

CS 816 - Software Production Engineering

Mini Project - Scientific Calculator with DevOps

- Vivek Tangudu(IMT2020110)

GitHub Link

[vivektangudu123/mini](https://github.com/vivektangudu123/mini)

DockerHub Link

[bean6792/calculator](https://hub.docker.com/u/bean6792/calculator)

Pipeline Script

<https://github.com/vivektangudu123/mini/blob/main/Jenkinsfile>

CONTENTS

- Problem Statement
- Setting up tools
 - Java
 - Maven
 - IntelliJ
 - Jenkins
 - Docker
 - Ansible
 - Ngrok
 - Git
- Git workflow
- Folder Structure
- Running the container
- Challenges

Problem Statement

DevOps is like teamwork for making and launching software. It helps teams work well together to release apps faster. In our project, we're building a scientific calculator with tools like GitHub, Maven, JUnit, Jenkins, Ansible, and Docker. We're using Jenkins to make a special way of working called a pipeline, which helps us learn about Continuous Integration, Continuous Deployment, and managing settings.

In the past, developers and operations teams didn't always get along. Developers wanted to add new stuff, but operations folks worried about system stability. DevOps is like a peacemaker, helping everyone work together smoothly. Our project shows how DevOps can boost productivity by bringing developers and operations folks together.

We're going to create a basic calculator program in Java with functions like finding the addition, subtraction, multiplication and percentile functions. But our main focus is not on writing the code itself. Instead, we want to understand and set up the DevOps (Development and Operations) process. To do this, we'll use various tools like Git, GitHub, Jenkins, Docker, Ngrok, and Ansible.

Later, we'll put together a Jenkins pipeline. This pipeline will help us automate the steps of building, testing, and deploying our code with just a single click.

Tools used for:

- Source control management: git and Github
- To perform webhooks: ngrok

- Automate the build the process: Github webhook
- Code editor : IntelliJ
- Building and packaging the code: Maven
- Continuous Integration: jenkins

Installation and Setting up tools

1. Java

- brew install java
- Java -version
-

2. Maven

- brew install maven
- mnv -v

Maven helps to build a dependency management tool for Java projects that automates the compilation, testing, and distribution of your code.

3. Git

- brew install git
- git version

4. Docker

- Brew install docker
- Docker -version

Docker simplifies the process of developing, shipping, and running applications inside lightweight, portable containers.

5. IntelliJ

Installed through web :

<https://www.jetbrains.com/idea/download/>

6. Jenkins

- brew install jenkins
- brew install jenkins-lts
- brew services start jenkins-lts
- Open localhost:8080
- Install required plugins.
- It will give a path where we can find the initial password and create a new user in jenkins.
- Install plugins related to docker,ansible,git,github.
-

Jenkins helps to automate the delivery process and facilitate continuous integration and continuous delivery (CI/CD).

7. ngrok

- brew install ngrok
- ngrok -version
- ngrok http 8080

8. Ansible

- Brew install ansible
- ansible --version

Ansible simplifies tasks like software installation, updates, configuration changes, and application deployments across multiple machines. Ansible uses a simple and human-readable language to define automation tasks, making it easy to create and maintain automation scripts.

```
vivek@viveks-MacBook-Air calculator % mvn -v
Apache Maven 3.9.5 (57804ffe001d7215b5e7bcb531cf83df38f93546)
Maven home: /opt/homebrew/Cellar/maven/3.9.5/libexec
Java version: 21.0.1, vendor: Oracle Corporation, runtime: /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home
Default locale: en_AU, platform encoding: UTF-8
OS name: "mac os x", version: "14.0", arch: "aarch64", family: "mac"
vivek@viveks-MacBook-Air calculator %
```

```
vivek@viveks-MacBook-Air calculator % java -version
java version "21.0.1" 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 21.0.1+12-LTS-29)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.1+12-LTS-29, mixed mode, sharing)
vivek@viveks-MacBook-Air calculator %
```

```
ngrok (Ctrl+C to quit)
Introducing Always-On Global Server Load Balancer: https://ngrok.com/r/gslb

Session Status      online
Account             vivektangudu@gmail.com (Plan: Free)
Version             3.3.5
Region              India (in)
Latency              40ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://ce40-103-156-19-229.ngrok-free.app -> http

Connections          ttl      opn      rt1      rt5      p50      p90
0                    0        0        0.00     0.00     0.00     0.00
```

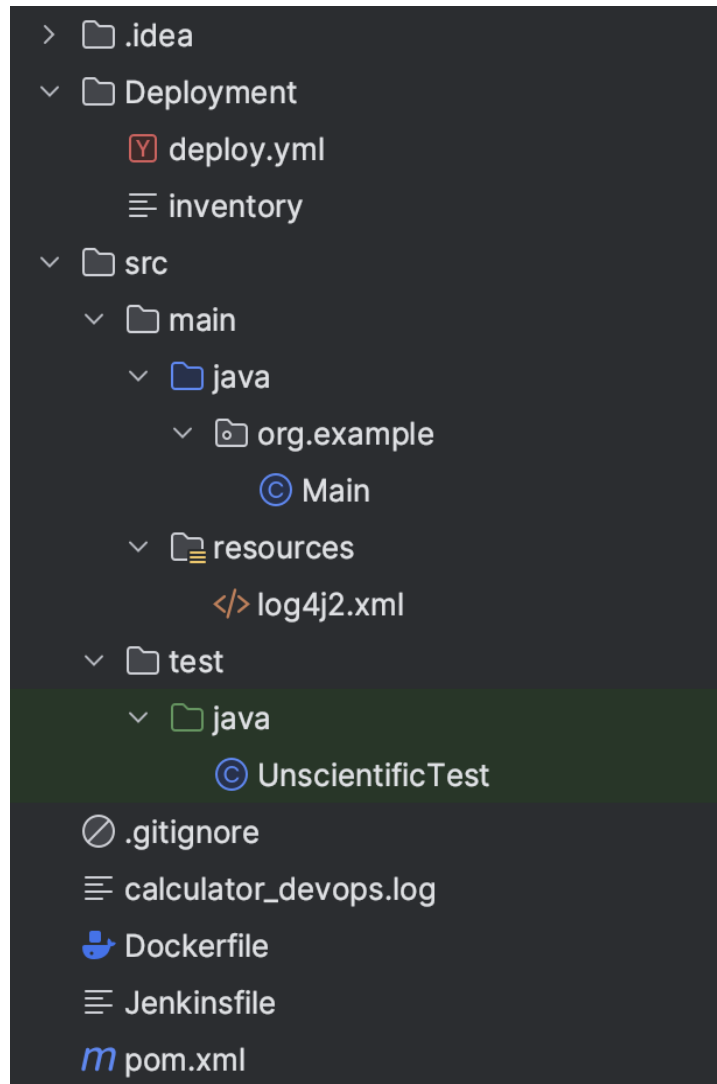
Git workflow

Build the project in the local repository that uses maven to build, now I need to push the code into github to automate the build process. It also helps in version control and helps to save the code of several versions.

- Initiate git in the local directory: *git init*
- To check the status of repository: *git status*
- To clone the remote repo to local : *git clone < remote repo URL >*
- After adding files in local repository, we add to the staging area : *git add*
- Commit changes: *git commit -m "message"*
- To check the changes and who made the changes and also with branches: *git log*

- Push into the remote repo: *git push*
- Add the link to the current local repository with the use of git remote add :
git remote add origin < link of remote repo >

Folder Structure



.idea – config files for intellij

Deployment - Files for Ansible

Src - java source code

Resources - config files for log4j2 dependency

.gitignore - git ignores files mentioned here

Pom.xml - config file for maven

Dockerfile - config file for docker

Jenkinsfile - pipeline script for jenkins


Create a repo in github:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Owner *

 sivanich9 ▾

Repository name *

/ Calculator 

Great repository names are short and memorable. Need inspiration? How about **ubiquitous-doodle?**

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

IntelliJ , Maven and JUnit :

We use Java to build our project as it is system independent and we use IDE IntelliJ. Maven streamlines our Java project development by automating builds. It's Java-based and makes it effortless to incorporate dependencies and generate executable JAR files, which helps to execute on any platform. Maven's Project Object Model (POM) outlines the project structure, its dependencies, and the build process. The POM.xml file holds project details like name, version, and dependencies. Adding project dependencies is a simple process, simply update the pom.xml file. Maven ensures easy dependency management, facilitating the conversion of projects into portable JAR files. Its extensive plugins empower developers to deploy, document, and test applications seamlessly.

We use JUnit to check our code. In our project, we have three main tags: @before, @test, and @after. The @test tag tells us which methods are for testing. Before every test, the @before method runs, and after every test, the @after method runs. This helps set up and finish tests properly. We use assertEquals to see if our test results are as expected. If not, it gives us a message about the error. We also use assertNotEquals to check if results are different when they should be.

Add JUnit, logger dependencies in pom.xml file



```
<dependencies>
  <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>2.14.0</version>
  </dependency>
</dependencies>
```

JUnit helps to test the code and logger helps to write logs in a log file when executing the code.

Along with the dependencies, we also include a maven plugin, to create a jar with all dependencies, eliminating the need to separately install these dependencies on the system where the jar file will be executed.

We have to create "log4j.xml" in "calculator/src/main/resources/log4j2.xml".

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
  <Appenders>
    <Console name="ConsoleAppender" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
    </Console>
    <File name="FileAppender" fileName="calculator_devops.log" immediateFlush="false" append="true">
      <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </File>
  </Appenders>
  <Loggers>
    <Root level="debug">
      <AppenderRef ref="FileAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

Log4j2.xml is a configuration file of how to set up logging in a Java application using Log4j2.xml, allowing developers to track and analyze the application's behavior by logging various messages.

We set a pattern of message that needs to be written in the format of "%d{dd/MM/yyyy:HH:mm:ss SSS}{IST} [%F] [%level] %logger{36} %msg%n"

Add logger method in main.java and logger.info in methods of main.java

" private static final Logger *logger* = LogManager.getLogger(Main.class);"

Run test cases using maven - *mvn test*

```
[INFO] Running calTest
ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleLogger to log to the console...

The Result is 0.0

The Result is 4.0

The Result is 4.0

The Result is 2.0

[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.046 s -- in calTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
```

Build the jar file with maven using the command: *maven install*

```
[INFO] --- install:3.1.1:install (default-install) @ cal ---
[INFO] Installing /Users/vivek/Desktop/calculator/pom.xml to /Users/vivek/.m2/repository/org/example/cal/1.0-SNAPSHOT/cal-1.0-SNAPSHOT.pom
[INFO] Installing /Users/vivek/Desktop/calculator/target/cal-1.0-SNAPSHOT.jar to /Users/vivek/.m2/repository/org/example/cal/1.0-SNAPSHOT/cal-1.0-SNAPSHOT.jar
[INFO] Installing /Users/vivek/Desktop/calculator/target/cal-1.0-SNAPSHOT-jar-with-dependencies.jar to /Users/vivek/.m2/repository/org/example/cal/1.0-SNAPSHOT/cal-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.116 s
[INFO] Finished at: 2023-10-31T11:28:16+05:30
[INFO] -----
```

This created a folder named target and we have the jar that we can run manually:

```
vivek@viveks-MacBook-Air calculator % cd target
vivek@viveks-MacBook-Air target % ls
archive-tmp                                generated-sources                          surefire-reports
cal-1.0-SNAPSHOT-jar-with-dependencies.jar  generated-test-sources                    test-classes
cal-1.0-SNAPSHOT.jar                       maven-archiver
classes                                    maven-status
```

To run the jar file:

Move to the target folder.

Java -jar <name of jar folder >

Now, it's time to push our code to github.using the commands:

git add . && git commit -m "<message> && git push

We need to provide a git username and auth token to push the code.

```
vivek@viveks-MacBook-Air calculator % git add . && git commit -m "made some changes" && git push
[master 38b29fe] made some changes
Committer: vivek <vivek@viveks-MacBook-Air.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

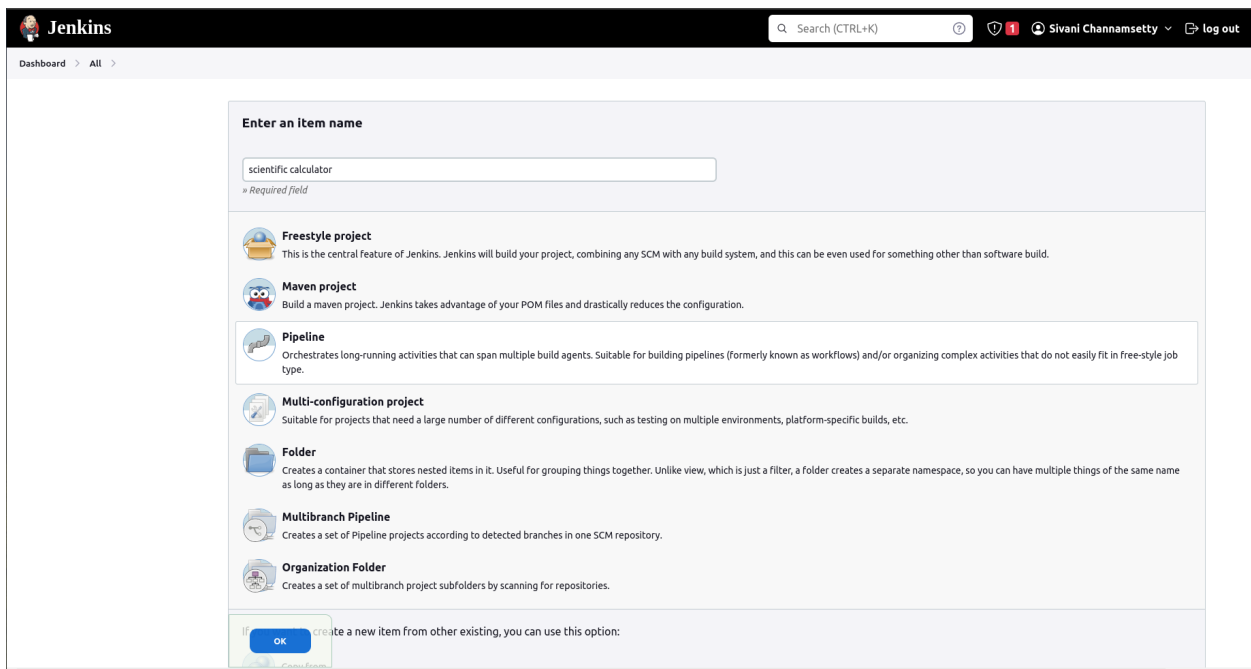
    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

2 files changed, 25 insertions(+), 7 deletions(-)
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 511 bytes | 511.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/vivektangudu123/calculator.git
335b2a3..38b29fe master -> master
```

→ This is the time to create a new pipeline in jenkins.



Step 1: Git clone

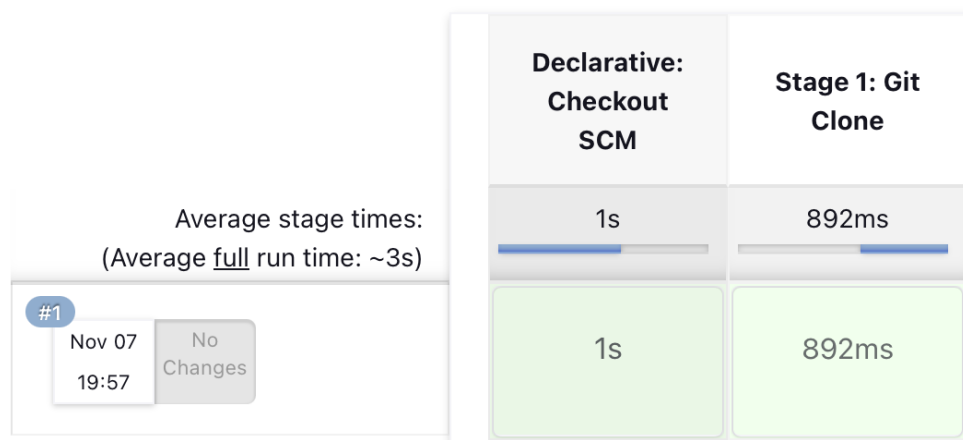
In configure, add the pipeline script

```
stages{
    stage('Stage 1: Git Clone'){
        steps{
            git branch: 'master',
              url: 'https://github.com/vivektangudu123/calculator.git'
        }
    }
}
```

It clones the repo from the url in that branch (master).

As the repo is public, no need to give credentials of github. If the repo is private, we need to do so.

Stage View



Step 2: Build using maven

Second step is to build using the code.

```

stage('Step 2: Maven Build'){
    steps{
        sh 'mvn clean install'
    }
}

```

-> mvn clean install will build the jar file. One with dependencies and another with not.



Step 3 : Build docker image

We would be to create docker image using docker

Create a Dockerfile in the project folder and add the commands to create a dockerfile.

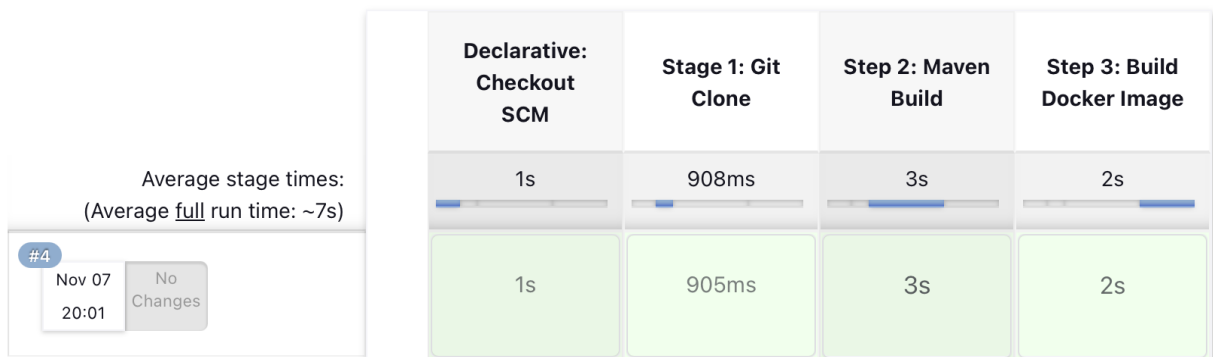
```
FROM openjdk:11
COPY ./target/cal-1.0.0-jar-with-dependencies.jar ./
WORKDIR ./
CMD ["java", "-cp", "cal-1.0.0-jar-with-dependencies.jar", "org.example.Main"]
```

In the jenkins pipeline the next step is to build the image.

"Bean6792" is my username of docker

```
stage('Step 3: Build Docker Image'){
    steps {
        script {
            docker_image = docker.build "bean6792/calculator:latest"
        }
    }
}
```

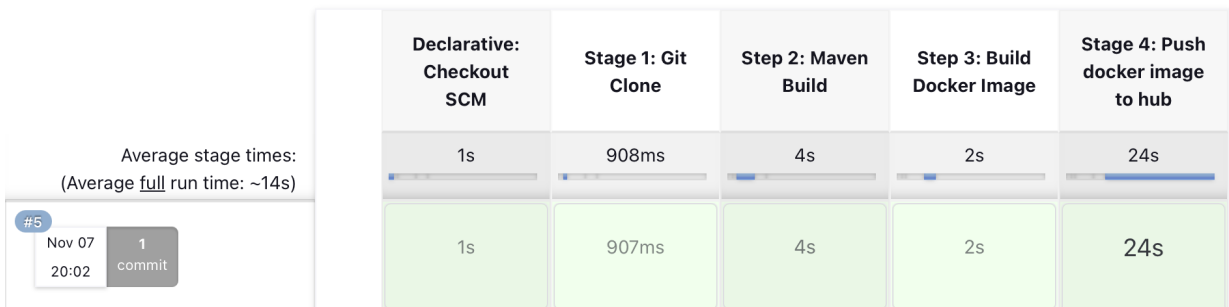
In the terminal, we can run the command - *docker images* to see the created images in the system.



Step 4: Push the docker image into hub

```
stage('Stage 4: Push docker image to hub') {  
    steps{  
        script{  
            docker.withRegistry('', 'docker-red'){  
                docker_image.push()  
            }  
        }  
    }  
}
```

Docker-red has my docker credentials in jenkins credentials. This will push the created image to the remote hub.



Step 5 : Clean docker images

```
stage('Stage 5: Clean docker images'){
    steps{
        script{
            sh 'docker container prune -f'
            sh 'docker image prune -f'
            sh 'docker ps -aq | xargs docker stop | xargs docker rm'
        }
    }
}
```

Average stage times: (Average full run time: ~19s)		Declarative: Checkout SCM	Stage 1: Git Clone	Step 2: Maven Build	Step 3: Build Docker Image	Stage 4: Push docker image to hub	Stage 5: Clean docker images
		1s	913ms	3s	3s	23s	2s
		1s	937ms	3s	4s	22s	2s

#6
Nov 07
20:03
1
commit

Step 6: Ansible

Docker container prune -f && docker image prune -f will clean all the images and containers which are stopped.

Docker -ps -aq will list all containers and the remaining two will delete all containers.

```

stage('Step 6: Ansible Deployment'){
    steps{
        ansiblePlaybook becomeUser: null,
        colorized: true,
        credentialsId: 'localhost',
        disableHostKeyChecking: true,
        installation: 'Ansible',
        inventory: 'Deployment/inventory',
        playbook: 'Deployment/deploy.yml',
        sudoUser: null
    }
}

```

\$ ansible-playbook Deployment/inventory -i Deployment/inventory

This is the command that is going to run. Deploy.yml is the name of the playbook file that will be executed. The inventory file that Ansible should use to determine which host is going to run the playbook.

We have the commands that need to run the docker in deploy.yml

```

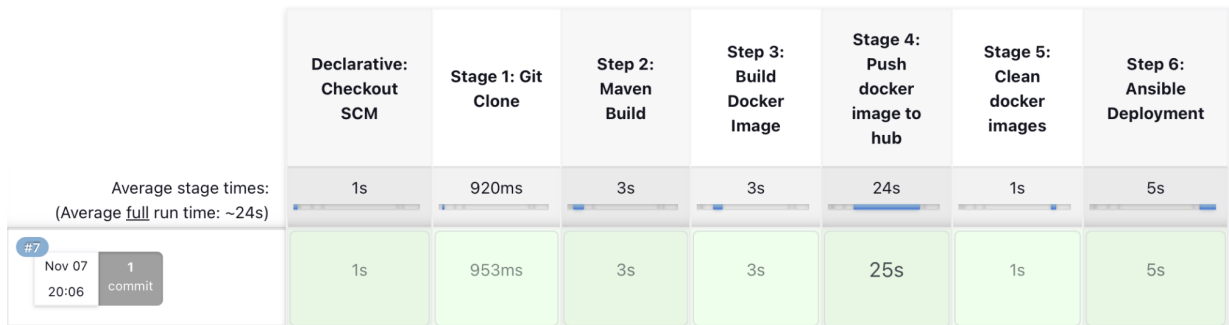
- name: Pull Docker image of Calculator
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3
  tasks:
    - name: Pull image
      shell: docker pull bean6792/calculator:latest
    - name: Running container
      shell: docker run -it -d --name calculator_container bean6792/calculator

```

We have two tasks

Task1: Pull the image `docker pull <docker image name with tag or image id>`

Task2 : Run the image `docker run -it -d --name <container name> <image_name>`



Webhook

To automate the build process, let's create a webhook to automate the build process when a change is made.

Using ngrok we can create a link which can trigger the change from github. Using the command

Ngrok http 8080

```
ngrok (Ctrl+C to quit)
Introducing Always-On Global Server Load Balancer: https://ngrok.com/r/gslb

Session Status      online
Account             vivektangudu@gmail.com (Plan: Free)
Version             3.4.0
Region              India (in)
Latency              -
Web Interface        http://127.0.0.1:4040
Forwarding            https://e117-103-156-19-229.ngrok-free.app -> http

Connections          ttl      opn      rt1      rt5      p50      p90
                     0        0        0.00     0.00     0.00     0.00
```

Change the jenkins URL in the system and paste the given link. Add A github server with credentials.

Add a webhook in the repo settings of the repo.

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 *

`https://2a65-103-156-19-229.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.

When the trigger is triggered, github sends the changes to the port 8080 as we configured before.

✓		e959e69a-7c88-11ee-9698-a907cd150a14	push	2023-11-06 15:13:18	...
✓		2a8d4824-7c88-11ee-924a-538c8bfb0c6f	push	2023-11-06 15:07:58	...
✓		44cb254a-7c87-11ee-8b93-b2d3be02d56e	push	2023-11-06 15:01:32	...
✓		524e7740-7c86-11ee-8902-ec13e1ef772a	ping	2023-11-06 14:54:45	...

Now When there is a change in the code and that was pushed to github, jenkins automates the build process and tests the code with tests that were given.

Let's run the docker image

```
vivek@viveks-MacBook-Air calculator % docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
bean6792/calculator latest          2bc273703cc6   2 minutes ago   645MB
vivek@viveks-MacBook-Air calculator % docker pull bean6792/calculator:latest
latest: Pulling from bean6792/calculator
Digest: sha256:b30eb658dd0731fb17b68b85c3aac74271fcfa30d4fb812d436b619b5990d2fa
Status: Image is up to date for bean6792/calculator:latest
docker.io/bean6792/calculator:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview bean6792/calculator:latest
vivek@viveks-MacBook-Air calculator %
```

The 1st command lists all the images that were present in system *docker images*

2nd command will run the image and creates an container *docker run -it -d --name <new name for container> <image name>*

```
vivek@viveks-MacBook-Air calculator % docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS          NAMES
c9b53d8fa1bf   bean6792/calculator "java -cp cal-1.0.0-..." 4 seconds ago  Up 3 seconds  8080           calc
4edf4aafa3a9   bean6792/calculator "java -cp cal-1.0.0-..." 10 minutes ago Up 2 minutes  8080           calculator_container
```

To execute the container : *docker start -a -i <name of container>*

```
vivek@viveks-MacBook-Air calculator % docker start -a -i calc

Operation:
1. Add
2. Subtract
3. Percentile function
4. Multiply
5. Exit

Enter your choice(number):1

Add two numbers!!
Enter number 1:
2
Enter number 2:
3

The Result is 5.0
```

Log file in that container

```
2023-11-07 13:46:21.428 [main] INFO org.example.Main - Calculator Begins!!
2023-11-07 13:46:30.338 [main] INFO org.example.Main - Calculator Begins!!
2023-11-07 13:46:34.536 [main] INFO org.example.Main - Executing addition for 2.0 and 3.0. Result: 5.0

2023-11-07 13:46:38.654 [main] INFO org.example.Main - Executing percentile operation for 3.0 and 2.0. Result: 1.0

2023-11-07 13:53:35.328 [main] INFO org.example.Main - Calculator Begins!!
2023-11-07 13:53:40.196 [main] INFO org.example.Main - Executing addition for 2.0 and 3.0. Result: 5.0
```

Challenges

- 1) Initially Jenkins ran on <http://127.0.0.1:8080> but not on localhost:8080. Reset that in the config file.
- 2) Jenkins needs to know the bin folder of the system's bin paths to execute Maven, Python or anything. Need to change a file in .jenkins and add paths.
- 3) Not able to understand the syntax in deploy.yml. And tried to rewrite the script path and wrote the command in shell script.