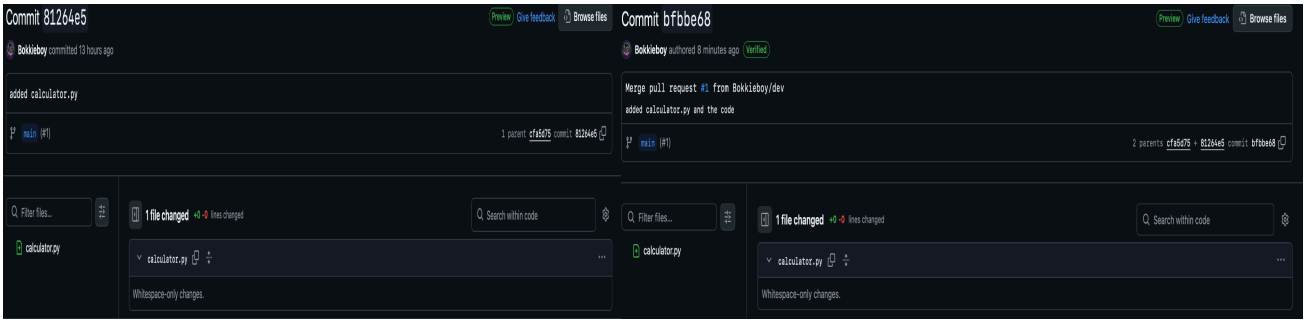
Network Development

Github repository setup and integration



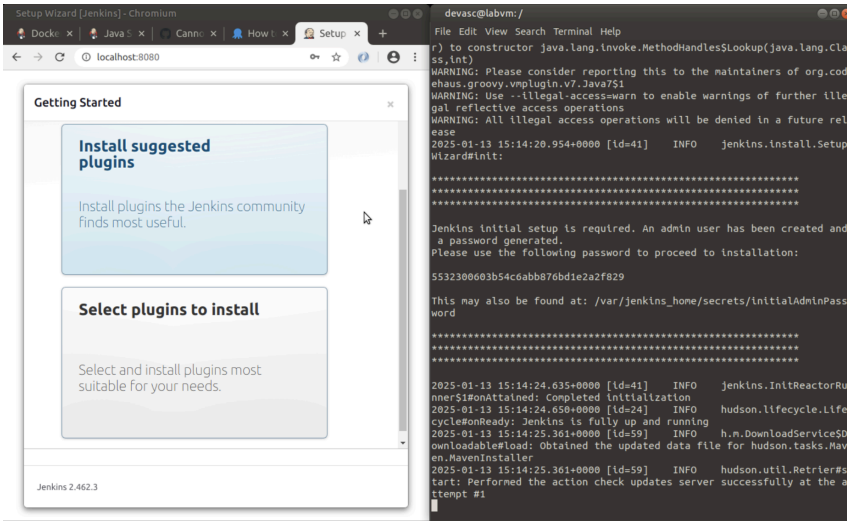
First I created the github repository - https://github.com/Bokkieboy/UniCalculator with two branches, main and dev. I then committed calculator.py to the dev branch. I then added the code for the application and committed that to the dev branch and then merged dev to the main branch with a pull request:

Next I installed jenkins to the DEVASC vm. I did this by first needing to update docker on the VM as it was outdated, then I ran the command:

```
docker run -d --name jenkins \
  -p 8080:8080 -p 50000:50000 \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v jenkins_home:/var/jenkins_home \
  jenkins/jenkins:lts
```

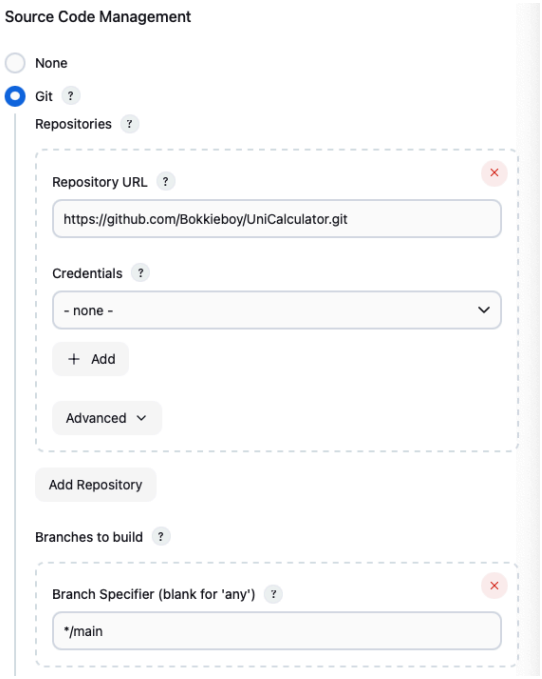
This started Jenkins on port 8080 while also giving it permission to user docker which will be needed on Task 4. It pulls the latest jenkins image and then runs.

Jenkins Configuration and Automation



I then installed all suggested plugins which included git, github branch source and pipeline. These plugins will allow me to complete the task.

I then created a new item which I called “Github”, then I freestyle project.



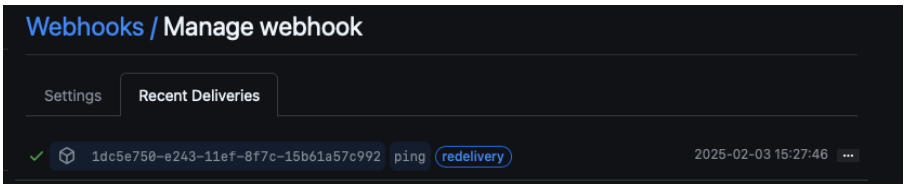
Then under the Source Code Management I selected git for the SCM, added my repository URL and specified the branch to */main.

Under build triggers I enabled Poll SCM and under the Build Steps I added a execute shell:
echo "Starting build"
python3 calculator.py
This says its starting the build then runs the script on python3

```
[Github-CICD] $ /bin/sh
/tmp/jenkins14598416862
+ echo Starting build
Starting build
+ python3 calculator.py
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4):
```

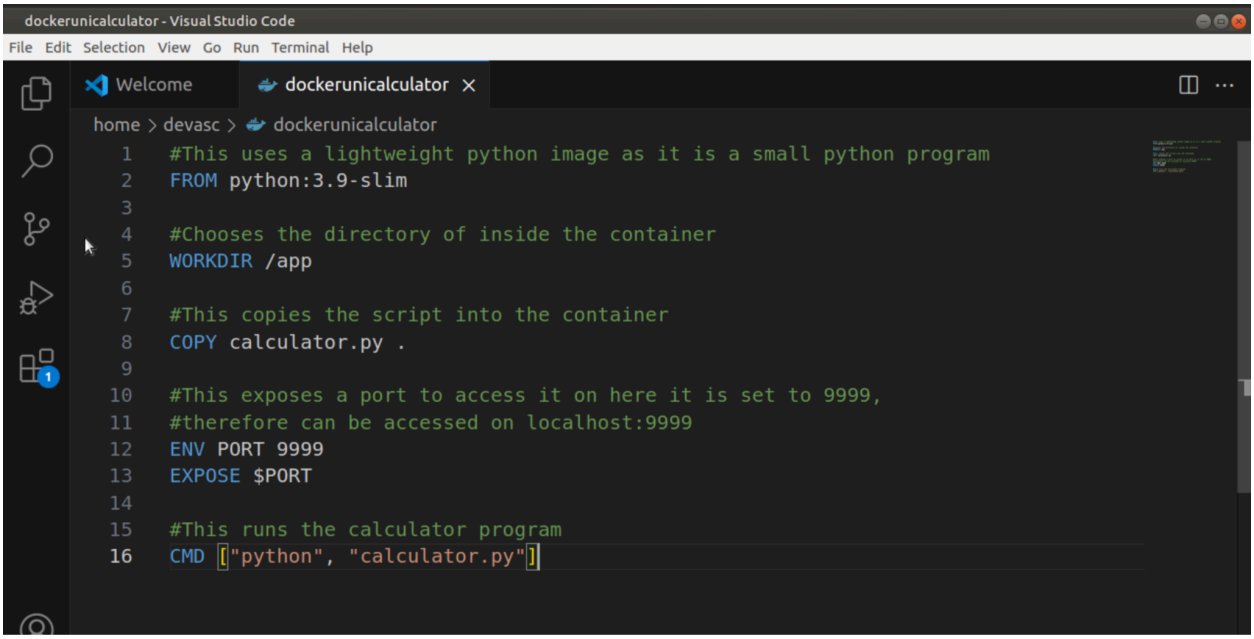
Here you can see the python script building and running.

I then created a github webhook to communicate with Jenkins by port forwarding the port 8080 and setting the payload URL to http://82.2.119.118:8080/github-webhook/ . This allows Github to instantly notify Jenkins when a push event occurs, reducing delays. It saves resources as Jenkins doesn't have to constantly poll, and it makes sure that every commit is built, tested and deployed automatically. I also changed the content type to application/json.

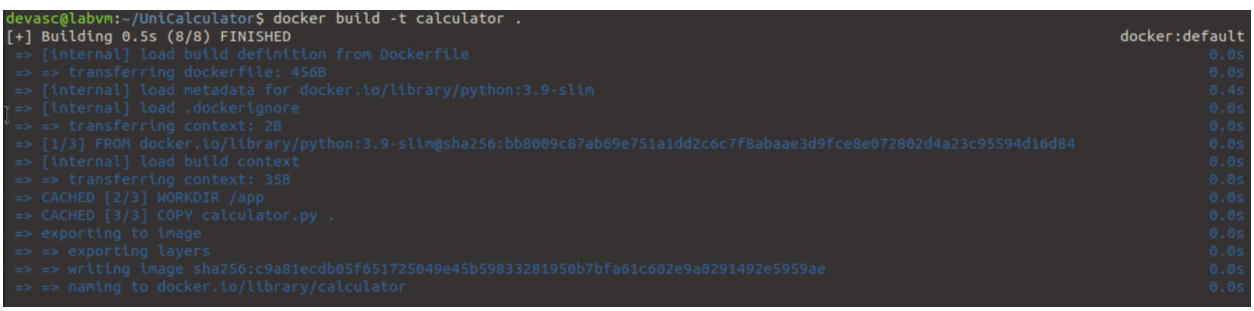


Docker Containerisation

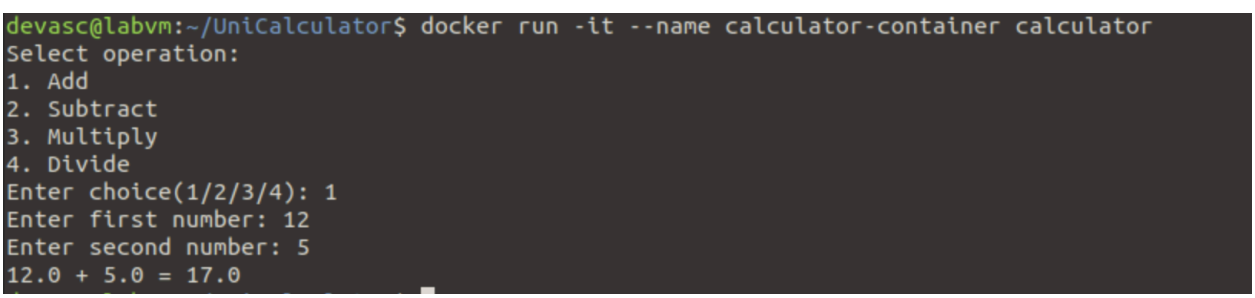
Firstly to I need to create the docker file to containerize calculator.py calling it Dockerfile then writing the code:



Then I cloned my git repo and built the application using the dockerfile:



To then test that I could access the dockerfile I used this command:



I also created a script that automatically clones the repo and sets up docker. The next runs then pulls the latest changes and restarts the docker, this makes sure that there is only one instance of the docker container running.

Welcome | Dockerfile | $ deploy_calculator.sh

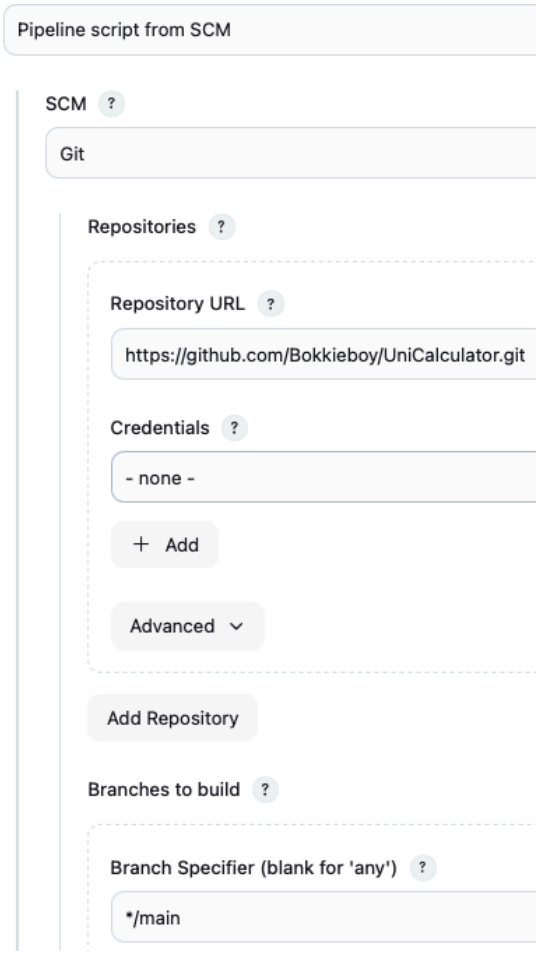
home > devasc > UniCalculator > $ deploy_calculator.sh

```bash
#!/bin/bash

# Define variables
REPO_URL="https://github.com/Bokkieboy/UniCalculator.git"
REPO_NAME="UniCalculator"
IMAGE_NAME="calculator"
CONTAINER_NAME="calculator_container"
PORT=5050

# Clone the repository if it doesn't exist
if [ ! -d "$REPO_NAME" ]; then
    echo "Cloning repository..."
    git clone "$REPO_URL"
else
    echo "Repository already exists. Pulling latest changes..."
    cd "$REPO_NAME" || exit
    git pull
    cd ..
fi

# Navigate to repository
cd "$REPO_NAME" || exit

# Build the Docker image
echo "Building Docker image..."
docker build -t "$IMAGE_NAME" .

# Stop and remove any running container with the same name
if [ "$(docker ps -q -f name=$CONTAINER_NAME)" ]; then
    echo "Stopping existing container..."
    docker stop "$CONTAINER_NAME"
    docker rm "$CONTAINER_NAME"
fi

# Run the container
echo "Running the container..."
docker run -d --name "$CONTAINER_NAME" -p "$PORT":5050 "$IMAGE_NAME"

# Verify if the container is running
if [ "$(docker ps -q -f name=$CONTAINER_NAME)" ]; then
    echo "Container is running successfully on port $PORT."
else
    echo "Error: Container failed to start."
fi
```

```
devasc@labvm:~/UniCalculator$ ./deploy_calculator.sh
Repository already exists. Pulling latest changes...
Already up to date.
Building Docker image...
[+] Building 0.0s (1/1) FINISHED                          docker:default
 => [internal] load build definition from Dockerfile                0.0s
 => => transferring dockerfile: 2B                                  0.0s
ERROR: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
Running the container...
646931e30c55f81dc0256ce3120074aceb834a23e9084facb753999b79a953f0
Container is running successfully on port 5050.
```

Pipeline Implementation

First I created a Pipeline on Jenkins I called it Github-Pipeline, then selected pipeline script from SCM, chose Git as my SCM, added my repo URL and specified the branch to */main



I then created a Jenkinsfile which pulls the code from Github, then builds the application inside a docker container and makes it accessible locally:

```
pipeline {
    agent any  - Allows it to run on any available Jenkins agent

    environment {
        DOCKER_IMAGE = 'simple-calculator'
        CONTAINER_NAME = 'calculator-app'
    }

    stages {
        stage('Checkout Code') {
            steps {
                git branch: 'main', url: 'https://github.com/Bokkieboy/UniCalculator.git'
            }
```

```
        }

        stage('Build Application') {
            steps {
                script {
                    sh 'docker build -t ${DOCKER_IMAGE} .'
                }
            }
        }

        stage('Test Application') {
            steps {
                script {
                    sh 'python3 -m unittest discover -s tests'
                }
            }
        }

        stage('Deploy Application') {
            steps {
                script {
                    sh 'docker run -d -p 5050:5050 --name ${CONTAINER_NAME} ${DOCKER_IMAGE}'
                }
            }
        }
    }

    post {
        success {
            echo 'Pipeline executed successfully!'
        }
        failure {
            echo 'Pipeline execution failed!'
        }
    }
}
```

```
e > #12
[Pipeline] sh
+ python3 -m unittest discover -s tests
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy Application)
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ docker run -d -p 5050:5050 --name calculator-app simple-calculator
9b742b5852e5db18cb15fc54edc25d02dd938cf3432282aadd8f96d3731698b1
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline executed successfully!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Then I added the test _calculator.py script I made below in Testing and Debugging. I put it in a folder called tests with **__init__.py** this tells python that all the files inside of the tests folder are packages allowing the test files to be imported properly. Then I built the pipeline and it succeeded:

Testing and Debugging

To first make sure the script runs I will create a python script which will test all the functions of the calculator script:

```
import unittest
from calculator import add, subtract, multiply, divide

class TestCalculator(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(5, 4), 9)
```
This will add 5 and 4 together and check to see that the output is 9

```
    def test_subtract(self):
        self.assertEqual(subtract(5, 3), 2)
```
This will subtract 5 and 3 together and check to see that the output is 2

```
    def test_multiply(self):
        self.assertEqual(multiply(2, 3), 6)
```
This will multiply 2 and 3 together and check to see that the output is 6

```
    def test_divide(self):
        self.assertEqual(divide(6, 2), 3)
```
This will divide 6 and 2 together and check to see that the output is 3

```
if __name__ == '__main__':
    unittest.main()
```

When I run this script I get the output:

```
devasc@labvm:~/UniCalculator$ python3 test_calculator.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```

Now I will add in self.assertEqual(add(1, 1), 7) This will add 1 and 1 together and expect 7 back but when it comes back as 2 there should tell me its wrong like this:

```
devasc@labvm:~/UniCalculator$ python3 test_calculator.py
F...
======================================================================
FAIL: test_add (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "test_calculator.py", line 7, in test_add
    self.assertEqual(add(1, 1), 7)
AssertionError: 2 != 7

----------------------------------------------------------------------
Ran 4 tests in 0.001s

FAILED (failures=1)
```