

CALCULATOR - Mini Project Report

IMT2020039 - Anshul Jindal

Introduction

As part of the project, I have developed a Calculator Program that performs the following operations:

1. Addition of 2 numbers.
2. Multiplication of 2 numbers
3. Factorial of a number
4. Square Root of a number

In this document, I intend to guide you through the code while also providing insights into the DevOps tools utilized in this project. I'll provide explanations about tools such as Jenkins, Docker, Ansible, Maven, and more.

Let us start with the basics of DevOps.

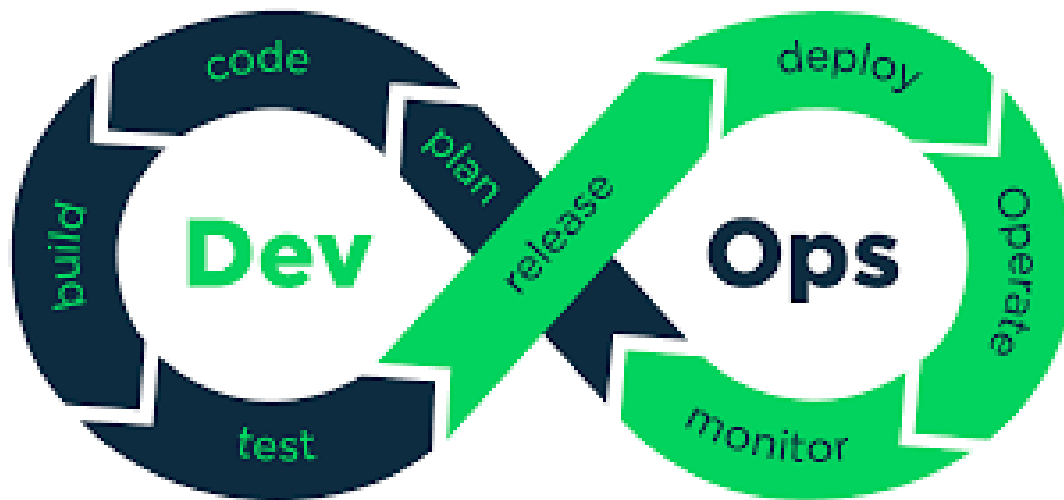
Important Links

1. [Github-Repo-Link](#)
 2. [DockerHub-Repo-Link](#)
-

Introduction to DevOps

DevOps is a software development approach that aims to break down the barriers between development (Dev) and operations (Ops) teams by promoting collaboration, communication, and automation. DevOps extend Agile principles beyond the boundaries of

“the code” to the entire delivered service. The DevOps approach was designed to ensure that high-quality updated software gets into the hands of users more quickly.



DevOps Model

Why to use DevOps

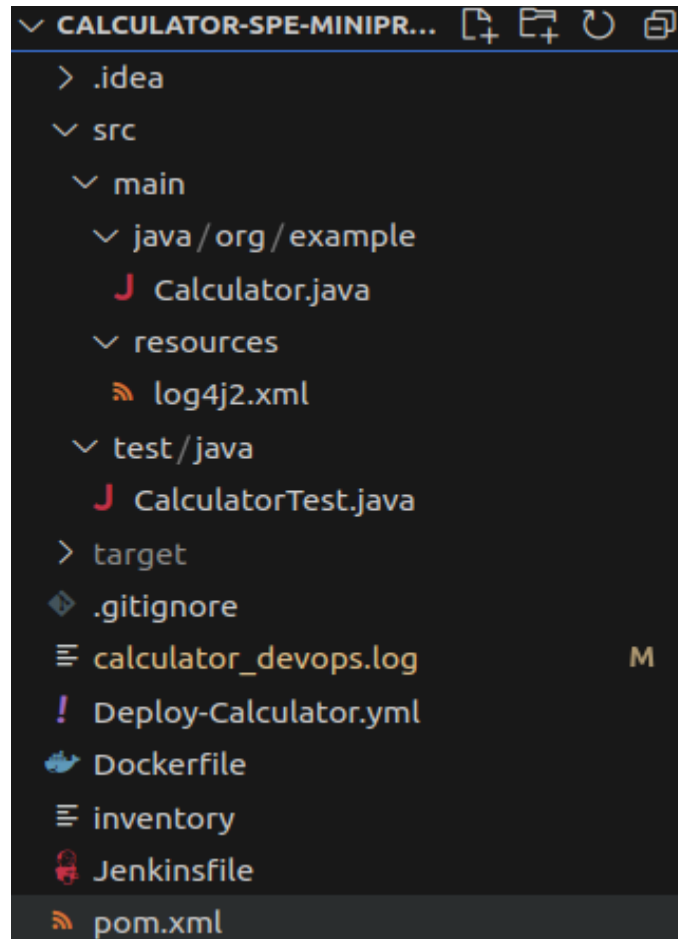
One of the key practices of DevOps is continuous integration and continuous delivery (CI/CD). CI/CD involves automating the process of building, testing, and deploying software, so that changes can be quickly and easily integrated into the production environment. Through this, DevOps can be beneficial in many ways:

- **Faster Time to Market:** One of the primary benefits of DevOps is that it enables organizations to deliver software updates more quickly. DevOps practices such as continuous integration and continuous delivery (CI/CD) help teams to automate the process of building, testing, and deploying software.
- **Increased Collaboration:** DevOps emphasizes collaboration between development and operations teams. Thus, miscommunications can be handled very effectively using DevOps methodology.

Code Files

Before we start talking about DevOps tools, let us first have a look at our file structure as well as a code for the calculator program and also the code for testing the program.

Folder Structure



Folder Structure

Here we have 2 directories mainly: **src** and **target**.

src is where developer writes his/her code and **target** is where all the outputs are generated. Inside the src directory, there is a **resources** directory that contains any resources relevant to the source code of the developer.

I have written my calculator program code in **Calculator.java** and testing code is present in **CalculatorTest.java**. We will talk about rest of the files later in this document.

Calculator.java

I have written a **Menu-Driven Program**. User has to select the operations that he/she wants to perform and then give further inputs accordingly.

The following is the look when we start the program:

```
----- Welcome to Calculator Program - SPE -----  
  
----- Please Enter the Operation Code: -----  
1. Addition  
2. Multiplication  
3. Factorial  
4. Square_Root  
5. Exit  
  
3  
Please Enter integer whose factorial you want to calculate: 10  
The Final Result is : 3628800  
anshul@anshul-Yoga-7-14ITL5:~$
```

Running of the Program

And the following are the snapshots of the functions that I have implemented in my program:

```
public static double Add(double n1, double n2)  
{  
    logger.info("Starting Add Operation");  
    double res = n1 + n2;  
    logger.info("Ending Add Operation");  
    return res;  
}  
  
public static double Multiply(double n1, double n2)  
{  
    logger.info("Starting Multiply Operation");  
    double res = n1*n2;  
    logger.info("Ending Multiply Operation");  
    return res;  
}
```

Functions Implemented

CalculatorTest.java

In this java file, I have tested all my functions that I have implemented in my original program. Testing has been done using **JUnit**.

Following is the snapshot of testing of Addition function:

```
@Test
public void testAddition(){
    double observed = Calculator.Add(12.9, 12.9);
    double expected = 25.8;
    Assert.assertEquals(observed, expected, 1e-10);
}
```

Test for Addition Function

This completes our code files and file structure. Now we will look at the DevOps tools that I have used to automate testing, deployment, etc. for this project.

DevOps Tools

In my project, I have used various DevOps tools and practices. We will look into all of them one by one.

Git and GitHub - For Version Control

Source Code Managment is an important jargon which is a practice of tracking and managing the changes made by the developer from time to time to the software code. This way the collaboration among the developers becomes easier and seamless.

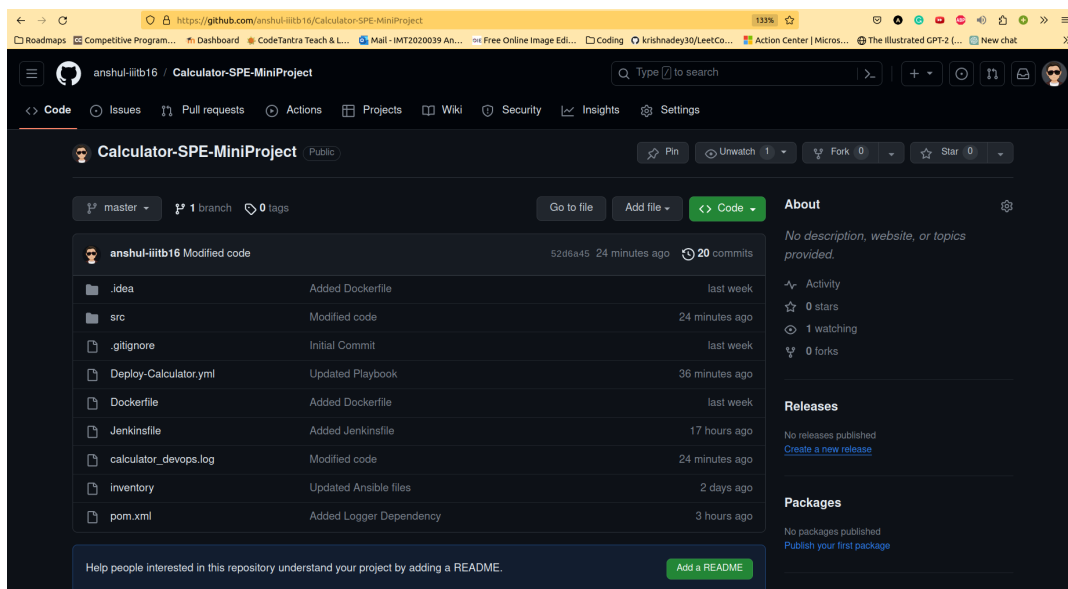
Git is a Distributed Version Control System (DVCS). It is one of the tools used for Source Code Management. Git allows developers to track the changes made in the code. It is a tool to manage your project source code history.

GitHub is a web based, git file hosting service which enables us to showcase/share our projects and files to others. It helps us to maintain remote repositories where various developers push their code, building and managing the software.

Following are some **Git Commands** that I have used for my project:

1. **git init:** It is used to initialise the local git repository in the developer's local system.
2. **git status:** Tells the active status, if commits have been done or not, and if any files were modified but not yet committed.
3. **git add:** It adds the specified files to the **Staging Area**.
4. **git commit -m "<imp commit message>":** Commits and save the version in the git.
5. **git remote add origin <URL>:** This command gathers all the committed files from your local repository and uploads them to our repository we created on github.
6. **git push:** This command will push all our local changes to our online repository that is our repository on Github.
7. **git clone:** The command git clone creates a new, local Git repository on your computer and copies all the files, commit histories, commit messages, branches, and etc.

Following is the snapshot of my GitHub Repository:



GitHub Repository

Maven - A Build Tool

Maven is a build tool which is provided by **Apache Software Foundation** and maven is completely Open-Source. It is a build tool that automatically build and test the application.

Why use Maven

Suppose a project needs third-party dependencies such as MySQL, Selenium, etc. So in order to run the project, we will first download the jar files of all the dependencies and then link it to our project. This process of downloading and linking has 2 major issues:

1. We have to download all the dependencies from different websites. Thus, this manual process is cuubersome.
2. Second issue is that if later in time, we update our project with latest dependencies. Then, we will have to first remove all existing libraries and download the latest ones. Hence, upgradation becomes difficult.

To overcome the above problems, we use Maven.

Working of Maven

Whenever we create a maven project, we get a **pom.xml** file (**Project Object Model**). Whole maven project is controlled by this pom.xml file. Also, a local maven repository is created.

We add the following to the pom.xml file:

1. **Dependencies** - Maven will automatically install libraries, drivers, etc that are used in your project once you mention them in pom.xml file.
2. **Plugins** - Plugins has all the configuration stuff needed to run the project.

Whatever dependencies we have mentioned in pom.xml file, maven will download the dependencies from the central/remote repository and bring it in your local repository. The website www.mvnrepository.com contains all kinds of third-party libraries. Then our project will refer to this local repository.

Apart from this, Maven also provides us with a default **Project Structure** with src directory and all. Maven also **packages your project** so that it can be easily shared with anyone.

For my project, I needed **JUnit Dependency** for testing and **org.apache.logging Dependency** for generating log file for the project. I also needed one plugin - **Maven Assembly Plugin** to package my project into a jar file.

Following is the format to write dependencies and plugins into the pom.xml file:

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/junit/junit -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.20.0</version>
  </dependency>

  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
  </dependency>
</dependencies>
```

Adding dependencies to pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
          <configuration>
            <archive>
              <manifest>
                <mainClass>Calculator</mainClass>
              </manifest>
            </archive>
            <descriptorRefs>
              <descriptorRef>jar-with-dependencies</descriptorRef>
            </descriptorRefs>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Adding Plugins to pom.xml

Maven Build Life Cycle

Maven Build Life Cycle are the series of steps that happens when we run maven project. Following are the steps that form the build cycle for maven:

- Validate
- Compile
- Test
- Package
- Integration Test (if integration is happening)
- Verify
- Install - install in local repository

One point to note here is that if we run any command, that command will automatically perform the operations that are below in the hierarchy. For example, if we run the command “mvn test”, then it will automatically compile also. Similarly, the command “**mvn install**” will compile, test and package our application.

Docker

Docker is an open-source platform that allows developers to build, ship, and run applications in containers. **Containers** are lightweight, portable, and self-contained units that encapsulate an application and all its dependencies, enabling it to run consistently across different environments and platforms.

Why use Docker

Suppose we built our python project in computer 1. Our project used python version 2.7 and computer 1 had the same version installed. Now let's say transferred our project files to computer 2 that had python version 3.8. Now our project will not run on computer 2 because of **Mismatched dependencies**.

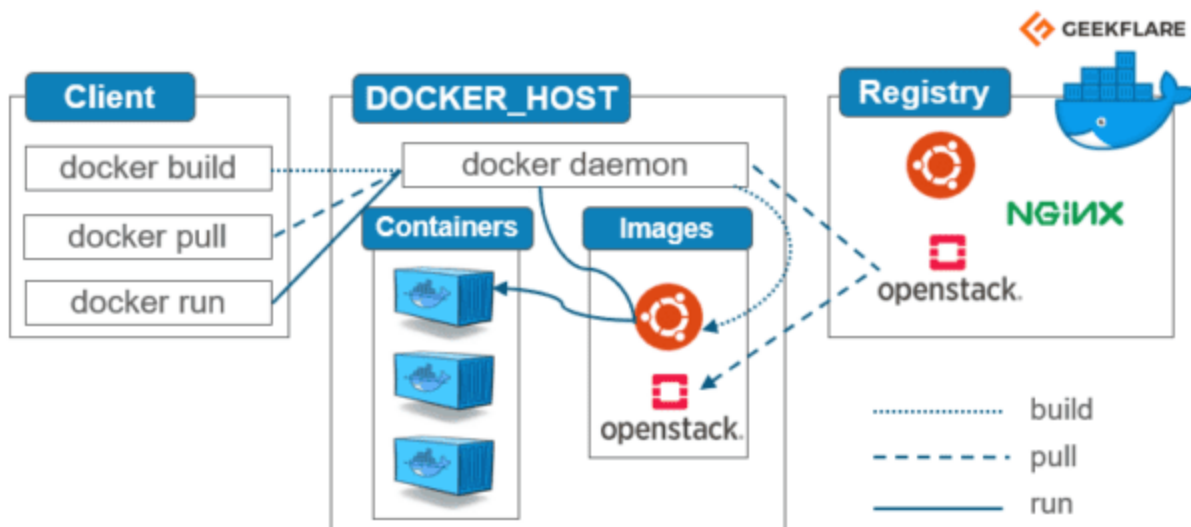
Generally, this issue arises when a developer is developing his code on computer 1 but tester will be testing the application on his computer 2.

Thus, to avoid the problem of Mismatched dependencies, we use docker. What we do is instead of taking just the project files, we create an image consisting of our project files as well as the dependencies using docker. Now we ship this image to computer 2 and run this image as a container with the help of docker.

Hence, through docker, we were able to package it along with the dependencies and then carry it across systems. **Containers run independent of each other**, thus we can run various containers having different versions of the same dependency.

Docker Architecture and Terminologies

Below is the simple architecture of Docker



Docker Architecture

Let us understand some of the important terms related to Docker:

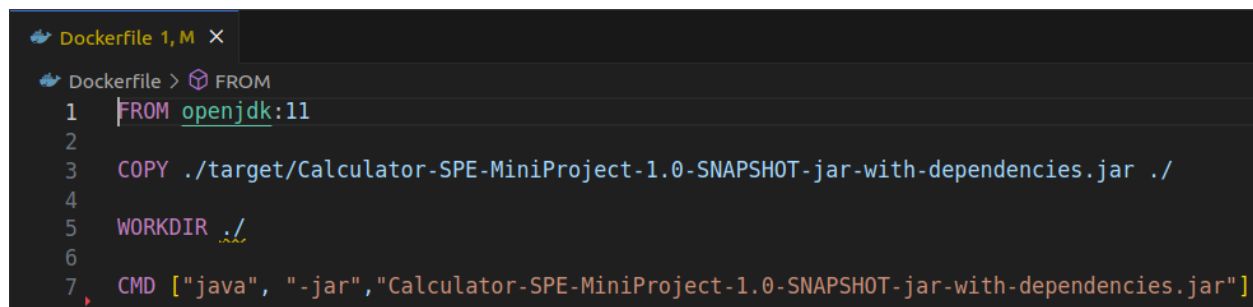
- **Docker Engine:**
 - Core part of the whole Docker system. Has **Client-Server Architecture**.
 - It is installed on the host machine. Client and Server interact via REST-API.
- **Docker Client:**
 - When any docker commands runs, the client sends them to dockerd daemon, which carries them out.

- **Docker Registries:**
 - It is the location where the Docker images are stored.
 - **Docker Hub** is the default place of docker images.
 - When you execute docker pull or docker run commands, the required docker image is pulled from the configured registry
- **Docker Images:**
 - Docker images are read-only templates with instructions to create a docker container.
 - Docker image can be pulled from a Docker-Hub.
- **Docker Containers:**
 - Docker images are run as Docker containers.
 - Containers can be called as running-images.

Working of Docker

In my project, I had to package the jar file that was created after running maven into an image. Image can be created through docker using a **Dockerfile**. Dockerfile is a text document that contains all the commands that a user can call on the command line to assemble an image.

Following is the snapshot of my Dockerfile:



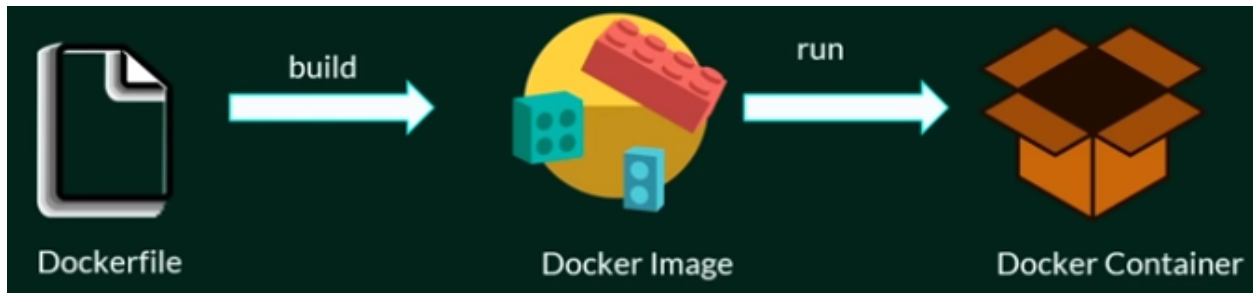
```
Dockerfile 1, M X
Dockerfile > FROM
1 FROM openjdk:11
2
3 COPY ./target/Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-dependencies.jar ./
4
5 WORKDIR ./
6
7 CMD ["java", "-jar", "Calculator-SPE-MiniProject-1.0-SNAPSHOT-jar-with-dependencies.jar"]
```

Dockerfile

Here as I need to have java installed on my system, I have used **base-image as openjdk version 11**.

Further I have written the commands to set the **Working Directory** as root in the image that will be created and I have copied the jar file into the root directory.

At last, I have mentioned the **command that will be executed** when the image will be run as a container. Basically, it is the command to run my calculator program.



Workflow

Docker Commands

Some of the docker commands that I have used for my project are as follows:

1. **"docker build"**: This command builds docker images using Dockerfile.
 - **"docker build -t my_image:latest ."**: -t is option to name and tag the image with whatever name and tag we want to. Here **name is my_image and tag is latest**. Also, "." signifies the path to dockerfile. Currently, we are running this command in the directory where dockerfile is present.
 - **"docker build -t anshul1601/calculator:latest"**: Here I have named the image same as my DockerHub username and tag is latest. This is required if you have to push your image to DockerHub.
2. **"docker images"**: This command is used to see the images that are created and present in our system (locally).
3. **"docker run --name <container_name> -it <image_name>"**: This command is used to run the image as a container.

-
- `—name` -> Allows us to name our container. Otherwise some random name it will give.
 - `-it` -> Allows to run the container in interactive mode. Otherwise the container will run for a second and exit itself.
4. **"docker ps"**: This command lists all the containers that are **currently in running status**.
 5. **"docker ps -a"**: This command lists all the containers irrespective of their status.
 - `-a` stands for `-all`.
 6. **"docker rmi -f <image_name>"**: This command will delete an image.
 - `-f` is force
 7. **"docker rm -f <container_name>"**: This command will delete a container.
 - `-f` is force
 8. **"docker login -u <DockerHub_username>"**: This command will login you to DockerHub from command line. After logging only, one can push the images to Hub.
 9. **"docker push <image_name:tag>"**: This command will push the image to Hub. Remember the image name has to be same as `<DockerHub_username/repo_name>`
 - **"docker push anshul1601/calculator:latest"** is the command I used.
 10. **"docker pull <image_name:tag>"**: This command will pull the image from Hub.
 - **"docker pull anshul1601/calculator:latest"** is the command I used.
 - Here tag is optional to mention.
 11. **"docker start -a -i <container_name>"**: this command will start any container.
 - `-a` = attaching STDOUT and `-i` = attaching STDIN
-

Ansible - Configuration Management Tool

Ansible is one of the configuration management tool that automates the deployment and management of infrastructure and applications.

Why use Ansible

Automation of configuration management plays a crucial role. Suppose we have 100 servers running. Now suppose that we have to upgrade a package. Now, no one would want to go to each server and then upgrade the package on each server manually. That would be a difficult task.

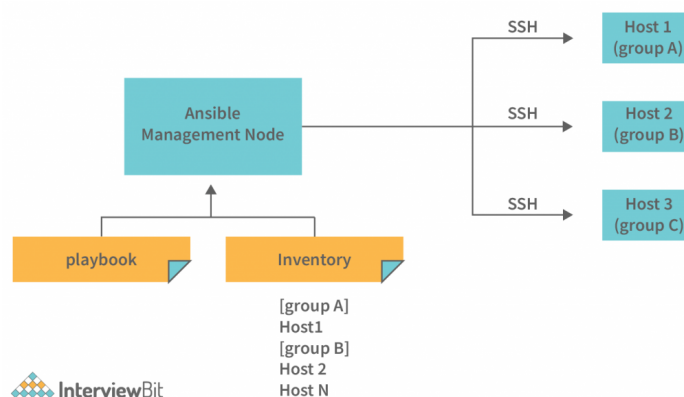
Thus, here is what configuration management system helps. There are various configuration management tools like **puppet, chef, Ansible, etc.**

Some advantages of Ansible are:

1. **Simple:** No special coding skills are required.
2. **Agentless:** You don't need to install any other software in the client systems you want to automate.

Working of Ansible

In Ansible architecture, there are two categories of systems: **Controller Node** and **Managed Hosts**.



Working of Ansible

-
- **Controller Node** - A device that runs Ansible Files and execute commands on other nodes.
 - **Managed Hosts** - The devices managed by the controller. They are also called clients that need to be configured.

Ansible works by connecting to Hosts on a network and sending a small program called an **Ansible Module** to these hosts. These modules are written in files known as **playbooks**.

- **Inventory** - This file is used to determine the set of managed hosts that controller node has to communicate with. Hosts can be defined by Port Number/Ip Address or Domain name. It also provides us with an option to make grouping among hosts. The advantage is that is we specify the group name, only the hosts under that group will be affected.
- **Playbook** - It contain one or more **plays** and each play contains one or more **tasks** that has to be executed in the **provided group of hosts**. Playbooks are written in **YAML format** (.yaml extension).

The syntax to write a playbook is as follows:

```
---
- name: Play 1 Name
  hosts: host_group_1
  tasks:
    - name: Task 11 Name
      module: parameters
    - name: Task 12 Name
      module: parameters
```

Playbook Syntax

In my project, the hosts are nothing but my own system (localhost). Hence in my inventory file, I have just 1 host.

Following is the snapshot of my inventory file that I made for this project:

```
inventory
1  [localhost]
2  127.0.0.1 ansible_connection = local ansible_user = anshul
```

Inventory File

Talking about Playbook, my target is to:

- Start the docker service
- Pull the image from DockerHub.
- Run that image as a container in the host (localhost here)

Hence, in my playbook file, I have defined 1 play, host is my **all (all nodes mentioned in inventory)** and I have set 3 tasks that I mentioned above. Here I have only one node that is my localhost.

Following is the snapshot of my playbook:

```
! Deploy-Calculator.yml X
! Deploy-Calculator.yml
1  ---
2  - name: Pull Docker Image of Calculator
3    hosts: all
4
5    tasks:
6      - name: Start Docker Service
7        service:
8          name: docker
9          state: started
10
11      - name: Pull Image
12        shell: docker pull anshul1601/calculator:latest
13
14      - name: Run the container
15        shell: docker create -it --name Calculator anshul1601/calculator
```

Playbook File

Ansible Commands

Following commands were useful for me during the project:

1. **"ansible-playbook <playbook_name>.yml -i <inventory_file_name>":** This will execute the Playbook file.

Jenkins

Jenkins is an open-source automation server that is used for continuous integration and continuous delivery (CI/CD) of software. Jenkins is designed to automate the software development process, from building and testing code to deploying it to production.

Why use Jenkins

Jenkins provides a range of features and tools for managing software projects, including:

- **Continuous Integration:** Jenkins provides built-in support for continuous integration, enabling developers to automatically build and test their code whenever changes are committed to the repository.
- **Continuous Delivery:** Jenkins provides support for continuous delivery, enabling developers to automatically deploy their code to production environments whenever it passes automated tests and meets predefined criteria.
- **Scalability:** Jenkins is designed to be highly scalable, enabling it to handle large and complex software projects. It can be run on a single server or distributed across multiple servers to handle high-volume workloads.

Using Jenkins

For this project, I made a new **pipeline style project** in Jenkins. Then I went to Configure and started writing **pipeline script**.

I implemented total of 6 stages in the pipeline. They are as follows:

1. Stage 1: **Cloning the GitHub Repository**

- First step is to clone the github repo where I have pushed all my files.
- Now once we have cloned the repo, we are ready for next stage of execution.

2. Stage 2: **Maven Build**

- I have run the command “**mvn clean install**” that will first clean and then install.
- In this stage, testing is performed and jar file is produced as output.

3. Stage 3: **Build Docker Image**

- Now once we have our jar file ready, we proceed towards making the docker image of the jar file.
- Here we need to add **Dockerfile** to the project.
- I ran the simple shell command “**docker build -t anshul1601/calculator:latest .**” to build the image.

4. Stage 4: **Pushing Docker Image to DockerHub**

- Now we will push the image that we created in stage 3 to DockerHub.
- First we need to login, for that we run “**docker login**” command.
- Then we run “**docker push**” command to push the image into DockerHub.

5. Stage 5: **Clearing Unwanted Images**

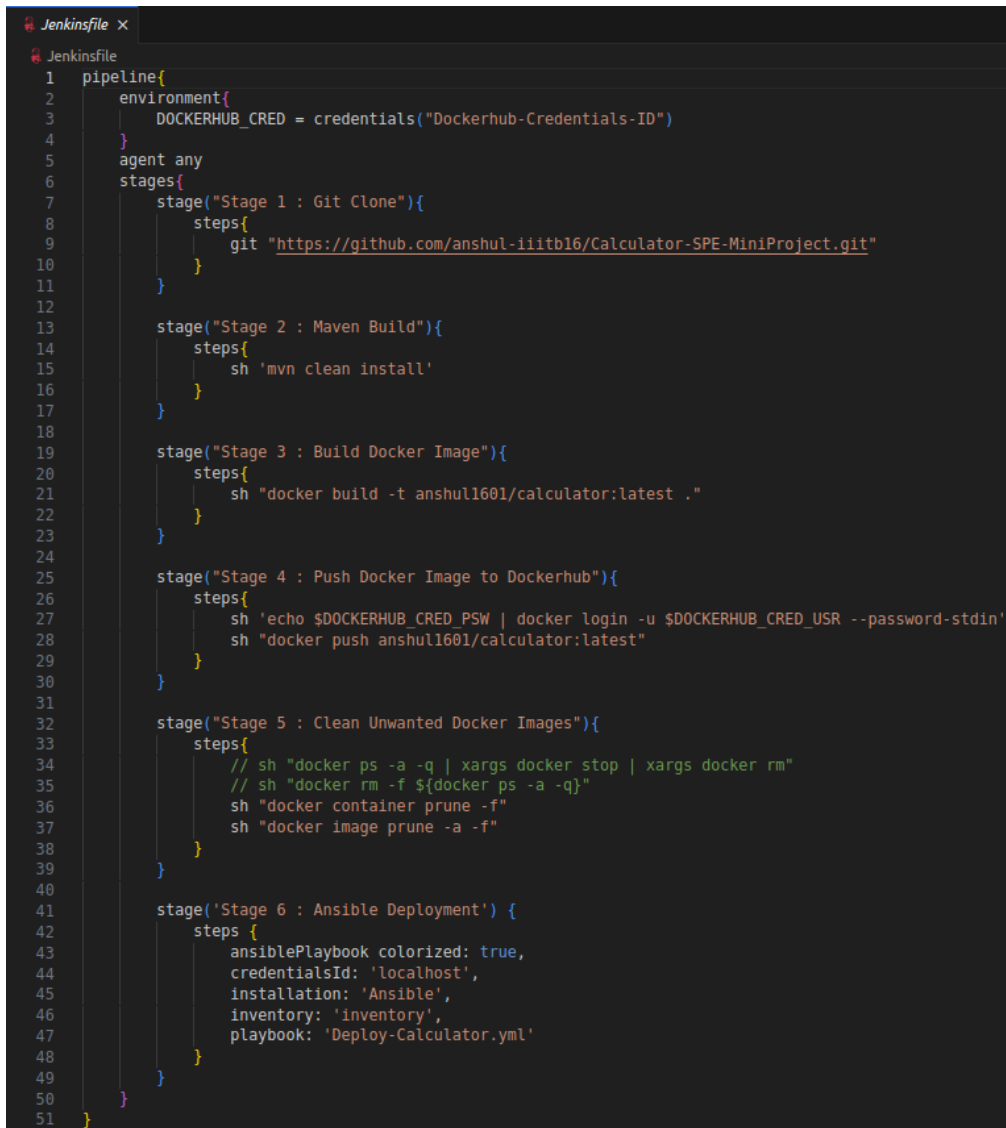
- Now after we ran our pipeline once, we are not able to run it again because we already have the images. Thus, before we run again the pipeline, the images need to be cleared.
- We cleared all the containers as well as all the images present in our system.

6. Stage 6: **Pulling Docker Image using Ansible**

- Here we need to add **inventory file** as well as **playbook file** to the project.
- In the playbook file here, we have mentioned the tasks to pull the image and run it as container.

Apart from implementing the above 6 stages, I also stored my **DockerHub credentials** (for pushing the image) as well as my **localhost credentials** (for Ansible) in Jenkins.

Snapshot of Jenkins file:



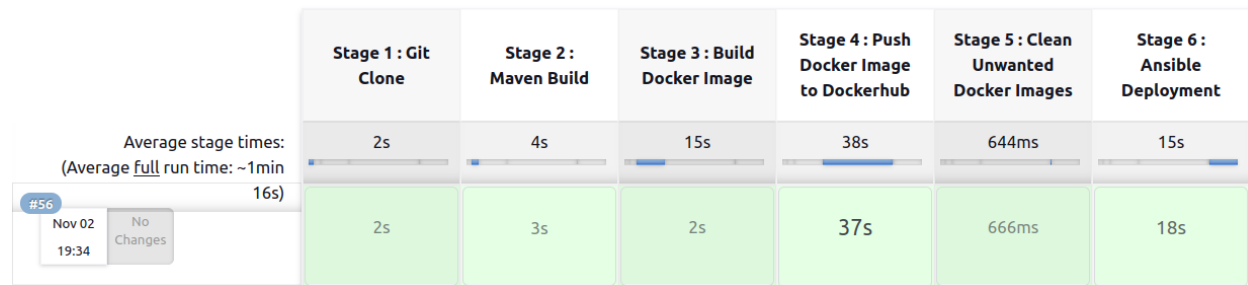
```
1 pipeline{
2   environment{
3     DOCKERHUB_CRED = credentials("Dockerhub-Credentials-ID")
4   }
5   agent any
6   stages{
7     stage("Stage 1 : Git Clone"){
8       steps{
9         git "https://github.com/anshul-iiitb16/Calculator-SPE-MiniProject.git"
10      }
11    }
12
13    stage("Stage 2 : Maven Build"){
14      steps{
15        sh 'mvn clean install'
16      }
17    }
18
19    stage("Stage 3 : Build Docker Image"){
20      steps{
21        sh "docker build -t anshul1601/calculator:latest ."
22      }
23    }
24
25    stage("Stage 4 : Push Docker Image to Dockerhub"){
26      steps{
27        sh 'echo $DOCKERHUB_CRED_PSW | docker login -u $DOCKERHUB_CRED_USR --password-stdin'
28        sh "docker push anshul1601/calculator:latest"
29      }
30    }
31
32    stage("Stage 5 : Clean Unwanted Docker Images"){
33      steps{
34        // sh "docker ps -a -q | xargs docker stop | xargs docker rm"
35        // sh "docker rm -f ${docker ps -a -q}"
36        sh "docker container prune -f"
37        sh "docker image prune -a -f"
38      }
39    }
40
41    stage('Stage 6 : Ansible Deployment') {
42      steps {
43        ansiblePlaybook colored: true,
44          credentialsId: 'localhost',
45          installation: 'Ansible',
46          inventory: 'inventory',
47          playbook: 'Deploy-Calculator.yml'
48      }
49    }
50  }
51 }
```

Jenkins Pipeline configuration

Running Jenkins Pipeline

Following is the image of my pipeline run:

Stage View



Pipeline Stages

NgRok and WebHooks

Now our task is to link GitHub with Jenkins to automate the building process whenever any commit is done on GitHub. For this, we use **NgRok and Webhooks**.

In order to achieve this, GitHub will have to poll jenkins whenever a commit is made. For this, we have to tell GitHub the IP Address where to ping on. The problem is that IP Address on which Jenkins is hosted in on our localhost which is not public. Thus GitHub requires some public IP to ping on and we want that public IP to later send the request to our local host i.e. to Jenkins.

Ngrok

It provides us with a temporary public IP. It will forward all the requests made to this temporary public IP to port 8080 where Jenkins is hosted.

```
anshul@anshul-Yoga-7-14iTL5: ~  
ngrok  
(Ctrl+C to quit)  
Introducing Always-On Global Server Load Balancer: https://ngrok.com/r/gslb  
Session Status  
Account Anshul Jindal (Plan: Free)  
Update update available (version 3.3.5, Ctrl-U to update)  
Version 3.3.4  
Region Asia Pacific (ap)  
Latency 61ms  
Web Interface http://127.0.0.1:4040  
Forwarding https://7d79-2a09-bac5-3b49-15f-00-23-423.ngrok-free.app -> http://localhost:8080  
Connections  
tll    opn    rt1    rt5    p50    p90  
0      0      0.00  0.00  0.00  0.00
```

Now this public IP that we have got, we will set up this public IP in GitHub and tell GitHub to sent the request to this public IP. For this, we use webhooks.

WebHooks

Webhooks allow external services to be notified when certain events happen. Following is the snapshot on how to add a Webhook in Github. That URL that we get from Ngrok has to be pasted in the payload URL field with adding **"/github-webhook/"** token at the end.

The screenshot shows the 'Webhooks / Add webhook' page in GitHub. It includes a description of the POST request, a 'Payload URL' field containing '79-2a09-bac5-3b49-15f-00-23-423.ngrok-free.app/github-webhook/', a 'Content type' dropdown set to 'application/x-www-form-urlencoded', an empty 'Secret' field, an 'SSL verification' section with 'Enable SSL verification' selected, a 'Which events would you like to trigger this webhook?' section with 'Just the push event.' selected, and an 'Active' checkbox checked. A green 'Add webhook' button is at the bottom.

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

79-2a09-bac5-3b49-15f-00-23-423.ngrok-free.app/github-webhook/

Content type

application/x-www-form-urlencoded

Secret

SSL verification

By default, we verify SSL certificates when delivering payloads.

☒ **Enable SSL verification** ☐ **Disable** (not recommended)

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me **everything**.

☐ Let me select individual events.

☒ **Active**

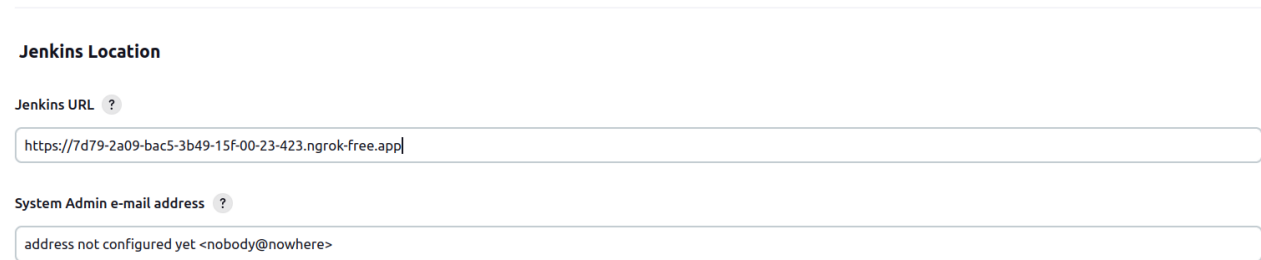
We will deliver event details when this hook is triggered.

Add webhook

Configure Jenkins

Once we are done setting up Webhook and ngrok, next is the set up Jenkins URL. We go to **Manage Jenkins -> configure system**.

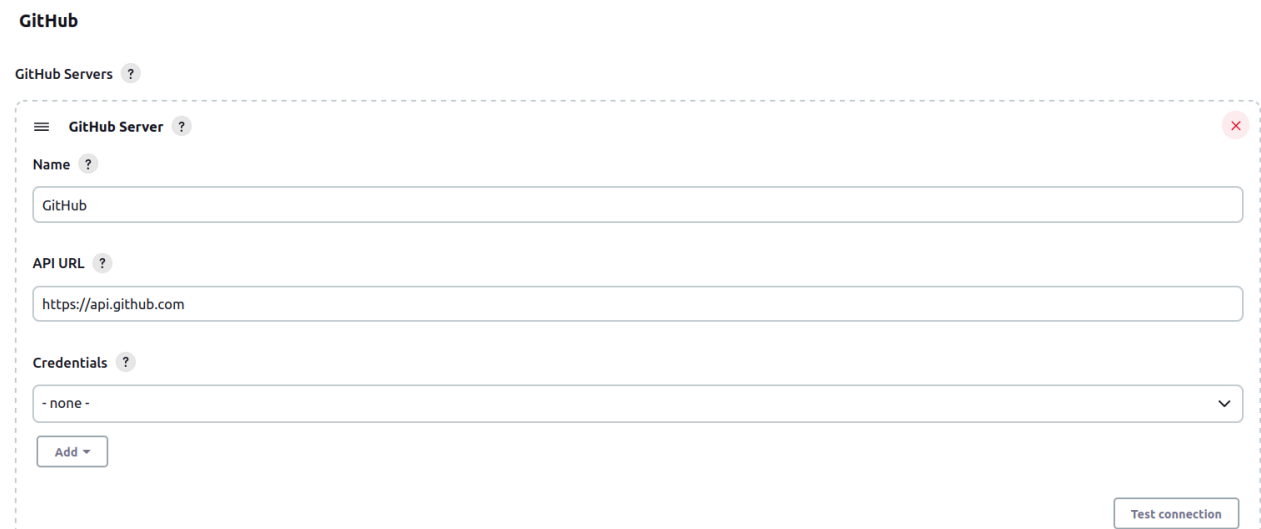
There we get a field named **Jenkins URL**. We will that URL provided to us by Ngrok in that field.



The screenshot shows the 'Jenkins Location' section of the Jenkins configuration page. It contains two input fields: 'Jenkins URL' with the value 'https://7d79-2a09-bac5-3b49-15f00-23-423.ngrok-free.app' and 'System Admin e-mail address' with the value 'address not configured yet <nobody@nowhere>'. Both fields have a help icon (?) to their right.

Set up Jenkins URL

Also here only scroll down and create one GitHub server:



The screenshot shows the 'GitHub' section of the Jenkins configuration page. It displays a list of 'GitHub Servers' with one server named 'GitHub' added. The server configuration details are shown in a dashed box: 'Name' is 'GitHub', 'API URL' is 'https://api.github.com', and 'Credentials' is set to '- none -'. There is an 'Add' button and a 'Test connection' button at the bottom right of the server configuration area.

Setting up a GitHub Server

Once we are done this the above step, we go to our project configuration and in Build Triggers section, we enable Github Poll SCM option.

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

Enabling Poll SCM

And we are done!

Just make a commit and it will automatically build now.

Running the Container

Following is the output when I run the container:

```
anshul@anshul-Yoga-7-14ITL5:~$ docker start -a -i Calculator
WARNING: sun.reflect.Reflection.getCallerClass is not supported. This will impact performance.
----- Welcome to Calculator Program - SPE -----

----- Please Enter the Operation Code: -----
1. Addition
2. Multiplication
3. Factorial
4. Square_Root
5. Exit

1
Please Enter 2 numbers to be added: 12 34
The Final Result is : 46.0
anshul@anshul-Yoga-7-14ITL5:~$
```

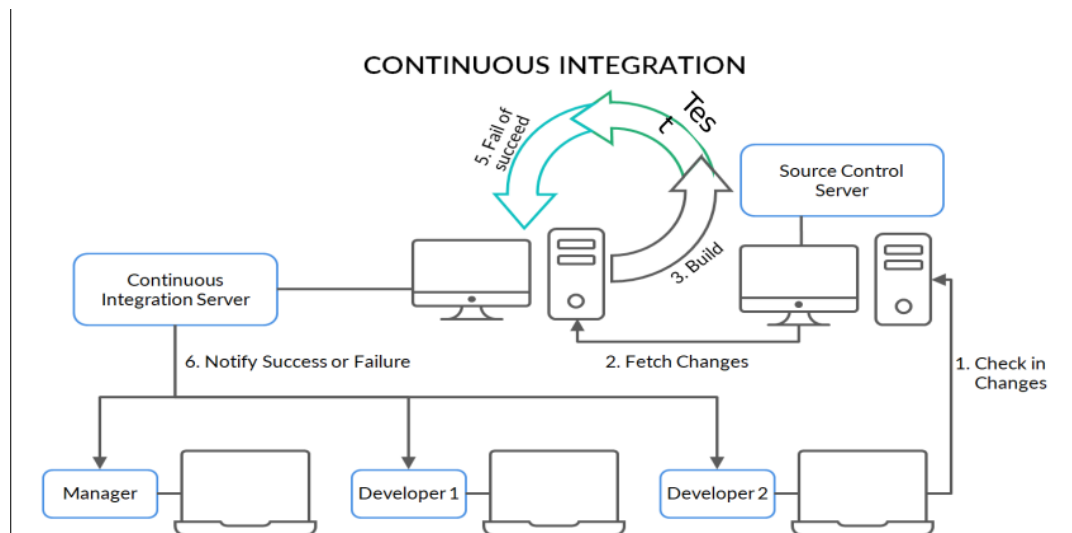
Running the Container

Achieving Continuous Integration

Now that we have learnt about all the tools and how exactly we have used them in our project, let's once look at how they together helped us achieve **Continuous Integration** for our project.

Continuous Integration

In a CI environment, developers regularly commit their code changes to a version control system (such as Git) several times a day. Each commit triggers an automated build and testing process. If any issues are detected during the automated build and testing process, developers are notified immediately.



Continuous Integration

With reference to the above diagram, we can performed all the steps given in the diagram through the following tools and techniques:

- **Step 1 - Check in Changes**
 - I have used **Git and GitHub** for Source Code Management.
 - GitHub remote repository is also maintained for the project.

- **Step 2 - Fetch Changes**

- I have set up Ngrok and Webhooks through which if I make any changes in my remote repository, GitHub pings Jenkins server for the same.

- **Step 3,4,5 - Building, Testing and Results**




- This is what exactly Jenkins does. Jenkins starts the build.
- Testing is done by Maven.
- If every stage of pipeline is successful, then the build is said to succeed or else it is said to fail.

Thus this is how through all the tools, we were able to achieve CI for our project.



References

I have referred to the following to get a better understanding of all the tools.



For Docker:

1.  Docker Tutorial for Beginners
2.   Complete Docker Tutorial in one video for Beginners in Hindi

For Maven

1.  Part 1 | Apache Maven Tutorial | Introduction
2.  Part 2 | Apache Maven Tutorial | maven Project | Folder Structure | Maven Bu...

For Ansible

1.  Day-14 | Configuration Management With Ansible | Puppet vs Ansible | Live Pro...
2.  Day-15 | Ansible Zero to Hero | #ansible #devops

THE END
