SPE

Mini Project Report

Scientific Calculator Application

P.S.V. Sai Madhav (IMT2018524)

Introduction

This is a Scientific calculator application written in java. The main theme of the project is to use the devops tools learnt in the class. We will be understanding the theory,installation and deployment of the tools.

The Calculator app has four operations:

- Square root function - √x
- Factorial function - x!
- Natural logarithm (base e) - ln(x)
- Power function - x^b

Devops

What is Devops?

Devops is a software engineering methodology which aims to automate and integrate the work of software development and software operations teams by facilitating a culture of collaboration and shared responsibility. Devops is a culture of shared accountability between the Dev and Ops teams.

The common goal of devops is - continually delivering value.One way agile techniques achieve this is by bringing people from all groups together in a team so that they begin to care about the same things. Devops is largely about making it easier for people from these backgrounds to care about the same things by removing roadblocks and streamlining delivery. This includes things like automated build, test, deployment pipelines (continuous integration and continuous delivery), writing code in a way that will handle production level load and can be monitored, and having for example devs and testers on the pager so that everyone in the team has responsibility for how well the system runs in production.

Why Devops?

Main reason for companies to use devops are:

- Faster time to market for software
- Lower DownTime
- Rapid improvement based on feedback
- Less Menial work Thanks to Automation
- less complexity to manage
- Quick defect fixing

Tools used

- Source control management: Git, GitHub
- Testing: J Unit
- IDE: Intellij IDEA CE
- Build: Maven
- Continuous Integration: Jenkins
- Containerize: Docker
- DockerHub
- Configuration management and deployment: Ansible
- VM for m1 mac : UTM
- Monitoring: ELK Stack

OpenJDK (Java)

Introduction

Java is a widely used programming language in application development. Java is an object oriented programming language. Java allows us to build an application which can also run on distributed server architecture.

Configuration

Installation of java:

```
$ brew install openjdk@11
```

$ export PATH="/usr/local/opt/openjdk@11/bin:$PATH"

Verification :

$ java –version

Git , GitHub

Introduction

Git is a Distributed version control system. This approach follows the peer to peer approach. This is used to maintain and manage the source code. GitHub is an online web hosting service that allows the developers who use git to share/showcase their code to their peers.

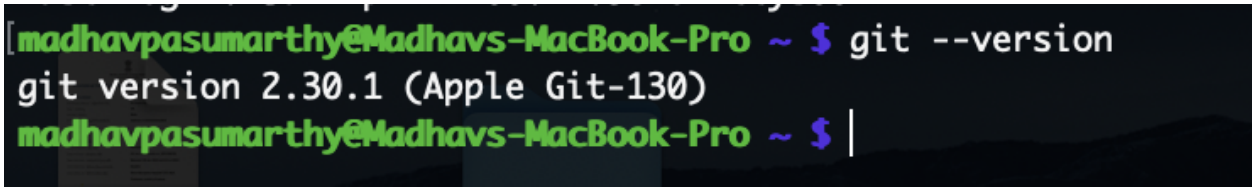
Configuration

Installation of git in My MacBook:

$brew install git

Verification:

$ git —version

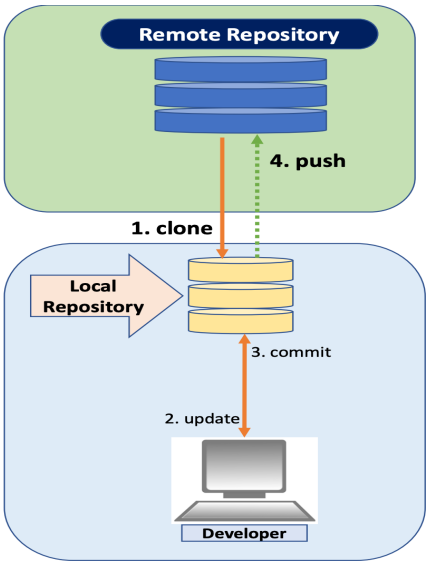


Configuration of Github:

$ git config -- global user.name “myusername”

$ git config -- global user.email “myemail@gmail.com”

Now we have to create a Repository in GitHub. Next step is to clone the repo into our system. Checkout to a branch, Then we use the git add command to add the files to have the changes in the development history. Next step is to commit the changes to the local repo with a proper commit message. Finally we push the code to our gitHub so our peers can access our code. The image displays the steps explained above. Lookout for Merge conflicts.



Steps to update the remote repo:

$ git pull origin main # do this before adding any feature so that repo becomes uptodate

$ git checkout -b <branch_name>

$ git add .

$ git commit -m “commit message”

$ git push origin main

Note: In git configuration password of github profile might not work so you need to generate a personal access token from [Personal access tokens (github.com)]

Maven

Introduction

Maven is a Dependency management and Build tool. A project will use some library defined code which can be using some other library code. It would be hard for us to install all the dependencies manually for a project and the same thing for every project. This is where the Dependency management part of maven helps us, we need to specify the libraries and their versions and maven does the work for us. Now that we have libraries we can write our application code. Maven helps us to bundle these dependencies for a build. Maven is going

to create a project structure, pom file. The pom file is going to have all the configurations required for the project. Maven supports various plugins for various use cases, we can just specify these in the pom file maven makes sure they are included in the project.

Maven builds a single jar file from our java files, test files, dependencies.

Configuration

Installation of maven:

```
$ brew install maven
```

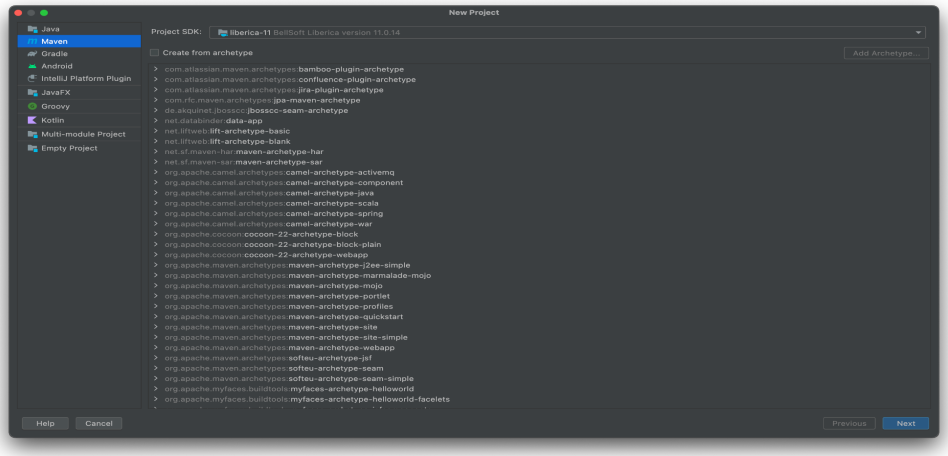
Verification :

```
$ mvn –version
```

```
madhavpasumarthy@Madhavs-MacBook-Pro ~ $ mvn --version
Apache Maven 3.8.5 (3599d3414f046de2324203b78ddcf9b5e4388aa0)
Maven home: /opt/homebrew/Cellar/maven/3.8.5/libexec
Java version: 18, vendor: Homebrew, runtime: /opt/homebrew/Cellar/openjdk/18/lib
exec/openjdk.jdk/Contents/Home
Default locale: en_IN, platform encoding: UTF-8
OS name: "mac os x", version: "11.6.1", arch: "aarch64", family: "mac"
```

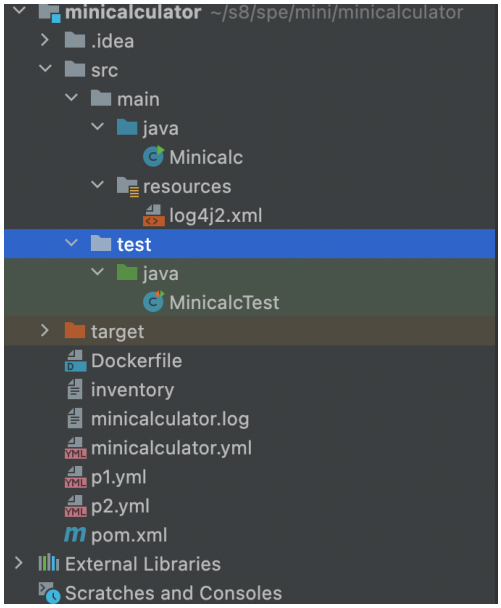
IntelliJ IDEA CE:

Open the IDE. Create a new project by Choosing from File -> New -> Project. Choose Maven and the java version (Eg 8,11) in the top dropdown. In the next step choose name, location



for the project. Add the dependencies in the pom file. This repo needs to be pushed to github. Do git init, git add, git remote add , git push in this order for the first time. For next time the process is explained in the Git, Github section.

Image depicts my folder structure which was created by Maven.



src folder contains the actual code of our application. Main folder contains the java classes, and the resources folder contains the config files for the application.

test folder contains the test files we run in JUnit

Note that Test class name should be the main class name followed by test.

Commands I majorly used :

1. $ mvn clean
2. $ mvn clean install

First command is to clear the previous target folder. Second command is to clear the previous target folder, build the application, run the test cases written for that application combined.

Make sure you have added all the dependencies in the pom file. All dependencies have to be inside <dependencies> </dependencies>. Each dependency has to be written inside<dependency> </dependency>.

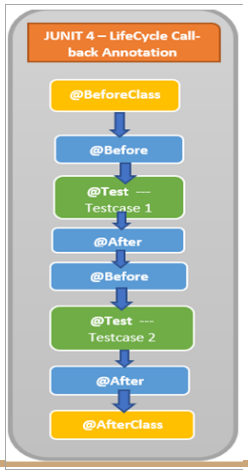
```xml
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-api</artifactId>
        <version>2.14.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.14.0</version>
    </dependency>
</dependencies>
```

J Unit

Introduction

J Unit is a unit testing library used by the developers for their java applications. Unit testing usually helps in the long run because manually testing for every method at every build is hard(code can be changed by anyone with github access). Here we will be testing for all the major class methods in our program. A crude example would be if I pass a parameter X to my function fun1 then the result should be Y or the result should not be Z.

Test cases must be preceded by the @Test annotation. Junit also supports defining a method to execute before (or after) each (or all) of the test methods with the @BeforeEach (or @AfterEach) and @BeforeAll (or @AfterAll) annotations.



There are 4 cases we can test a function for:

1. True Positive: This case is when the code is perfect and the test cases pass.

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

As this is a simple project we mainly focused on one true positive and one false positive for each operation. The file is written in the test folder of our project with an appropriate class name.

There are two functions i used for testing:

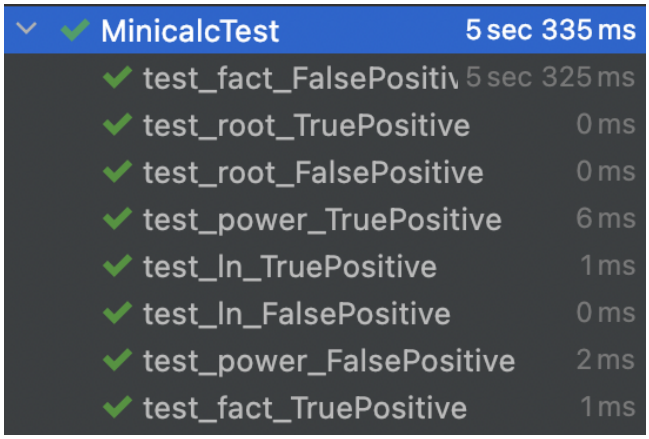
1. assertEquals(expectedValue,returnedValue,delta) // for true positive

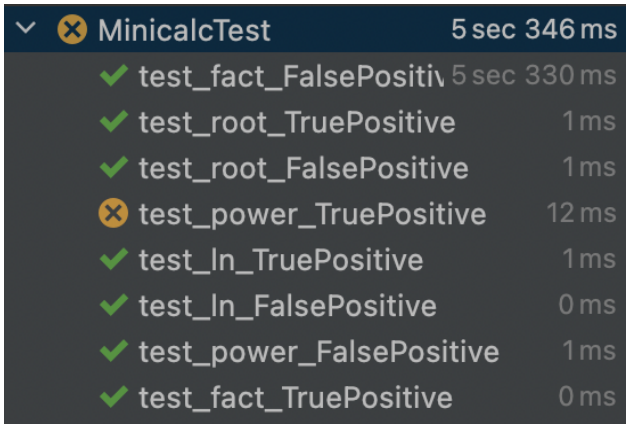
```java
import static org.junit.Assert.*;
import org.junit.Test;

public class MinicalcTest {
    private static final double DELTA = 1e-11;
    Minicalc calculator = new Minicalc();
    @Test
    public void test_root_TruePositive() {
        double actual=calculator.root( number1: 121.0);
        double exp=11.0;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_fact_TruePositive() {
        double actual=calculator.factorial( number1: 7);
        double exp=5040;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_ln_TruePositive() {
        double actual=calculator.log( number1: 1096.63315843);
        double exp=7;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_power_TruePositive() {
        double actual=calculator.power(3.0,5.0);
        double exp=243.0;
        assertEquals(actual,exp, DELTA);
    }
}
```

2. assertNotEquals(expectedValue,returnedValue,delta) // for false positive

```java
    @Test
    public void test_root_FalsePositive() {
        double actual=calculator.root( number1: 645.0);
        double exp=6.0;
        assertNotEquals(actual,exp, DELTA);
    }
    @Test
    public void test_fact_FalsePositive() {
        double actual=calculator.factorial( number1: 3);
        double exp=3241;
        assertNotEquals(actual,exp, DELTA);
    }
    @Test
    public void test_ln_FalsePositive() {
        double actual=calculator.log( number1: 52342);
        double exp=6.0;
        assertNotEquals(actual,exp, DELTA);
    }
    @Test
    public void test_power_FalsePositive() {
        double actual=calculator.power(5.0,3.0);
        double exp=2.0;
        assertNotEquals(actual,exp, DELTA);
    }
}
```

delta of the function parameter is the maximum allowed difference in the expected and returned values.

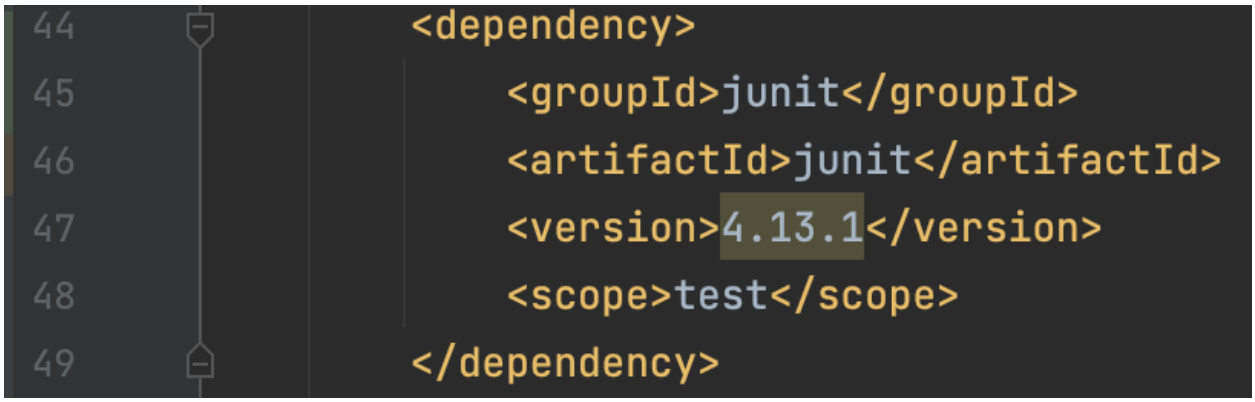




On successful Run of the test cases we can find the which test cases have passed and which aren't in the bottom right corner in intellij in the above format

Configuration

In maven we can simply add the JUnit to our project by adding the dependency in the pom.xml file. Maven commands can be used to test the build.



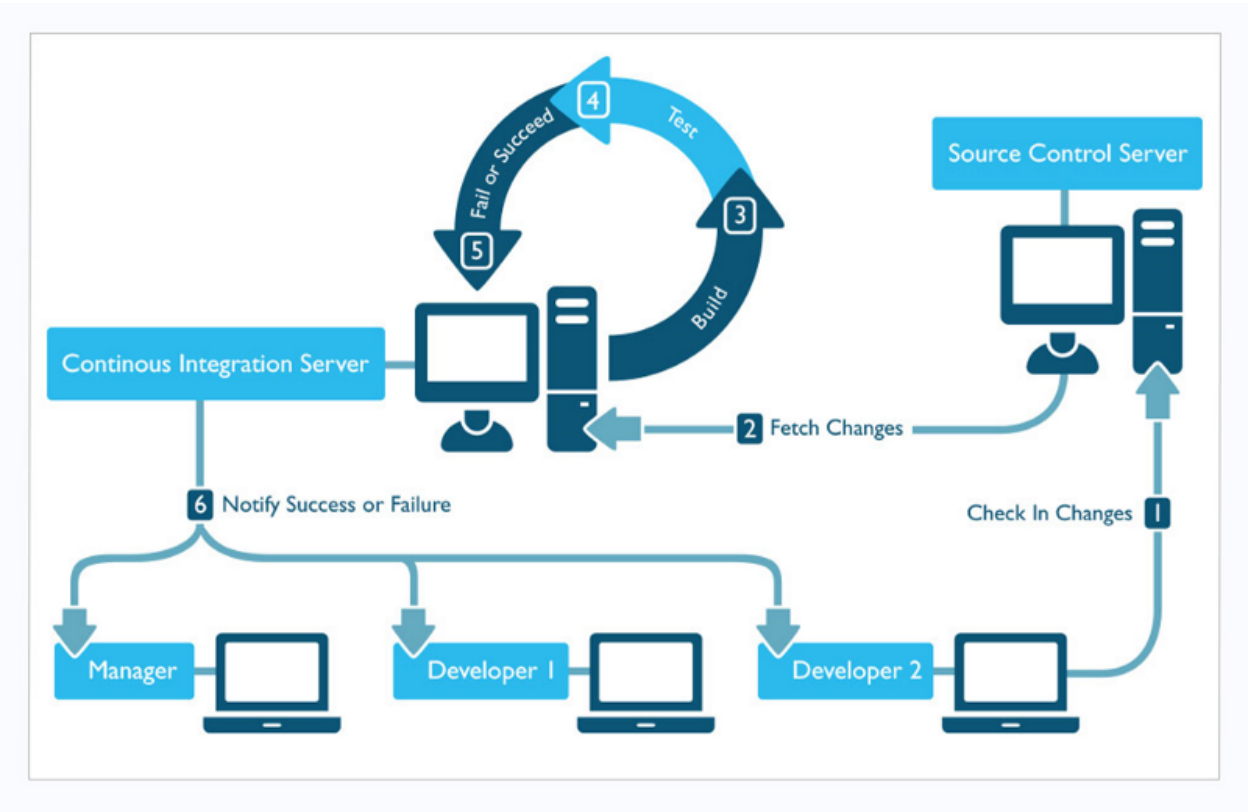
Continuous Integration

Introduction

It is a process of automating the build and testing of code every time when a change is committed to the version control system. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Without Continuous Integration we cannot automate from one stage to another stage. In each stage we can automate with the help of an automation tool but if we want to trigger from one stage from another stage then we definitely need continuous integration.

Along with running the unit and integration checks, you can check the quality of your code and profile performance and format the documentation from the source code, thereby facilitating QA processes.



Jenkins

Jenkins is an open source tool for CI integration. In Jenkins we will be automating the SDLC components like build,test. Jenkins clearly specifies the status of each stage. Also it can notify the Developer in case of build failure/success.

Configuration

Installation of jenkins in macOS:

```
$ brew install jenkins-lts
```

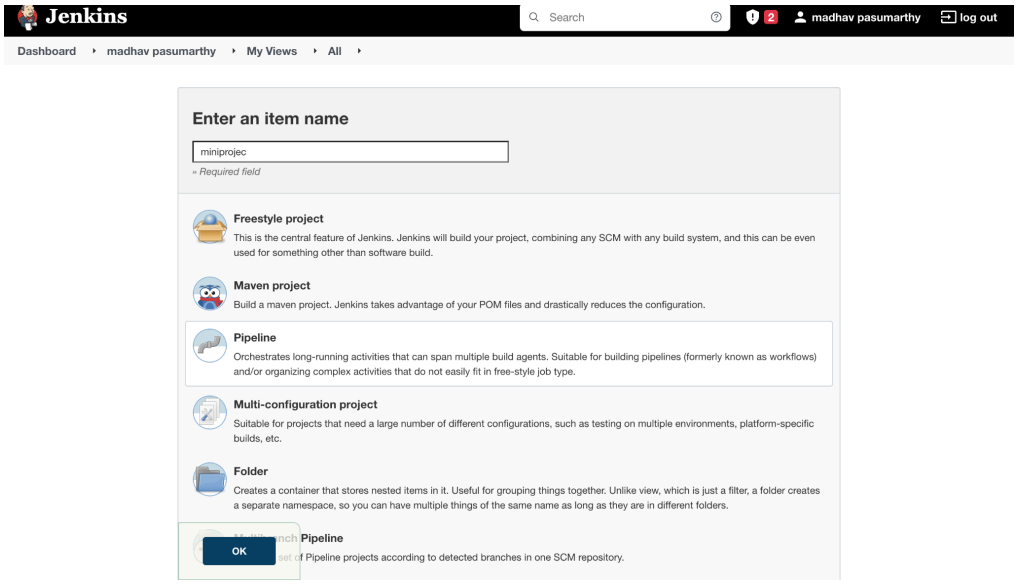
To start and stop Jenkins:

```
$ brew services start jenkins-lts
```

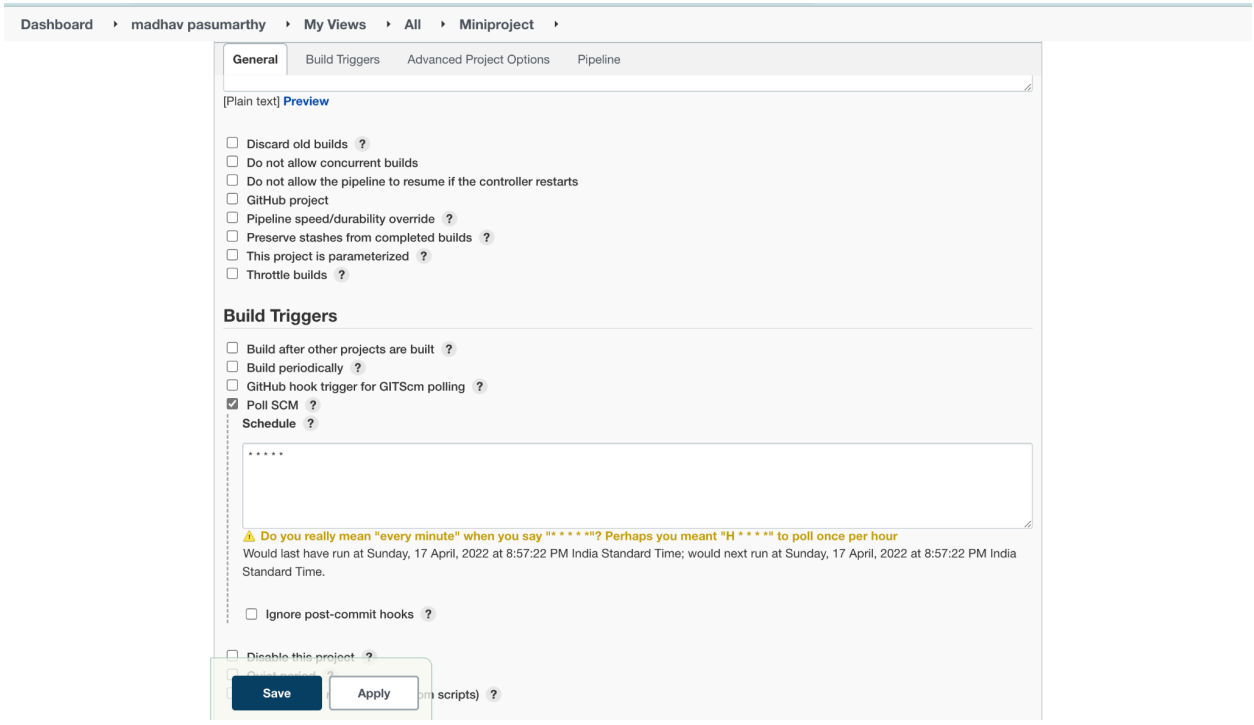
```
$ brew services stop jenkins-lts
```

Open http://localhost:8080 to use jenkins. Complete the setup and create an account. For the admin password run “sudo cat /var/lib/Jenkins/secrets/initialAdminPassword” in the terminal.

Install the suggested plugins. After this we can create a project.



Enter the project name and choose pipeline as we have separate stages for this project.



Here I have chosen the PollSCM build trigger. This trigger checks periodically if there are any new commits in the version control system, If so a new jenkins build will be triggered else no job will be triggered. As seen in the image, the time period for this trigger is to be given in the crontab format and I have given 1 min in the above format.

By now we have created Github and Maven. We create a pipeline for these two.

Make sure that Github and maven plugins are installed before running the script

Script ?

```
pipeline {
    environment {
        PATH="/opt/homebrew/bin/:/usr/local/bin/:$PATH"
    }
    agent any

    stages {
        stage('Step 1: Git clone') {
            steps {
                git branch: 'main', url: 'https://github.com/madhav2391/minicalculator.git'}
        }

        stage('Step2 : Maven Build') {
            steps {
                sh 'mvn clean install'
            }
        }
    }
```

This script is to be used for the first two stages. Not that path variable is necessary as m1 mac has a different path. If not used we will get an error. In the Git clone branch I have chosen main as my default branch name is main not master. Output of the pipeline will be as below.

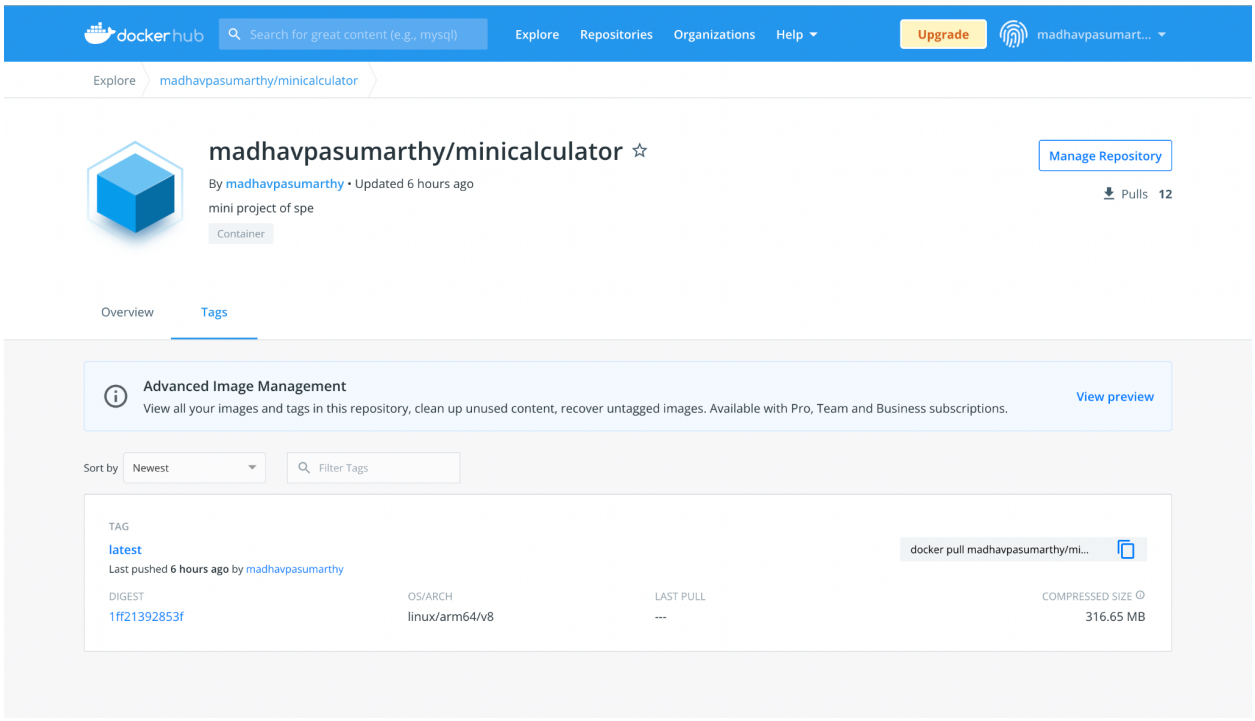
Docker

Introduction

Docker is based on the concept of "containers," which are dedicated environments in which a Docker image's services can execute. VM requires a hypervisor, custom OS extensions and sophisticated management of resources which are unnecessary if you want to deploy small applications like a local web server. One of the main uses of docker is that your code becomes mostly platform independent which helps in your deployment.

VM's use Virtual Domains (space VM uses) while docker creates "Containers," which are essentially sandboxes for whatever image is running inside of them. As a result Containers are very light weight. These containers are going to be loosely isolated, So Rapid deployment and horizontal scaling of the product is going to come in handy.

DockerHub is a way to manage your team's images. This is similar to the Github here we can upload to share our image to our peers. We can also find many official images released by the companies or custom modified of those images of individuals.



Configuration

Installation in macOS:

1. Go to the official docker site (Docker Desktop for Apple silicon | Docker Documentation)
2. Download and open the image
3. After successful installation click on the signin on the top right corner which redirects you to Docker hub. Create an account in the Docker Hub.

Installation in VM Ubuntu server:

1. $ sudo apt update
2. $ sudo apt install apt-transport-https ca-certificates curl software-properties-common
3. $ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
4. $ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"
5. $ sudo apt install docker-ce
6. $ sudo service docker status
7. Execute docker without sudo (mandatory)
    a. $ sudo groupadd docker
    b. $ sudo usermod -aG docker ${USER}
    c. $ newgrp docker
    d. Restart the OS
    e. Check if docker runs without sudo
        i. $ docker run hello-world

Docker Commands:

- docker pull: docker pull <image name>
- show all the running containers: docker ps -a
- Docker run: docker run -i <imageID>
- Remove a image: docker rmi <imageID>
- Remove a container: docker rm <imageID>

Project Containerisation

Two ways to create a docker image:

- Running the jar file in the container
- Compile the application inside the container to generate a jar file and then run it

First process is very simple so we will be using it. Docker file has to be written in the project file which helps in copying the jar file.

```
FROM --platform=linux/arm64/v8 openjdk:11
COPY ./target/minicalculator-1.0-SNAPSHOT-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java", "-jar", "minicalculator-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

First line we are specifying which java version the base image has to be compiled. Extra specification of the platform has to be mentioned as It is built on an M1 machine and deployment is on other platforms.

Next we are copying the jar file, changing the working directory. Last command is to run our jar file.

Adding Docker to jenkins

In Jenkins we need to add the Docker hub credentials so that the Jenkins pipeline can directly push the image to DockerHub.

Go to Dashboard-> manage jenkins -> Manage Credentials -> Jenkins(store) -> Global Credentials -> Add credential. add DockerHub credentials here

Display gets updated in manage credentials as above.

Last step before editing pipeline script Install all docker related essential plugins so that docker can successfully run in the pipeline.

We have to update our pipeline script to incorporate the two docker stages i.e, Docker image build and Docker image push.

Script ?

```
    }
    stage('Step3 : Building Docker Image'){
    steps{

        script{
            dockerImage = docker.build registry
        }
    }
    }
    stage('Step4 :Push Docker Image'){
        steps{
            script{
                dockerImage.push()
            }
        }
    }
```

Note: Don't forget to add the line in the environment in the pipeline

registry = "madhavpasumarthy/minicalculator"

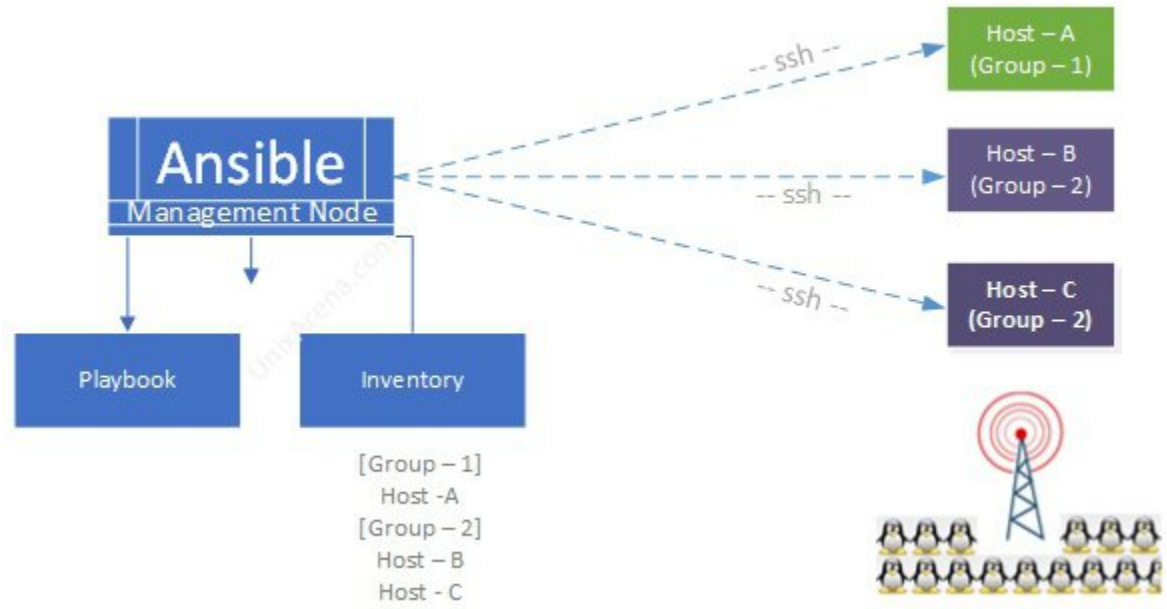
This line tells Jenkins which image to be pushed and build. All users have to add this line.

Updated Pipeline is going to be as below.

Ansible

Introduction

Ansible is a configuration tool and Automation tool. This uses infrastructure as a service approach to manage various hosts at a single time. Ansible is agentless while other configuration management tools such as Puppet are not. As a result you can have more space resources for the application. Ansible will be talking to the clients using ssh. Ansible requires python to run on the client side. Ansible can manage both local and remote hosts.



Ansible connects to the server through SSH and pushes out little programmes i.e, Ansible Playbooks(Yaml file). These play books follow human readable syntax so the developer need not learn a new advanced language instead directly work on the program. This playbook contains the instructions for the client to execute. This playbook can contain different instructions for different instructions for different hosts/Groups.

Configuration

Installation in macOS:

```
$ brew install ansible
```

Verification :

```
$ ansible –version
```

Ubuntu server in VM

To complete implementation of Ansible we need to create a remote Target machine. This can be done by installing a VM ubuntu server. I have used UTM to create a VM.

Openssh has to be installed in both local and target machines. Make sure that the target machine has python, docker installed. Note the ip address of the target.

Make sure you can run docker in normal user mode. Steps are specified in the docker section.

For ssh we need to create a public and private key combination in the local machine. This can be done by the command

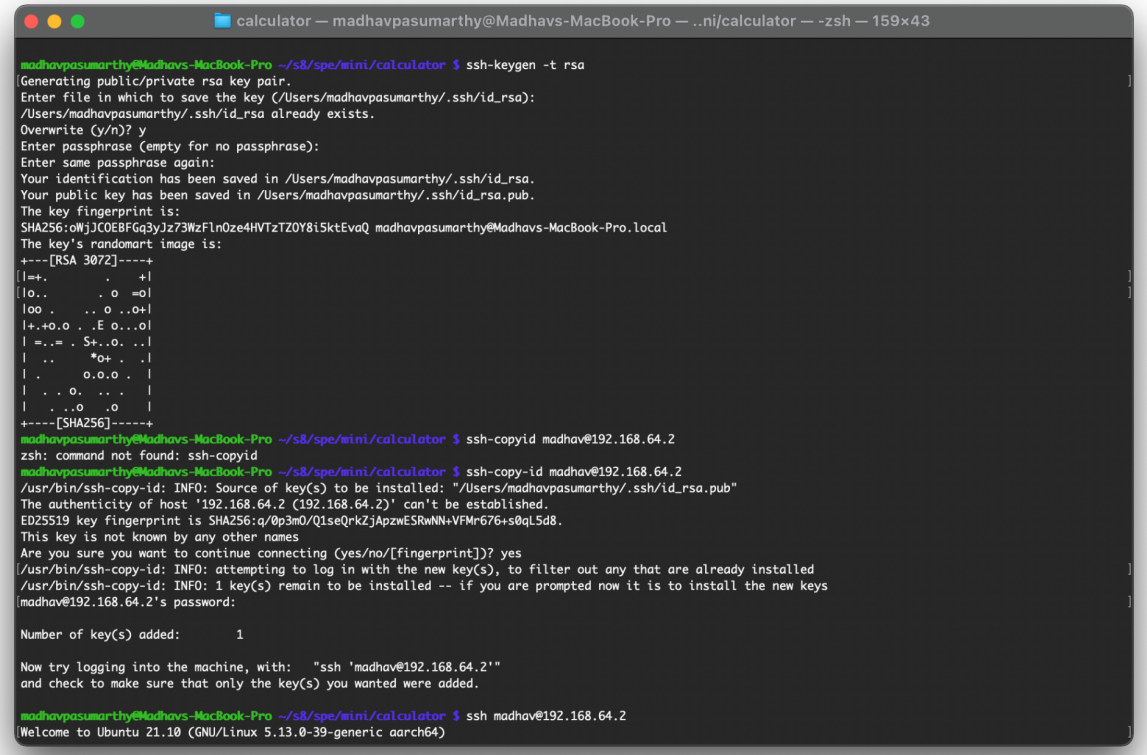
```
$ ssh-keygen -t rsa
```

And the next step would be to copy this generated key to the remote machine. It can be done by

```
$ ssh-copy-id <remote username>@<remote ip address>
```

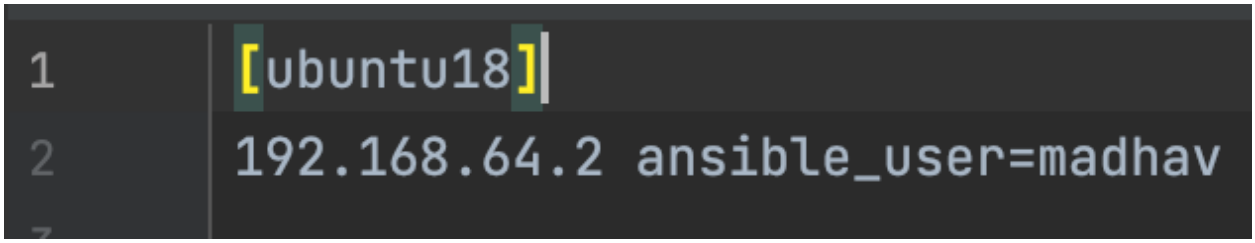
After success we can ssh into the remote with

```
$ ssh <remote username>@<remote ip address>
```



Inventory

Inventory file contains all the target machine details. Only targets added here can be used in the Playbook.



The first line here is the name of the target to be used in the playbook. Next line contains the remote ip address of the target followed its username

Ansible Playbook

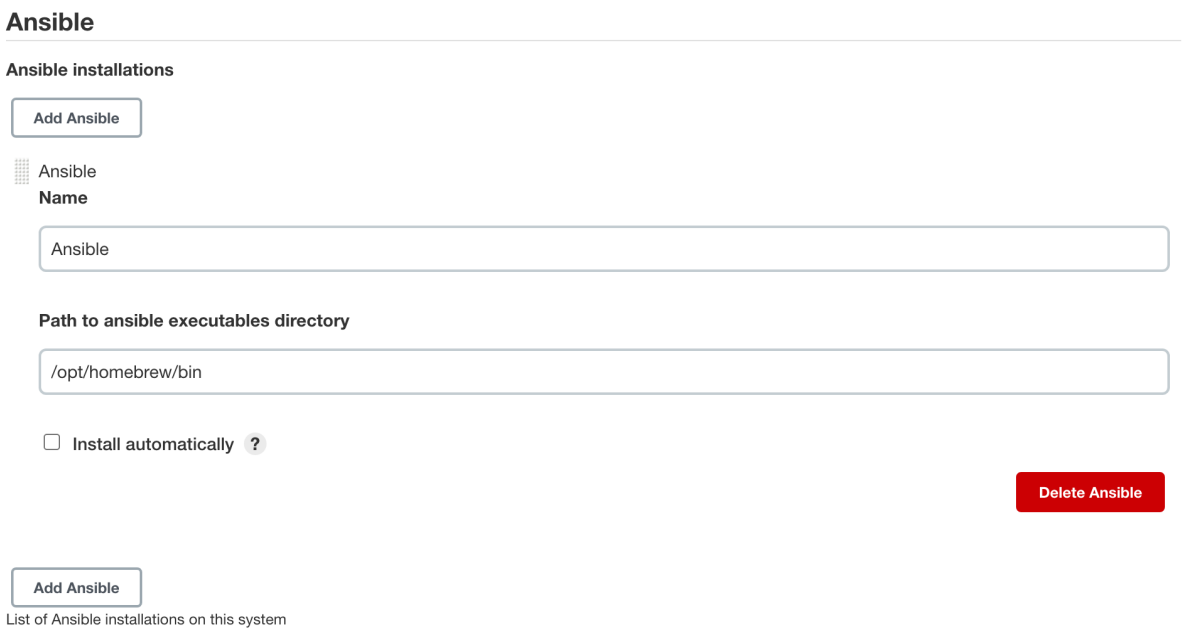
As mentioned Playbooks contain the instructions to be executed in the target machines

```yaml
---
- name: Deploy docker img
  hosts: all
  tasks:
    - name: Pull calculator image
      docker_image:
        name: madhavpasumarthy/minicalculator:latest
        source: pull

```

Above playbook is used in the pipeline. Here I am instructing the target machine to the docker image from my docker hub. In the third line i have used hosts: all as i have only one target machine i have used all. If I have multiple target machines and different instructions have to be sent we need to write different sections with different host names.

Finally before adding the pipeline we need to add Ansible in the global tool configuration.

Go to Dashboard-> manage jenkins-> global tool configuration



Add these exactly the same way.

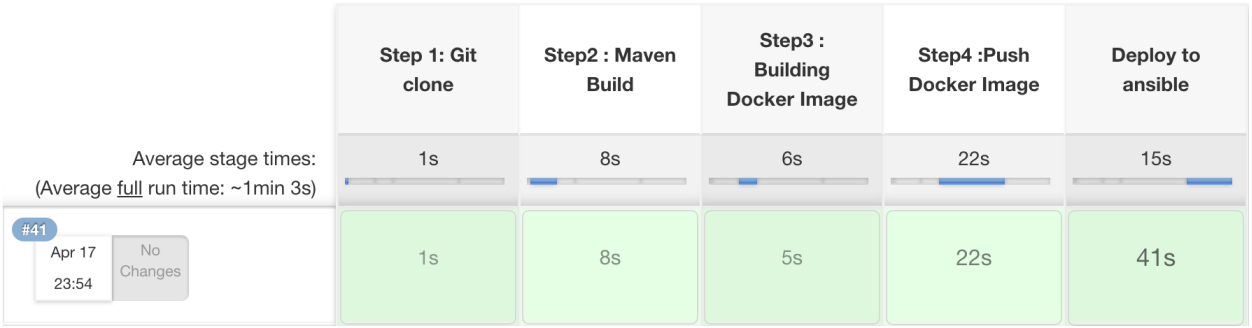
Pipeline Script

Before starting the pipeline make sure that all the relevant ansible plugins are installed.

Append this to the pipeline script:

```
stage('Step5 :Deploy to ansible'){
        steps{
            ansiblePlaybook becomeUser: null, colorized: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inventory', playbook: 'minicalculator.yml', sudoUser: null
        }
    }
```

Final Pipeline of the minicalculator



As we can see every stage is a success. Let us now ssh into my VM and check if the application is working.

```
madhavpasumarthy@Madhavs-MacBook-Pro ~/s8/spe/mini/minicalculator (main?) $ ssh madhav@192.168.64.2
Welcome to Ubuntu 21.10 (GNU/Linux 5.13.0-39-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun Apr 17 06:28:12 PM UTC 2022

  System load:  0.33               Processes:               165
  Usage of /:   52.7% of 13.18GB   Users logged in:         1
  Memory usage: 9%                 IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for enp0s10: 192.168.64.2

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

45 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


Last login: Sun Apr 17 18:27:07 2022 from 192.168.64.1
madhav@madhav:~$ docker images
REPOSITORY                       TAG       IMAGE ID       CREATED         SIZE
madhavpasumarthy/minicalculator   latest    3b37914e3eff   3 minutes ago   651MB
madhav@madhav:~$
```

We can see that docker images are copied to the VM. Now let us run the application.

```
REPOSITORY                       TAG       IMAGE ID       CREATED         SIZE
madhavpasumarthy/minicalculator   latest    3b37914e3eff   3 minutes ago   651MB
madhav@madhav:~$ docker run -i 3b
Welcome !! This is my mini project
Select an option
1: square root
2: factorial
3: natural logarithm
4: power function
Any other number to exit
2
Enter a number
6
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
18:29:40.473 [Minicalc.java] INFO  Minicalc - [FACTORIAL] - 6
18:29:40.484 [Minicalc.java] INFO  Minicalc - [RESULT - FACTORIAL] - 720.0
Factorial of the given number is  : 720.0
Select an option
1: square root
2: factorial
3: natural logarithm
4: power function
Any other number to exit
1
Enter a number
525
18:29:49.224 [Minicalc.java] INFO  Minicalc - [ROOT] - 525.0
18:29:49.227 [Minicalc.java] INFO  Minicalc - [RESULT - ROOT] - 22.9128784747792
Square root of the given number is : 22.9128784747792
```

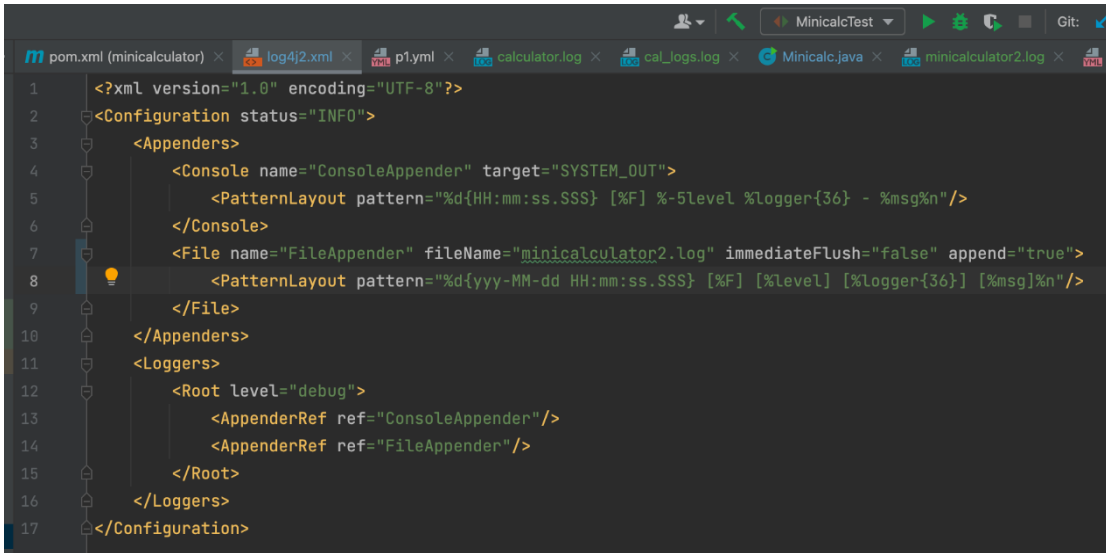
Log4j

Introduction

Log4J is a java library that enables you to put the equivalent of a “println” statement all over your code. Additionally in Log4j different levels of logging can be set such as TRACE,DEBUG,INFO,WARN,ERROR and FATAL.If a log level is set messages will be logged for that level and level below it. This is extremely useful because the larger your program gets, log files get unnecessarily cluttered and debugging becomes a pain if we use system.println().

Configuration

Make sure you add the dependencies in the pom file. Dependencies are shown in the maven section. Next we have created a new log4j2.xml file inside the src->main->resources folder.



Above is my log4j2.xml file. In the 8th line I have specified the format to be appended in the output log file.

ELK Stack

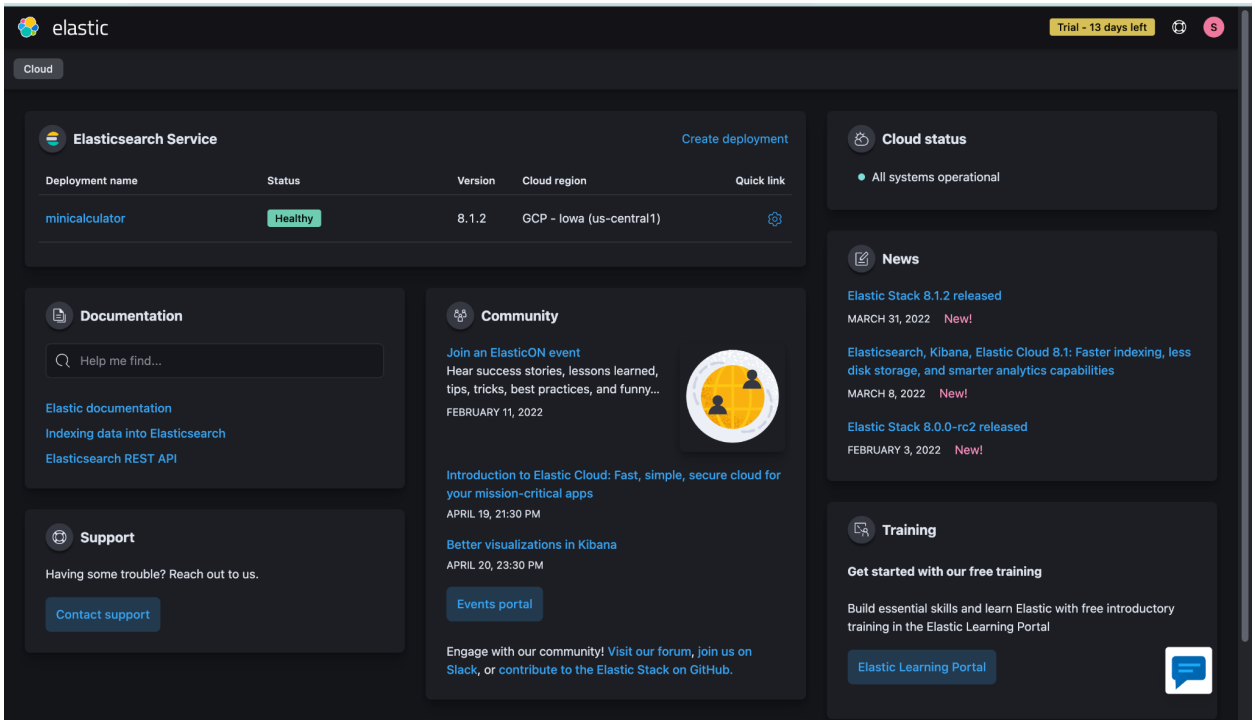
Introduction

"ELK" stands for Elasticsearch, Logstash, and Kibana which are three open source projects. Elasticsearch is a search and analytics engine. Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch. Kibana visualizes data with charts and graphs in Elasticsearch.

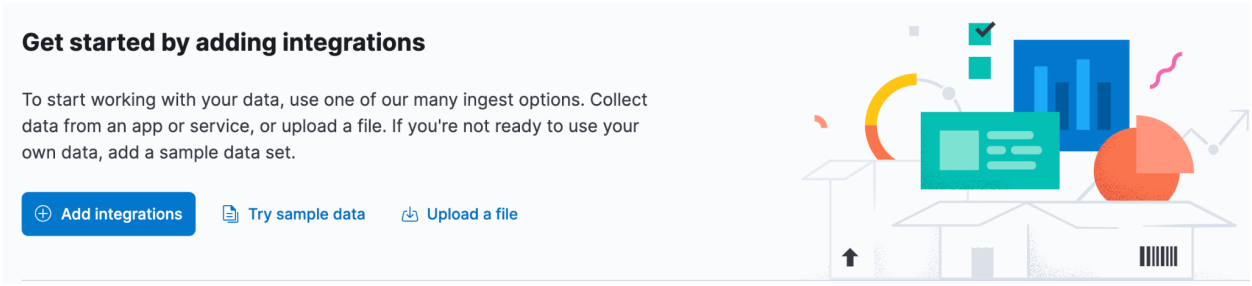
Configuration

Create an account in[The ELK Stack: From the Creators of Elasticsearch | Elastic](#).

Select Create a deployment. Fill in the details and click on the deployment. Display will be similar to the below image



Next step is to upload a file. Click on the upload file option.



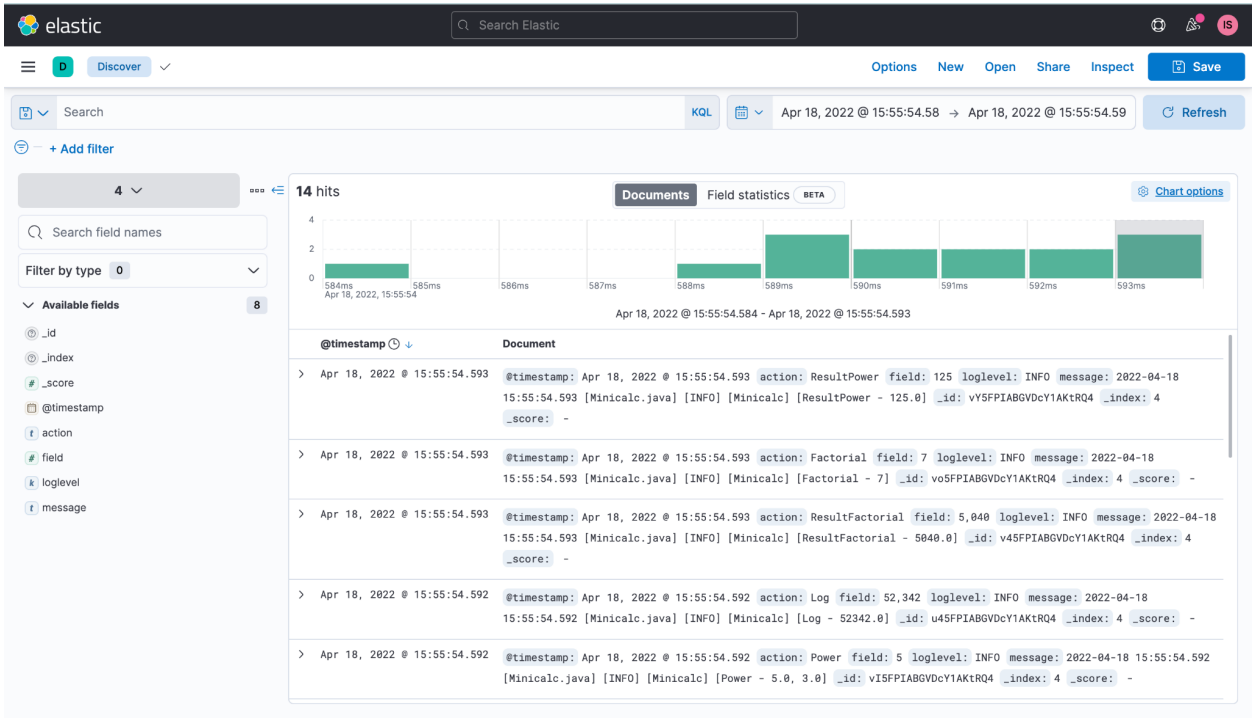
Once we have selected that option we will be asked to drag a file. Drop the log file created by the project. If the parsing was successful you get an option to import. Click on import.

In the next step we get to choose simple or advanced. Click on Advanced.

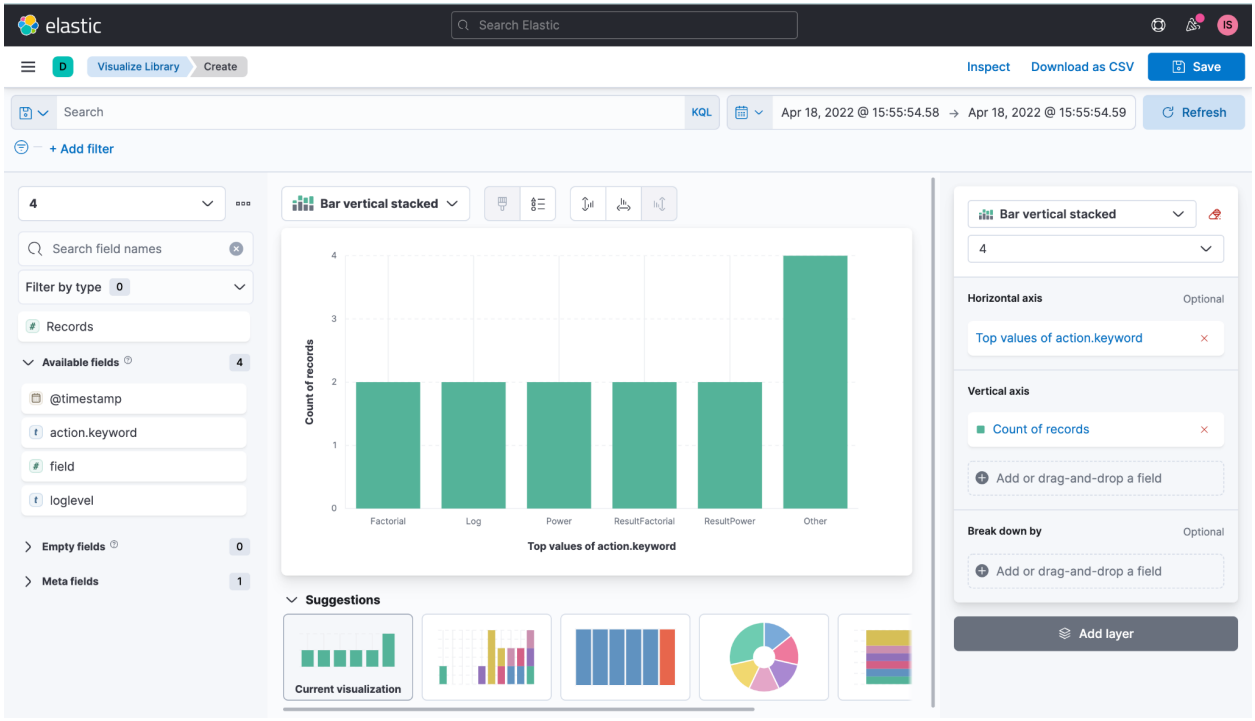
```
[
  {
    "grok": {
      "field": "message",
      "patterns": [
        "%{TIMESTAMP_ISO8601:timestamp} \\[.*?\\] \\[%{LOGLEVEL:loglevel}\\] \\[.*?\\] \\[%{WORD:action} .*?%{NUMBER:field}.*"
      ]
    }
  },
  {
```

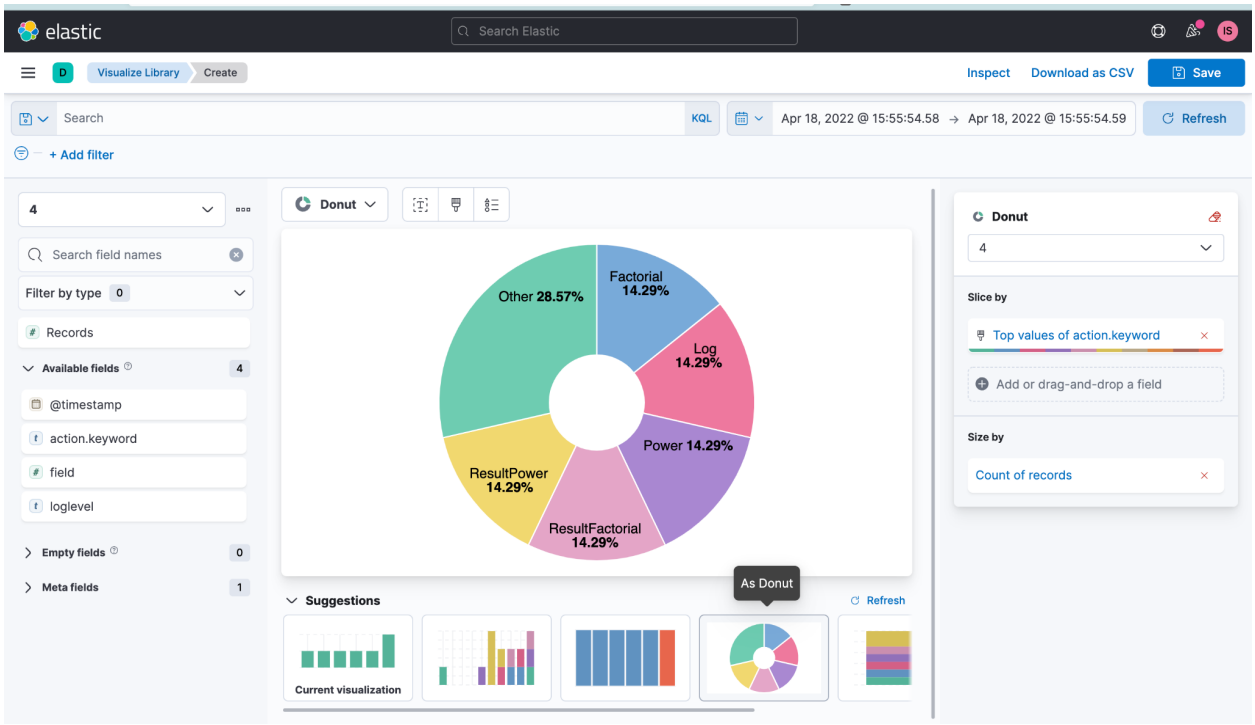
In the ingest pipeline section make sure you edit the grok part to the above statement. I have added the {WORD:action} part to the default statement so that we can make that message as a parameter. We also have the numeric values as a parameter.

Also fill in the index name. On click of import the pipeline will be completed you can see a graph and we can start visualizing the data.



Select action from the Available fields and click on visualize to see distributions of the executed operations.





Above are two ways of visualizing the Actions.

Code

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class MinicalcTest {
    private static final double DELTA = 1e-11;
    Minicalc calculator = new Minicalc();
    @Test
    public void test_root_TruePositive() {
        double actual=calculator.root( number1: 121.0);
        double exp=11.0;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_fact_TruePositive() {
        double actual=calculator.factorial( number1: 7);
        double exp=5040;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_ln_TruePositive() {
        double actual=calculator.log( number1: 1096.63315843);
        double exp=7;
        assertEquals(actual,exp, DELTA);
    }
    @Test
    public void test_power_TruePositive() {
        double actual=calculator.power(3.0,5.0);
        double exp=243.0;
        assertEquals(actual,exp, DELTA);
    }
```

```
                else if (input == 4) {
                    System.out.println("Enter a numbers");
                    System.out.println("Power function of the given number is : " + calculator.power(scanner.nextDouble(),
                }
                else {
                    System.out.println("exit option choosed");
                    break;
                }
            }
            System.out.println("cool stuff byeeee");
        }
        catch(InputMismatchException error)
        {
            System.out.println("Invalid input type");
            logger.error("Invalid input type");
        }
    }
    public double root(double n)
    {
        logger.info("Root - " + n);
        double ret = Math.sqrt(n);
        logger.info("ResultRoot - " + ret);
        return ret;
    }
    public double factorial(int n)
    {
        logger.info("Factorial - " + n);
        if(n<0)
            return Double.NaN;
        double ret=1;
        for(int i=1;i<=n;i++)
            ret=ret*i;
        logger.info("ResultFactorial - " + ret);
        return ret;
    }
```

```
        return ret;
    }
    public double log(double n)
    {
        logger.info("Log - " + n);
        double ret = Math.log(n);
        logger.info("ResultLog- " + ret);
        return ret;
    }
    public double power(double bas, double pow)
    {
        logger.info("Power - " + bas + ", " + pow);
        double ret = Math.pow(bas , pow);
        logger.info("ResultPower - " + ret);
        return ret;
    }
}
```

References

Github link: https://github.com/madhav2391/minicalculator.git

DockerHub link : madhavpasumarthy/minicalculator Tags | Docker Hub

The ELK Stack: From the Creators of Elasticsearch | Elastic