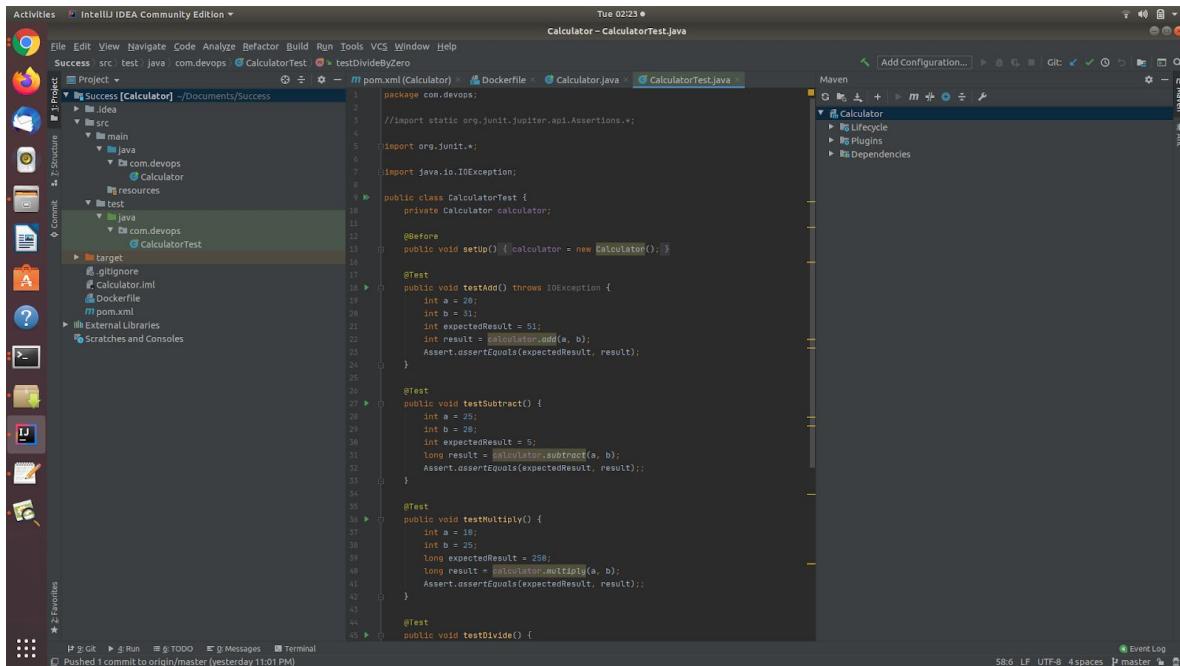
    public static int subtract(int a, int b) {
        return a - b;
    }

    public static long multiply(int a, int b) {
        return a * b;
    }

    public static double divide(int a, int b) {
        double result;
        if (b == 0) {
            throw new IllegalArgumentException("Divisor cannot divide by zero");
        } else {
            result = Double.valueOf(a) / Double.valueOf(b);
        }
        return result;
    }
}

```

Fig 4. Calculator.java



```

package com.devops;

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

import java.io.IOException;

public class CalculatorTest {
    private Calculator calculator;

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

    @Test
    public void testAdd() throws IOException {
        int a = 28;
        int b = 31;
        int expectedResult = 51;
        int result = calculator.add(a, b);
        Assert.assertEquals(expectedResult, result);
    }

    @Test
    public void testSubtract() {
        int a = 28;
        int b = 28;
        int expectedResult = 0;
        long result = calculator.subtract(a, b);
        Assert.assertEquals(expectedResult, result);
    }

    @Test
    public void testMultiply() {
        int a = 10;
        int b = 10;
        long expectedResult = 250;
        long result = calculator.multiply(a, b);
        Assert.assertEquals(expectedResult, result);
    }

    @Test
    public void testDivide() {
        int a = 10;
        int b = 0;
        double expectedResult = 0.0;
        double result = calculator.divide(a, b);
        Assert.assertEquals(expectedResult, result);
    }
}

```

Fig 5. Calculator.java

The next job is to look at the services we will need in our project and write the correct pom.xml file. This is the which contains all the dependencies we require to build our project. The very basic need of our project is to have JUnit and maven dependencies included in pom.xml. Another important file is Dockerfile which Jenkins will need during Docker Build and Publish step. Jenkins will read Dockerfile to build and push docker image of our project to Docker Hub.

```

<groupId>com.devops</groupId>
<artifactId>Calculator</artifactId>
<version>1.0-SNAPSHOT</version>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
</properties>

<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-surefire-plugin</artifactId>
            <version>3.0.0-M4</version>
        </plugin>
        <plugin>
            <!-- Build an executable JAR -->
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.2.0</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>com.devops.Calculator</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>

```

Fig 6. pom.xml

```

FROM openjdk:8-jre-alpine as target
RUN mkdir -p /opt/app
ENV Project_reside /opt/app

COPY /target/Calculator-1.0-SNAPSHOT.jar $Project_reside/calc.jar

WORKDIR $Project_reside

EXPOSE 8882
ENTRYPOINT ["java", "-jar", "./calc.jar"]

```

Fig 7. Dockerfile

AWS EC2 Instance

To make the pipeline reachable to the outer world, I decided to build the devops pipeline through aws (Amazon Web Services). As this project is just to learn DevOps tools the ec2 instance is taken in free tier. By following the steps given below setup two ec2 instances.

1. Make an aws account. Some services are free for a duration of one year. (Do not worry about the credit or debit card details as after the full usage of the free tier they do not charge without notifying.)
2. Log in to your aws account, hover on services and click on EC2 under compute services.
3. Launch a new instance and follow as shown in pictures below.

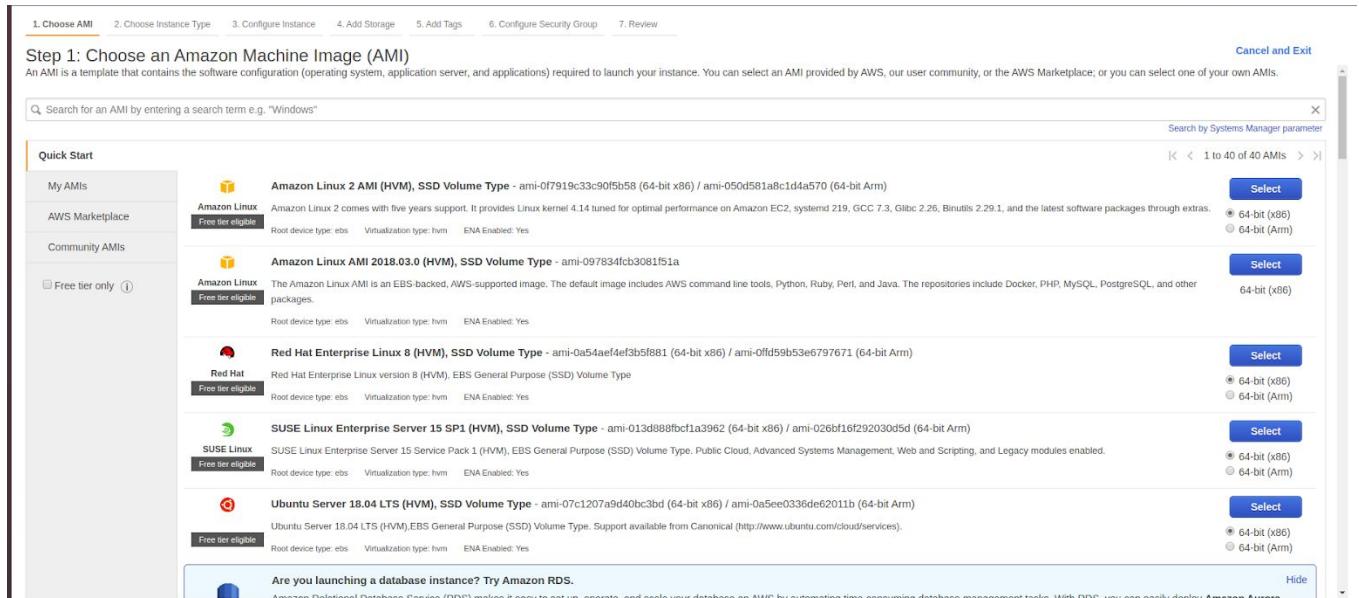


Fig 8. AWS EC2 configuration

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
General purpose	t3a.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3a.micro	2	1	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3a.small	2	2	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3a.medium	2	4	EBS only	Yes	Up to 5 Gigabit	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Fig 9. Select Instance Type

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security group name:

Description:

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop

Add Rule

Warning: Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel Previous Review and Launch

Fig 10. Create a new Network Security Group

Add inbound rules for the ports into the security groups which will be accessed from this instance for building pipeline. This EC2 instance is for Jenkins and Docker working.

Inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	Custom	0.0.0.0/0
HTTP	TCP	80	Custom	::/0
Custom TCP	TCP	8080	Custom	0.0.0.0/0
Custom TCP	TCP	8080	Custom	::/0
SSH	TCP	22	Custom	0.0.0.0/0
Custom TCP	TCP	4440	Custom	::/0
Custom TCP	TCP	4440	Custom	0.0.0.0/0

Add rule

Fig 11. Add inbound rules for NSG

Step 7: Review Instance Launch

Security Groups

Security Group ID	Name
sg-06124480b67ef5d0ba	launch-one-devops
sg-067f6fe5e206554cba	jenkins-security-group
sg-cedab4b6	default

All selected security groups inbound rules

Type	Protocol
SSH	TCP
HTTP	TCP
HTTP	TCP
Custom TCP Rule	TCP
Custom TCP Rule	TCP
SSH	TCP
All traffic	All

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair
Key pair name: rundeck-ec2
Download Key Pair

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location**. You will not be able to download the file again after it's created.

Cancel Launch Instances

Fig 12. Download key pair file

Do not forget to download the key pair file before launching the instance.
Perform the same steps to create another EC2 instance which will be for Rundeck but this time no need to create a new security group just select it.

SSH Connection

To connect to aws EC2 instance from local machine follow the given steps:

1. Open an SSH client i.e. terminal inside the folder where the key-pair rundeck-ec2.pem file is present.
2. The wizard automatically detects the key used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command
`$ chmod 400 rundeck-ec2.pem`
4. Connect to your instance using its Public DNS: (xyz is DNS Public IPv4)
ec2-xyz.us-east-2.compute.amazonaws.com with the command

```
$ ssh -i "rundeck-ec2.pem"  
ec2-user@ec2-xyz.us-east-2.compute.amazonaws.com
```

Github

Go to <https://github.com> and sign up in github. Go to intelliJ and allow VCS - Version Control Services and add github credentials in it. After that, Commit and Push the whole intelliJ project on Github. You might want to remove an error inside jenkins regarding Github plugin at that time you might need to install git in EC2 instance. You can do that using following command

```
$ sudo yum install git
```

Docker

Install docker using following commands:

```
$ sudo yum update -y //updates system  
$ sudo amazon-linux-extras install docker //install docker  
$ sudo service docker start //starts docker  
$ sudo usermod -a -G docker ec2-user //adds ec2-user in docker group
```

After adding ec2-user to docker make sure you reboot your instance.

Docker Hub

Go to <https://hub.docker.com/> and make an account in Docker Hub. Do not create any repository manually. Just create an account and note down its credentials.

Jenkins

Install jenkins in ec2 instance using following commands:

```
$sudo yum update -y //update system  
$sudo yum install java-1.8.0-openjdk-devel //install jdk  
$curl --silent --location  
http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo | sudo tee  
/etc/yum.repos.d/jenkins.repo //get GPG key  
$sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key  
//add repo to system  
$sudo yum install jenkins //install jenkins  
$sudo systemctl start jenkins //start jenkins  
$systemctl status jenkins.service // check status
```

Now, our machine is ready with jenkins installed in it. Open jenkins in the web browser at http://xyz:8080. Port no. 8080 is the default port for jenkins.

Copy the initial admin password using following command to Unlock Jenkins:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Install initial suggested plugins and wait for the download to complete.

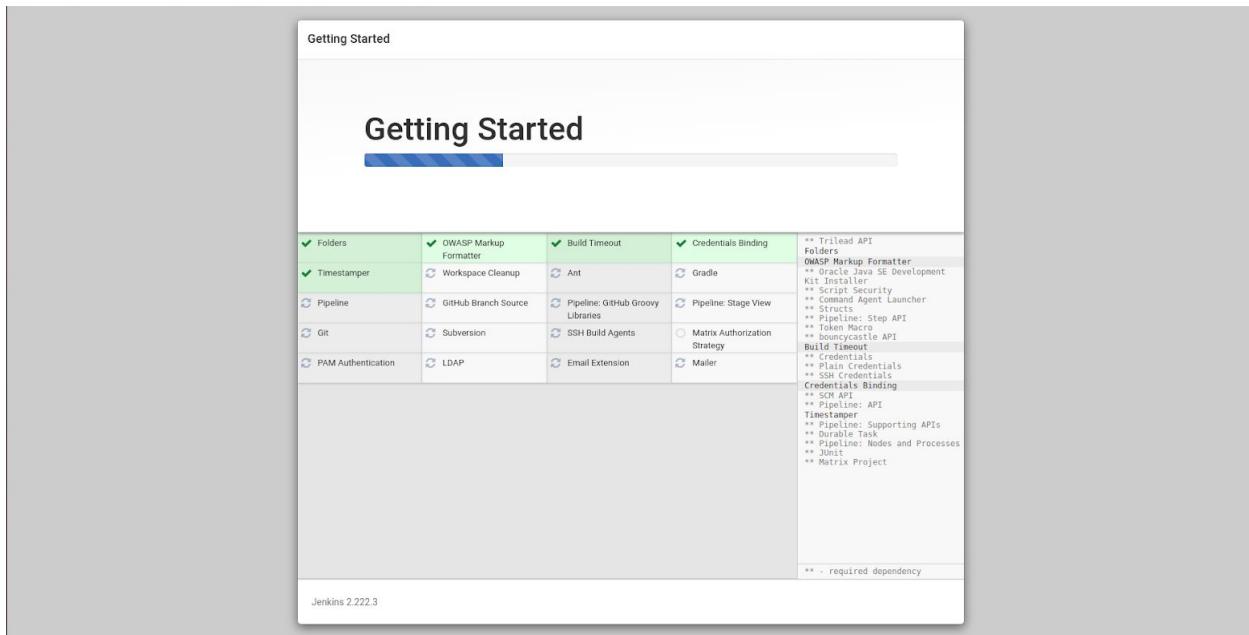


Fig 13. Getting Started with Jenkins

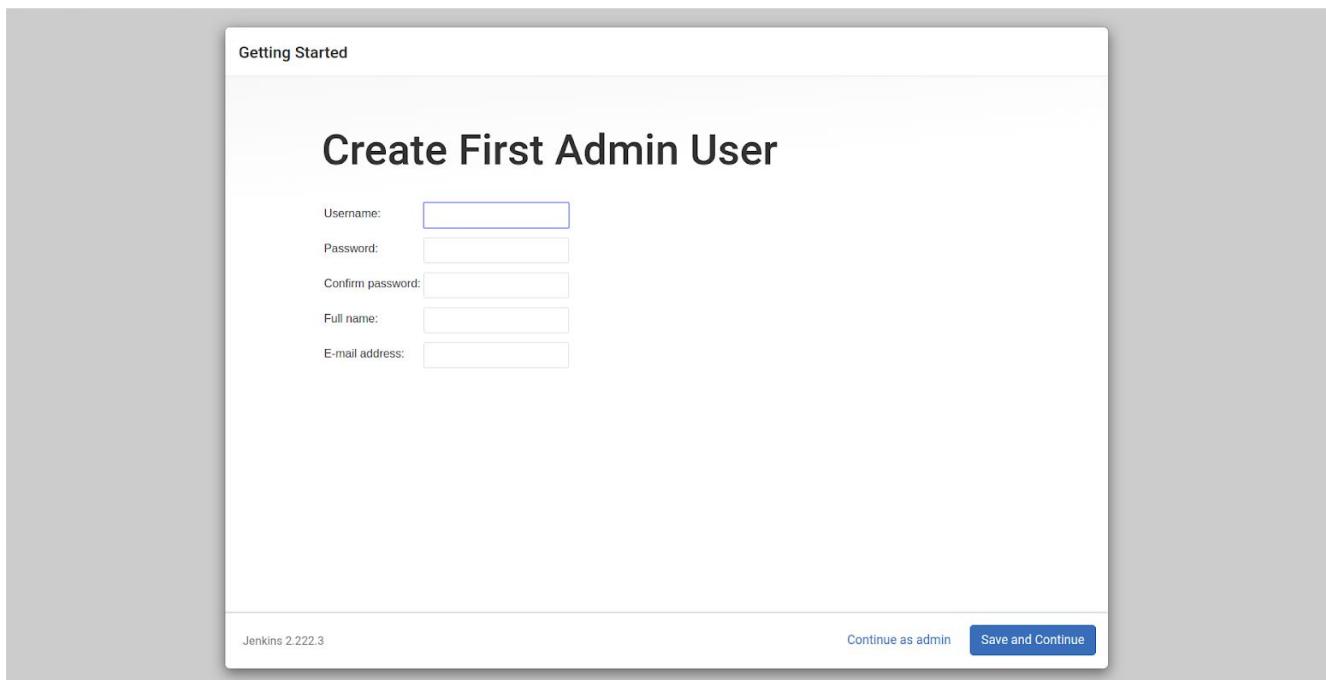


Fig 14. Jenkins User Account Setup

Save and Finish !

Your jenkins is now ready to use.

Plugins

Next step is to install all the necessary plugins which we will need in our project. Click on Manage Jenkins and go to Manage Plugin. Under the Available plugin tab check the following plugins and install them.

1. CloudBees Docker Build and Publish plugin
2. Docker plugin
3. Docker-build-step
4. Git plugin
5. Github plugin
6. Junit plugin
7. Maven Integration plugin
8. Pipeline plugin
9. Pipeline : GitHub Groovy Libraries
10. Rundeck plugin

Webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, Webhook sends a POST request to each of the URLs you provide.

For setting up Webhook follow given steps:

1. Login to Github
2. Go inside the repository you want to give access to aws to pull files from.
3. Click on the Settings tab and then go to the Webhook section.
4. Add webhook and enter the url where jenkins will be hosted i.e. `http://xyz:8080` (remember xyz is Public DNS IPv4)
5. As soon as you add the webhook, github will make a dummy post call to the jenkins webhook. Make sure the call is successful to the jenkins. It can be verified from the status icon on left of added webhook which should be a green tick.

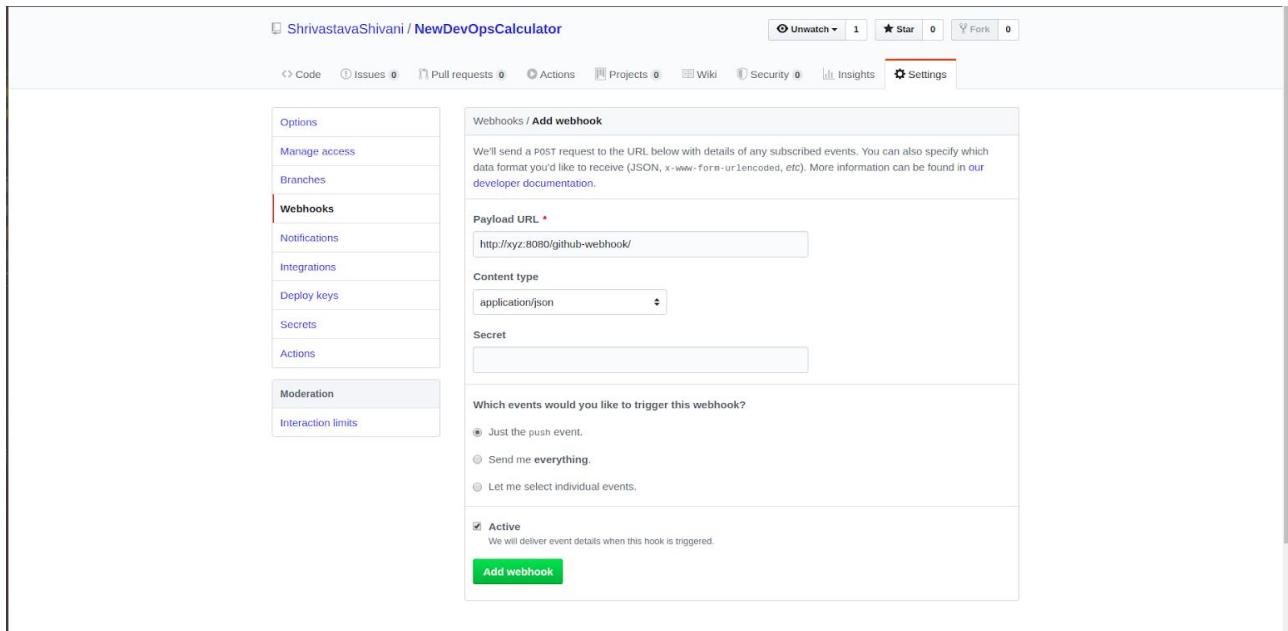


Fig15. Setting up Webhook

Creating Items

Create 3 new items in Jenkins -

1. Calculator-1-Build
2. Calculator-2-Test
3. Calculator-3-Main

Configure each one of them as shown in figures below

For configuring Calculator-1-Build project, you need to give access to your github account. Add correct username and password and proceed. In the Build step below, you might need to add Maven for Invoking top level maven targets to build jar file from the dockerfile commands. This can be done by going to Manage Jenkins then into Global Tool Configuration and then in Maven section click on Add Maven. Give a name to your maven version (mine - Maven_latest) and select one version out of the drop down bar. Click on save and you are done!

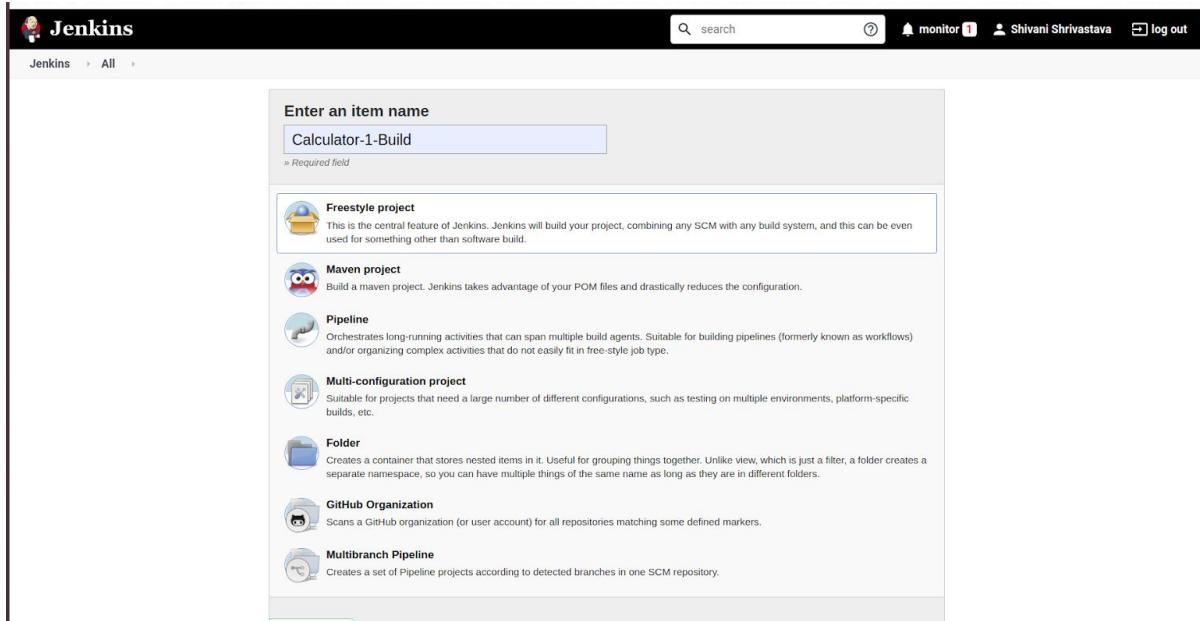


Fig16. Calculator-1-Build setup-1

A screenshot of the Jenkins 'General' configuration page for the 'Calculator-1-Build' project. The 'General' tab is selected, showing the following details:

- Description: A devops Project
- Project url: https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git
- Checkboxes (unchecked): Commit agent's Docker container, Define a Docker template, Discard old builds, This build requires lockable resources, This project is parameterized, Throttle builds, Disable this project, Execute concurrent builds if necessary.
- Buttons: Advanced..., Save, Apply

The 'Source Code Management' section shows 'None' selected. The bottom right of the page has 'Advanced...' and 'Save' buttons.

Fig17. Calculator-1-Build setup-2

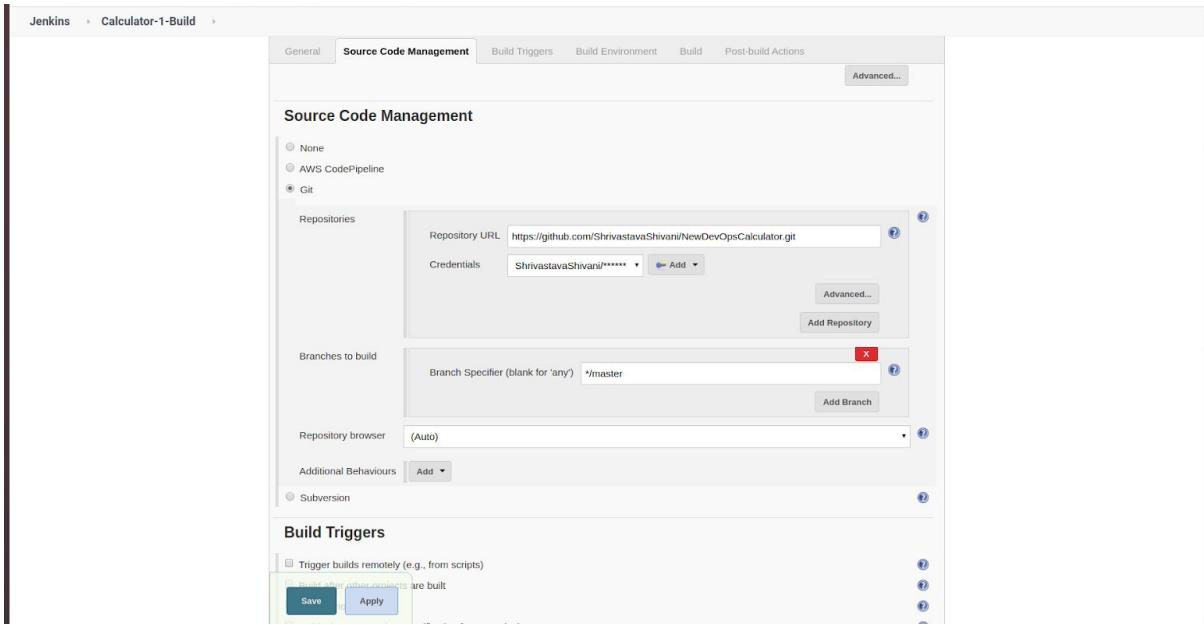


Fig18. Calculator-1-Build setup-3

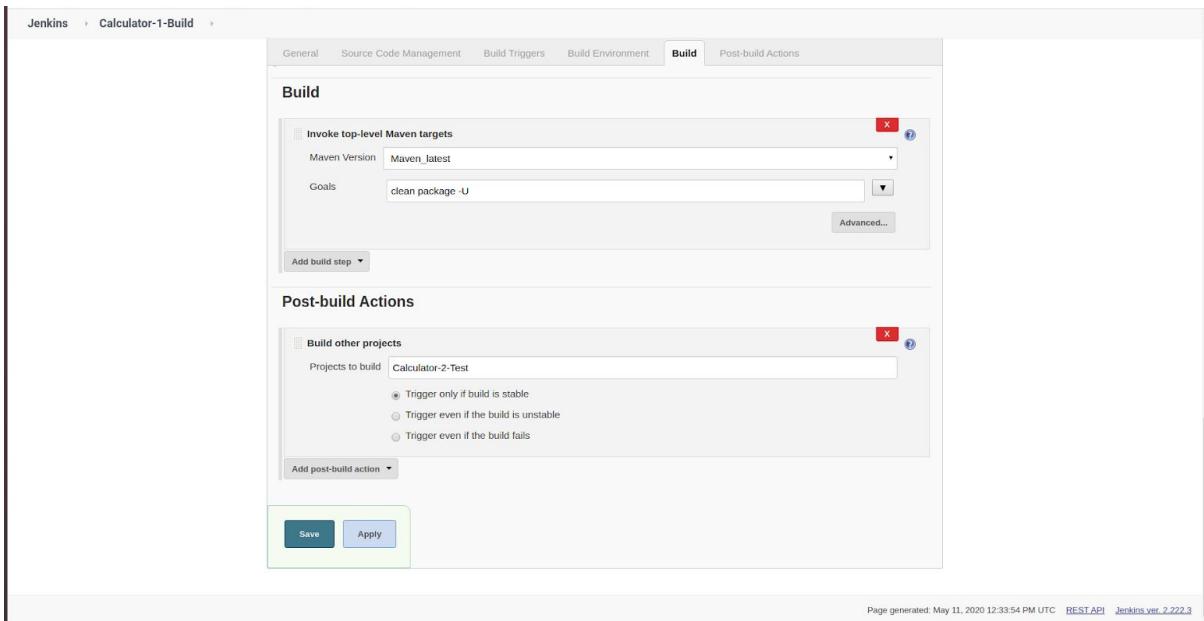


Fig19. Calculator-1-Build setup-4

Click on Apply and then Save. To check individually, you can now click on Build Now and see if Console Output is Success.

Now configure the Calculator-2-Test project as shown in figures.

The screenshot shows the Jenkins General configuration page for the 'Calculator-2-Test' project. The 'General' tab is selected. In the 'Description' field, the text 'Test Project' is entered. Under the 'GitHub project' section, the 'Project url' is set to 'https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git'. There are several checkboxes for build triggers and behaviors, none of which are checked. At the bottom of the General tab, there are 'Save' and 'Apply' buttons.

Fig20. Calculator-2-Test setup-1

The screenshot shows the Jenkins Source Code Management and Build Triggers configuration page for the 'Calculator-2-Test' project. The 'Source Code Management' tab is selected. Under 'Source Code Management', the 'Git' option is chosen, and the 'Repository URL' is set to 'https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git' with credentials 'ShrivastavaShivani*****'. The 'Branches to build' section has a branch specifier of '*/master'. Under 'Build Triggers', the 'Build periodically' option is selected. At the bottom of the Source Code Management tab, there are 'Save' and 'Apply' buttons.

Fig21. Calculator-2-Test setup-2

The screenshot shows the Jenkins configuration page for the 'Calculator-2-Test' project. The 'Build' section contains a step to 'Invoke top-level Maven targets' with 'Goals' set to 'test'. The 'Post-build Actions' section includes 'Build other projects' for 'Calculator-3-Main' and 'Publish JUnit test result report' with 'Test report XMLs' set to 'target/surefire-reports*.xml'. Buttons for 'Save' and 'Apply' are visible at the bottom.

Fig22. Calculator-2-Test setup-3

Apply and Save. Build using build now and check if it's giving success.

Let us now configure the third Calculator-3-Main project

The screenshot shows the Jenkins configuration page for the 'Calculator-3-Main' project. The 'General' tab has 'Description' set to 'Main Project' and 'GitHub project' checked with 'Project url' set to 'https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git'. Under 'Source Code Management', 'None' is selected. Buttons for 'Save' and 'Apply' are visible at the bottom.

Fig23. Calculator-3-Main setup-1

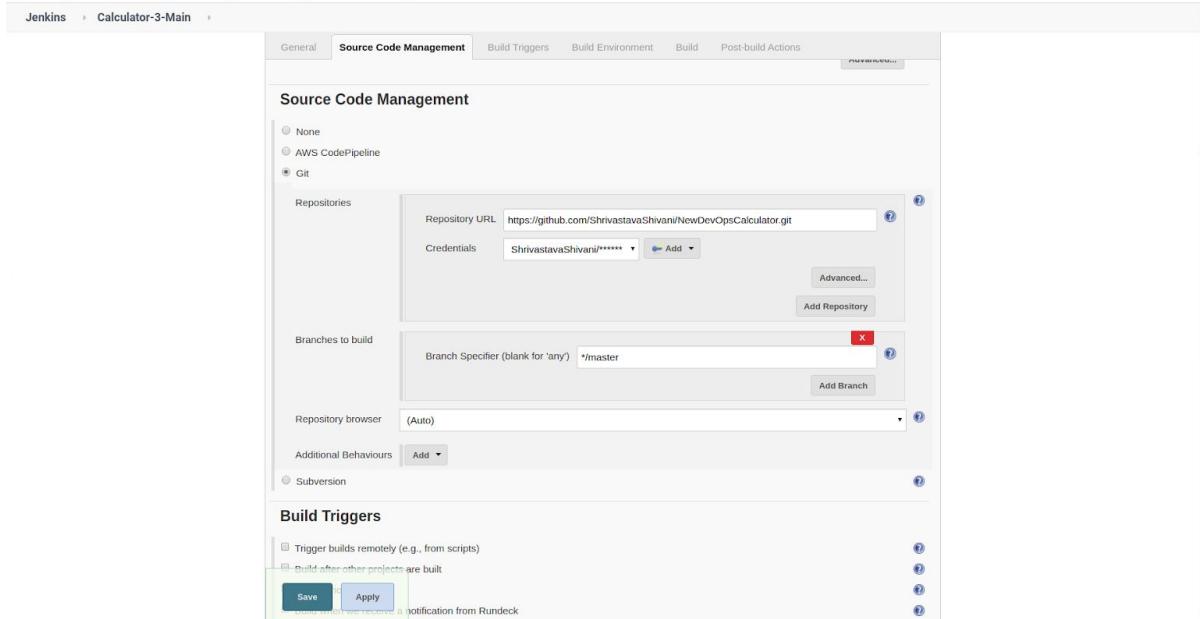


Fig24. Calculator-3-Main setup-2

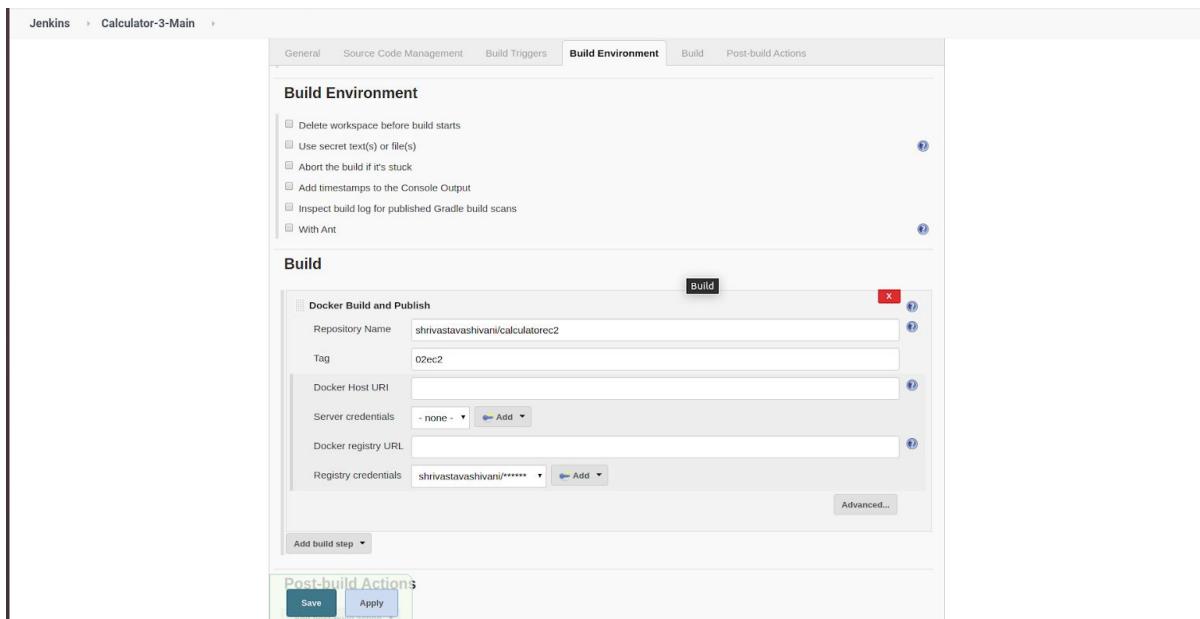


Fig25. Calculator-3-Main setup-3

In Docker Build and Publish, Repository Name should be your docker hub username followed by the name you want to build a repository for. Also, new credentials need to be added here which will contain username and password of docker hub.

Before adding Post-Build Actions, you must install and configure rundeck and add a job as shown after this section. This will assign a Job identifier to the rundeck job which will be provided in the Job Identifier. Also, Rundeck instance need to be added in Manage Jenkins before this step.

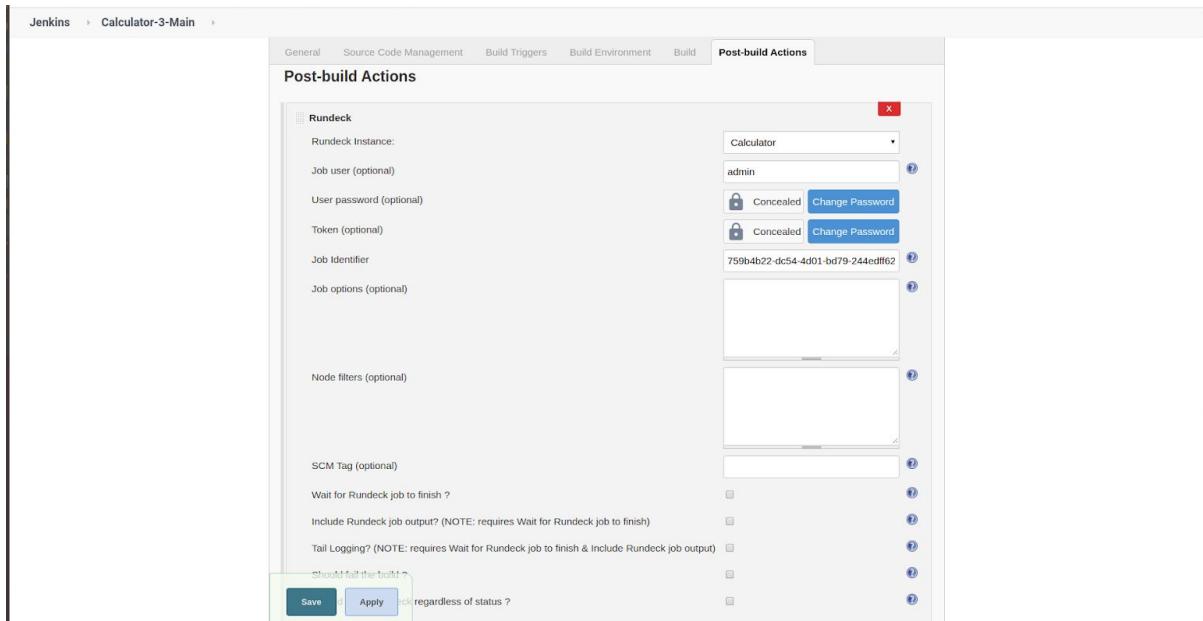


Fig26. Calculator-3-Main setup-4

Rundeck

Use another ec2 instance rundeck-ec2 for installing Rundeck. Login in terminal with ssh command for the new rundeck ec-2 instance. Install rundeck in this instance using following commands:

```
$Sudo yum update
$sudo rpm -Uvh http://repo.rundeck.org/latest.rpm
$sudo yum install rundeck java
$Sudo yum update rundeck
$Start rundeck
$sudo service rundeckd start
```

Now after installation, check if the Java version is 11.0.7. If yes, then install Java with version 1.8.0 using following command as rundeck runs on Java8 support.

```
$sudo yum install java-1.8.0-openjdk
```

Switch between java versions using following command:

```
$sudo update-alternatives --config java
```

Reboot system.

Before login to Rundeck dashboard you have to update public dns name at parameter name “grails.serverURL” in the configuration file /etc/rundeck/rundeck-config.properties. For doing so use command:

```
$sudo nano /etc/rundeck/rundeck-config.properties
```

And then edit line -

```
grails.serverURL = http://ec2-xyz.compute-1.amazonaws.com:4440
```

After editing press Ctrl+S to save and Ctrl+X to exit.

Open Rundeck in <http://ec2-xyz.compute-1.amazonaws.com:4440> url in web browser.

Enter Username : admin

Password : admin

Create new Project in Rundeck as shown below:

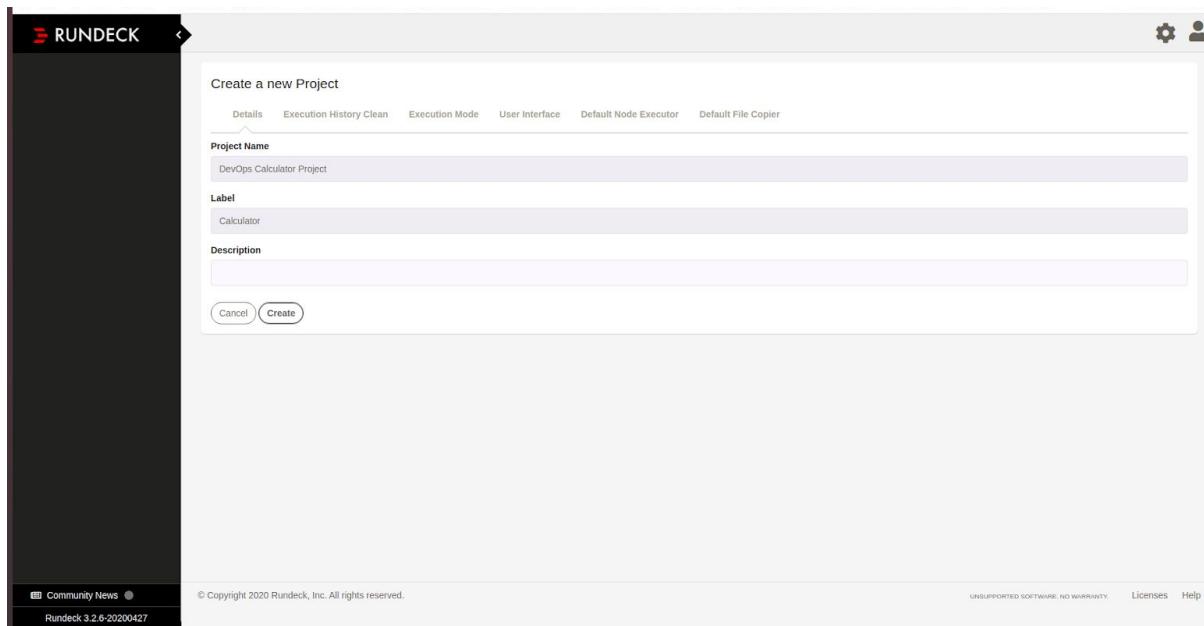


Fig 27. Rundeck Create New Project

Now, add a new Job to this rundeck project. This job will pull image from dockerhub and make run it in docker container for deployment.

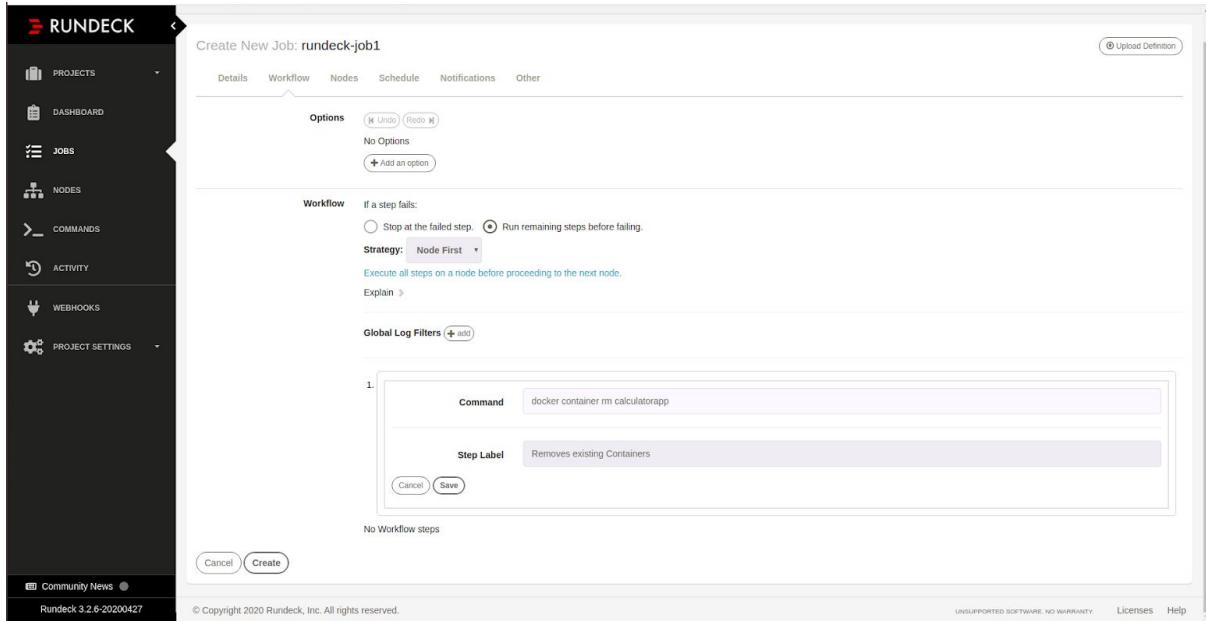


Fig28. Rundeck job setup-1

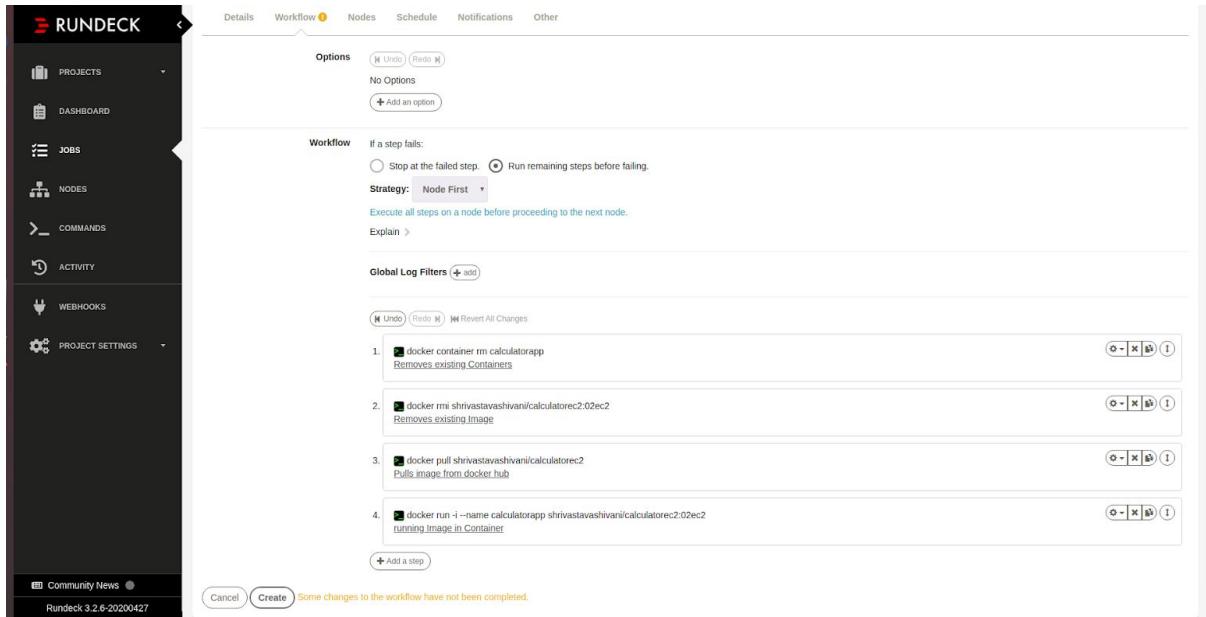


Fig29. Rundeck job setup 2

The first two commands in rundeck job are to remove any previous image of docker or container with the same name as the one rundeck will create after pulling image from docker hub. This is why it is important to check Run remaining steps before failing.

Adding Rundeck Instance in Jenkins

We will need to add a rundeck instance in jenkins in order to configure post build actions of item 3 Calculator-3-Main. Go to Manage Jenkins and then in Configure System and add Rundeck in Rundeck section as shown below:

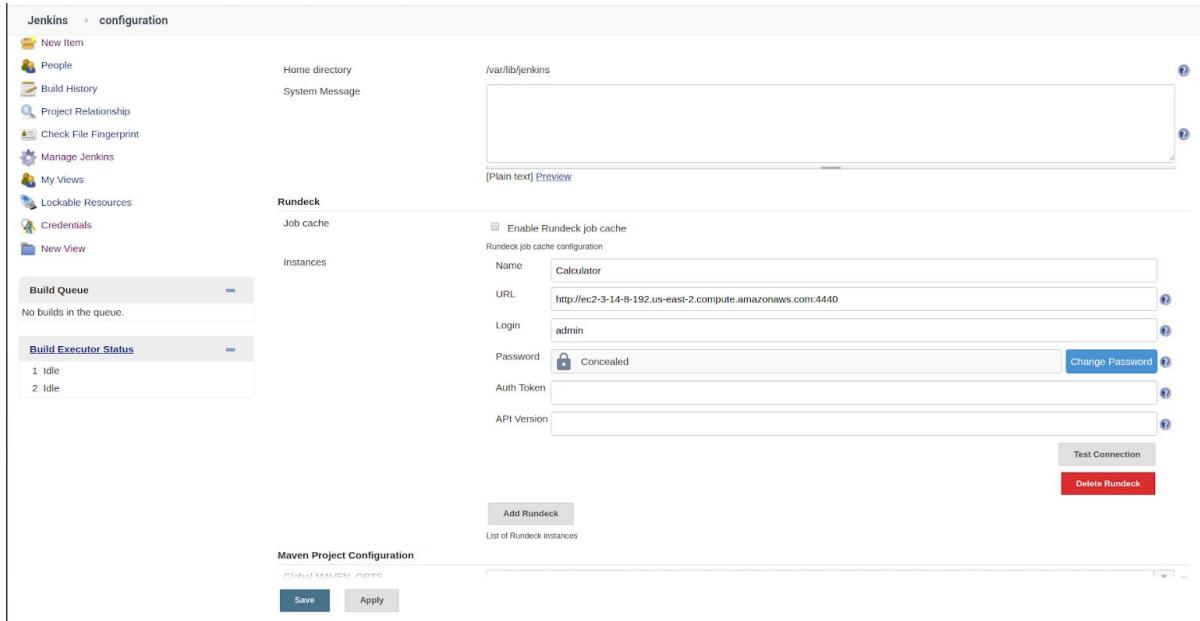


Fig 30. Rundeck instance in Jenkins

Build Pipeline

Install Build Pipeline plugin from Manage Jenkins -> Manage Plugins. After this on the dashboard, click on ‘+’ for creating pipeline view. Add view name and select Build Pipeline. Click OK.

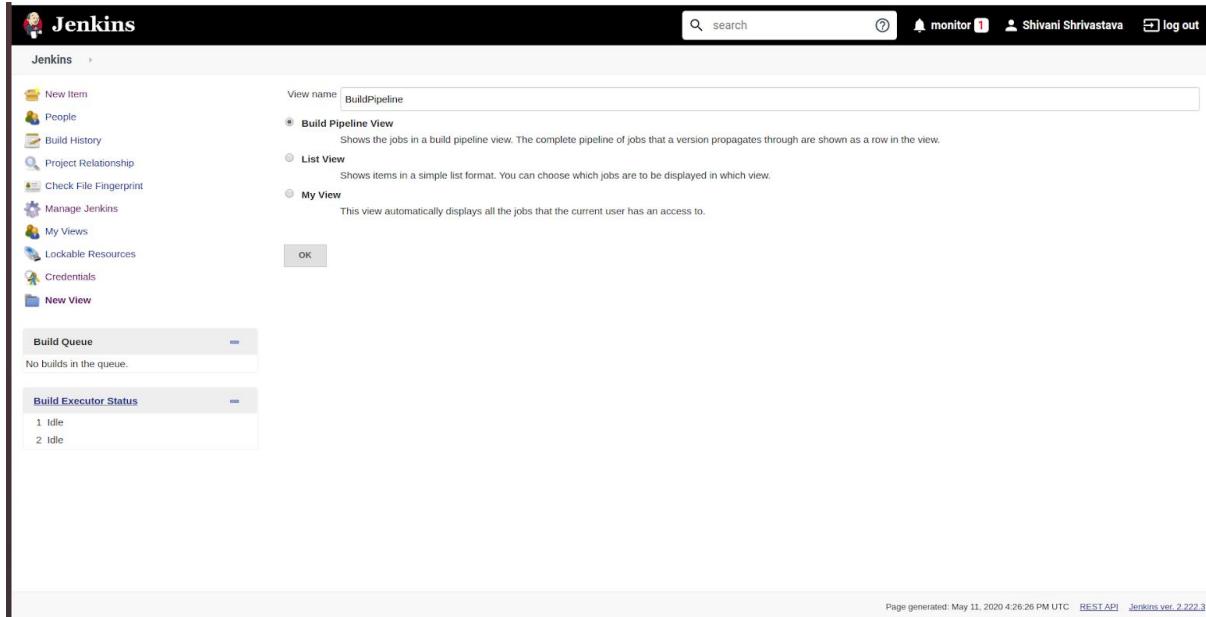


Fig 31. Build Pipeline View setup 1

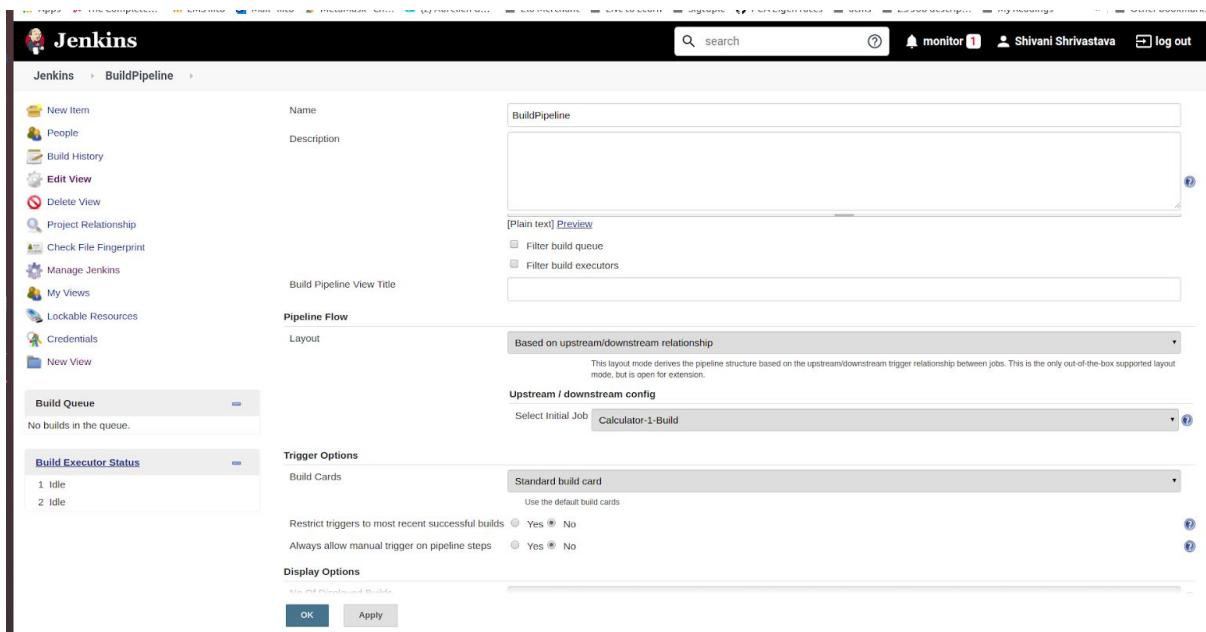


Fig 32. Build Pipeline View Setup 2

Provide description of the pipeline view and select initial job as Calculator-1-Build and keep the rest options as default. Click on Apply and then OK. Pipeline is ready.

Outputs

Whole pipeline is ready to use now. Start adding other operations subtraction, multiplication and division one after the other and build the projects in jenkins and see the results in pipeline view.

The screenshot shows the Jenkins dashboard with the 'BuildPipeline' view selected. The main area displays a table of build jobs:

S	W	Name	Last Success	Last Failure	Last Duration
		Calculator-1- Build	7 hr 30 min - #23	14 hr - #17	8 sec
		Calculator-2- Test	7 hr 30 min - #12	N/A	7.6 sec
		Calculator-3- Main	7 hr 29 min - #24 shrivastavashivani/calculatorec2:02ec2 shrivastavashivani/calculatorec2:latest	8 hr 45 min - #19 shrivastavashivani/calculatorec2:02ec2 shrivastavashivani/calculatorec2:latest	6.5 sec

Below the table, there are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 idle, 2 idle). The bottom right corner shows the page was generated on May 12, 2020 at 1:03:11 AM UTC, with Jenkins version 2.222.3.

Fig 33. Jenkins Dashboard after everything is done

The screenshot shows the 'Build Pipeline' view for the 'BuildPipeline' project. It displays three completed build steps:

- #23 Calculator-1-Build: Started on May 11, 2020 at 5:32:49 PM, took 1 sec, and is building.
- #12 Calculator-2-Test: Started on May 11, 2020 at 5:32:04 PM, took 16 sec.
- #24 shrivastavashivani/calculatorec2:02ec2 shrivastavashivani/calculatorec2:latest Calculator-3-Main: Started on May 11, 2020 at 5:32:19 PM, took 15 sec.

The top navigation bar shows the path 'Jenkins > BuildPipeline'. The top right corner includes a search bar, monitor icon, user name 'Shivani Shrivastava', and log out link.

Fig 34. Build Pipeline View

```

Started by user Shivani Shrivastava
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Calculator-1-Build
using credential 6c8ced6f-b737-48b2-a0d6-840815f0f7b3
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git # timeout=10
Fetching upstream changes from https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git
> git --version # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master{commit} # timeout=10
> git rev-parse refs/remotes/origin/master{commit} # timeout=10
Checking out Revision 0ad3bf986688f4283a9467f88110aa5ea6a3fe0 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 0ad3bf986688f4283a9467f88110aa5ea6a3fe0 # timeout=10
Commit message: "Added Division"
> git rev-list --no-walk 79e23aea6718da5cbadec09fd5694d1eece7b96 # timeout=10
[Calculator-1-Build] $ /var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/Maven_latest/bin/mvn clean package -U
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.devops:Calculator >-----
[INFO] Building Calculator 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ Calculator ---
[INFO] Deleting /var/lib/jenkins/workspace/Calculator-1-Build/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calculator ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Calculator-1-Build/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calculator ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /var/lib/jenkins/workspace/Calculator-1-Build/target/classes

```

Fig 35. Console Output for Calculator-1-Build

```

Started by upstream project "Calculator-1-Build" build number 23
originally caused by:
Started by user Shivani Shrivastava
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Calculator-2-Test
using credential 6c8ced6f-b737-48b2-a0d6-840815f0f7b3
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git # timeout=10
Fetching upstream changes from https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git
> git --version # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress -- https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master{commit} # timeout=10
> git rev-parse refs/remotes/origin/master{commit} # timeout=10
Checking out Revision 0ad3bf986688f4283a9467f88110aa5ea6a3fe0 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 0ad3bf986688f4283a9467f88110aa5ea6a3fe0 # timeout=10
Commit message: "Added Division"
> git rev-list --no-walk 79e23aea6718da5cbadec09fd5694d1eece7b96 # timeout=10
[Calculator-2-Test] $ /var/lib/jenkins/tools/hudson.tasks.Maven_MavenInstallation/Maven_latest/bin/mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.devops:Calculator >-----
[INFO] Building Calculator 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Calculator ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /var/lib/jenkins/workspace/Calculator-2-Test/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ Calculator ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /var/lib/jenkins/workspace/Calculator-2-Test/target/classes
[INFO]

```

Fig 36. Console Output for Calculator-2-Test

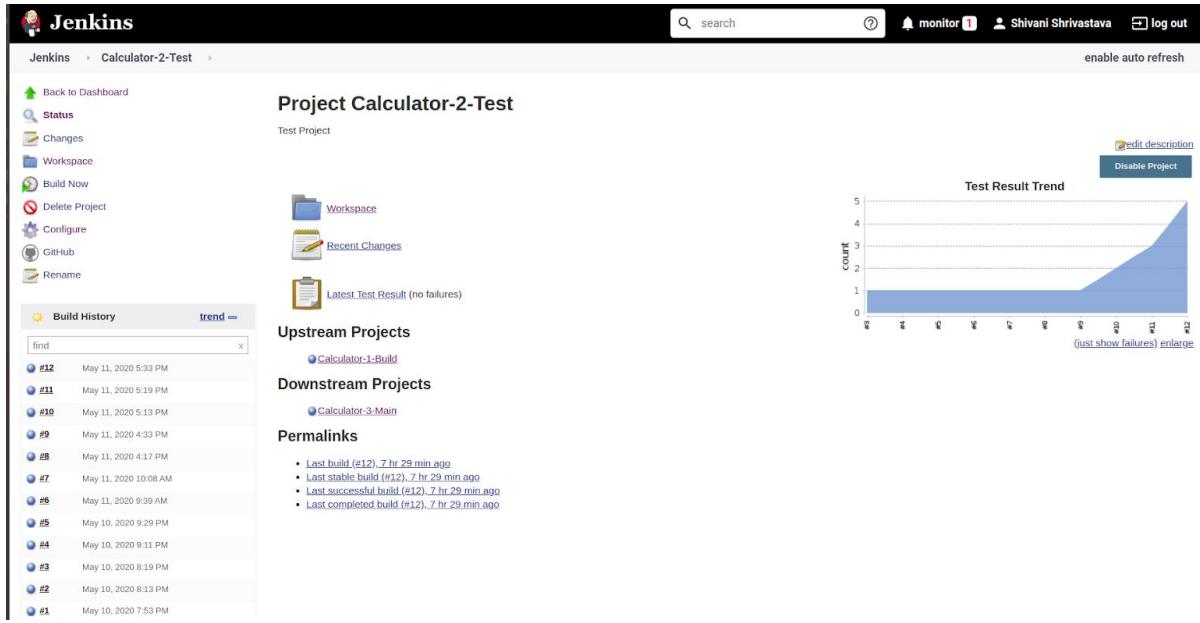


Fig 37. Test Result for Calculator-2-Test

```

Started by upstream project "Calculator-2-Test" build number 12
originally caused by:
Started by upstream project "Calculator-1-Build" build number 23
originally caused by:
Started by user Shivani Shrivastava
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Calculator-3-Main
using credential 6c0cd6f-b737-48b2-a0d5-840815f0f7b3
> git rev-parse --is-inside-work-tree & timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git # timeout=10
Fetching upstream changes from https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git
> git -version # timeout=10
using GIT_ASKPASS to set credentials
> git fetch --tags --force --progress .. https://github.com/ShrivastavaShivani/NewDevOpsCalculator.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 0ed3bf986688f4428a9467f8810aa5ea6a3fe8 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 0ed3bf986688f4428a9467f8810aa5ea6a3fe8 # timeout=10
Commit message: "Added Division"
> git rev-list --no-walk 79e23aea6718da5cbadac89fd594d1eece7b99 # timeout=10
[Calculator-3-Main] $ docker build -t shrivastavashivani/calculatorec2:02ec2 --pull=true /var/lib/jenkins/workspace/Calculator-3-Main
Sending build context to Docker daemon 205.3kB
Step 1/7 : FROM openjdk:8-jre-alpine as target
8-jre-alpine: Pulling from library/openjdk
Digest: sha256:f1362b165b879ef129cbef730f29065ff37399c0aa8bcab3e44b51c302938c9193
Status: Image is up to date for openjdk:8-jre-alpine
--> 47a2029ab70c
Step 2/7 : RUN mkdir -p /opt/app
--> Using cache
--> ad41b6819c61
Step 3/7 : ENV Project_reside /opt/app
--> Using cache

```

Fig 38. Console Output for Calculator-3-Main

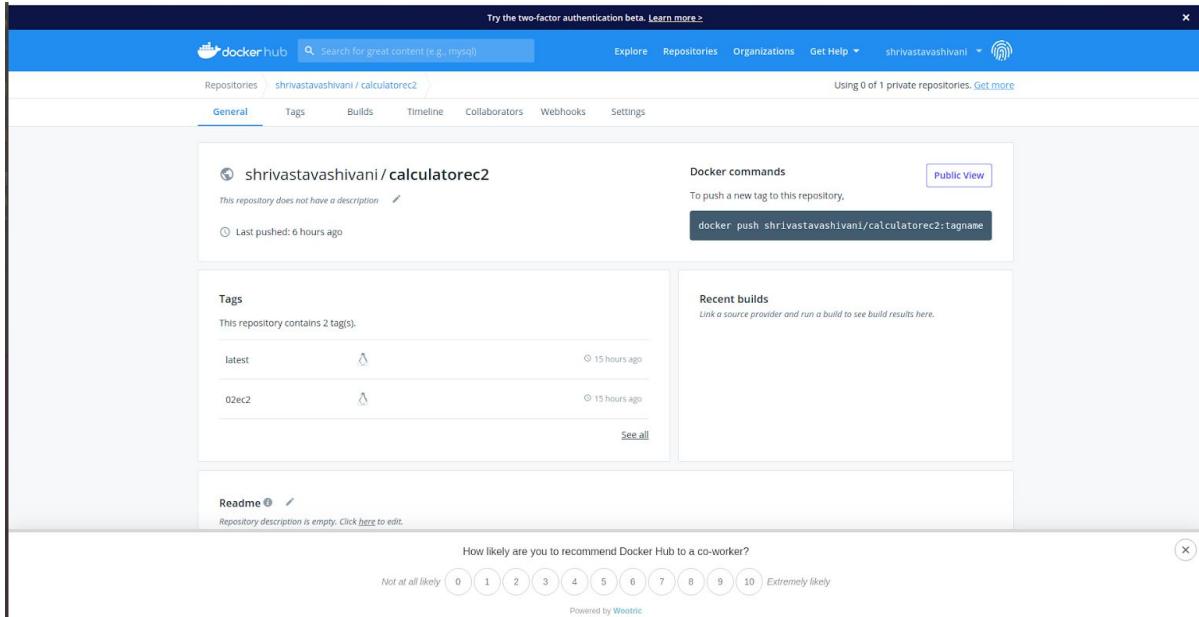


Fig 39. My Docker Hub

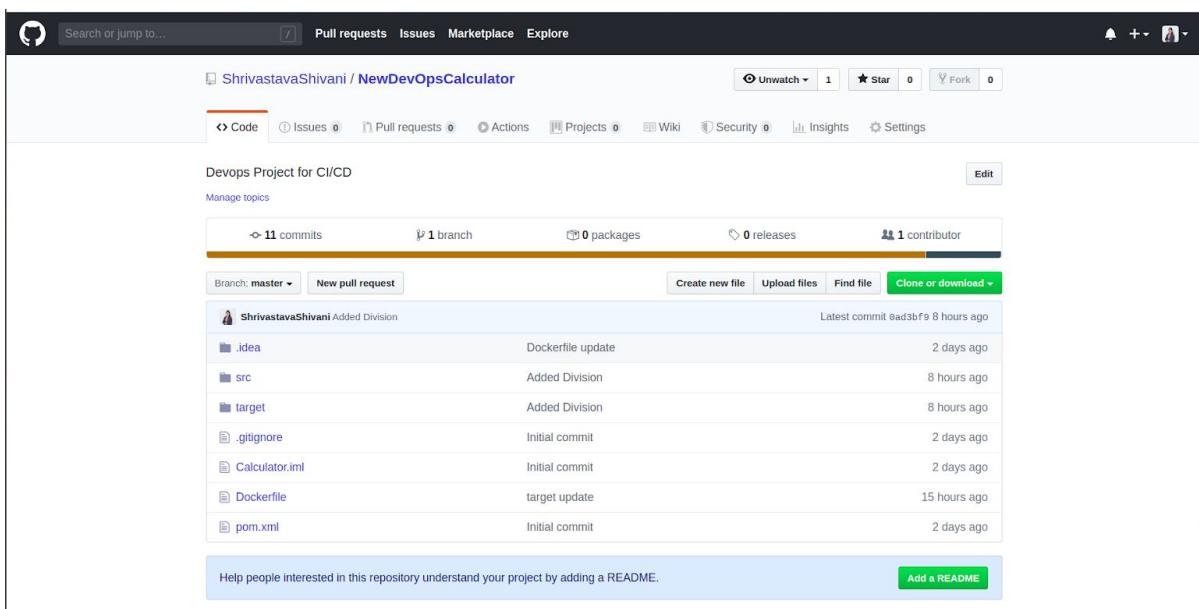


Fig 40. My GitHub