Calci App — Assignment Runbook & Steps I Followed

I built a simple Python calculator (add / subtract / multiply for positive integers), wrote pytest unit tests, wrapped everything in a Docker image, created a Jenkins pipeline to run tests and build, and deployed the function to AWS Lambda using AWS SAM — with the pipeline triggered by Git pushes.

Project structure (final)

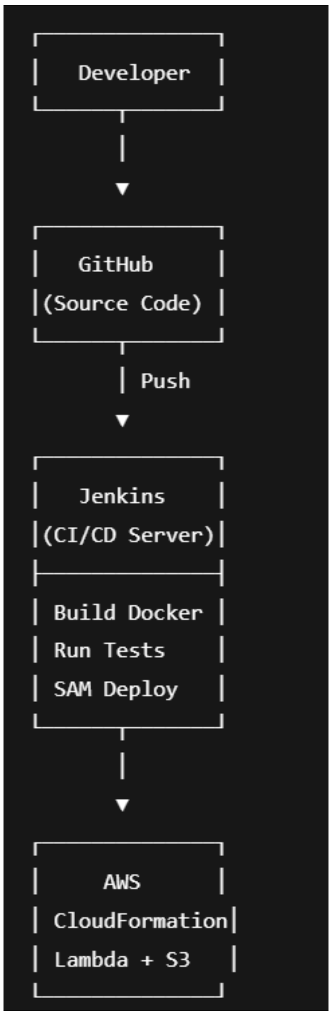
This is the repository layout I used (project root):

```
calci-app/
├── calci.py                 # Lambda entry + calculator functions
├── test_calci.py           # pytest unit tests
├── requirements.txt        # pytest (and other deps if needed)
├── Dockerfile              # docker image used by Jenkins for tests
├── sam-template.yaml       # SAM template (Lambda infra)
├── Jenkinsfile             # Jenkins pipeline (CI + CD)
├── event.json              # test event for local invoke (optional)
└── .gitignore
```

`.gitignore` contents I used:

```
.aws-sam/
samconfig.toml
*.pyc
__pycache__/
.idea/
.vscode/
```

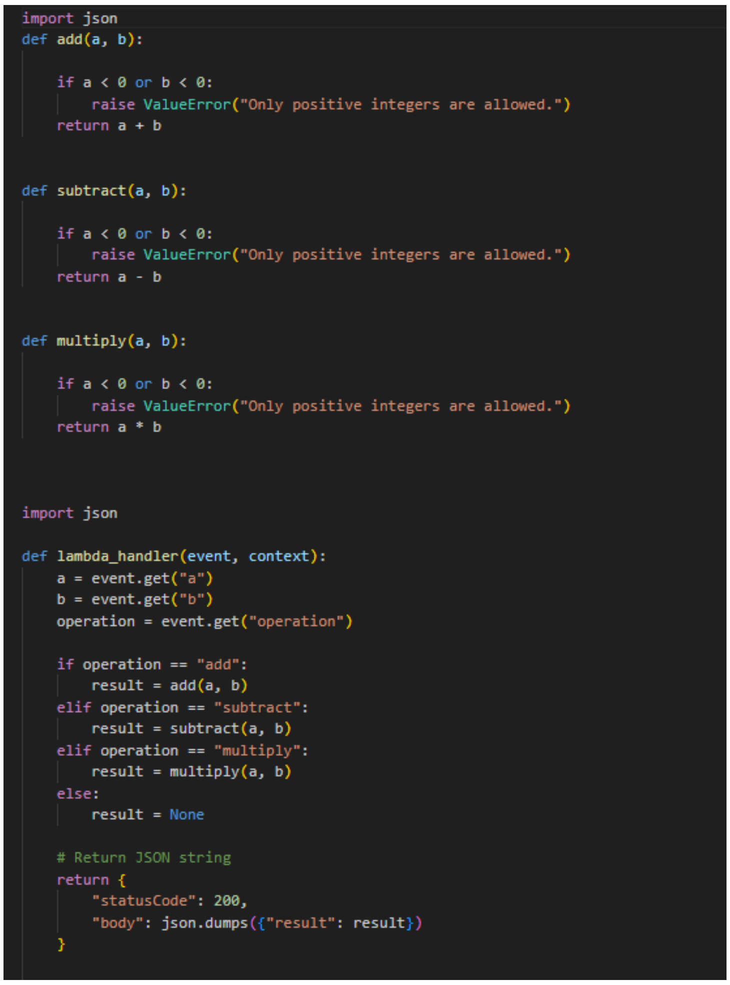
Flow Of Assignment :-



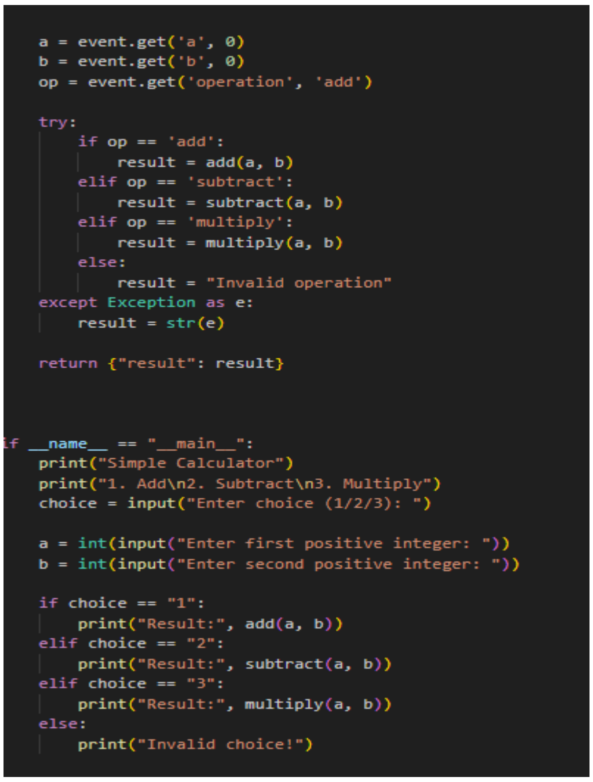
```
┌──────────────┐
│   Developer  │
└──────┬───────┘
       │
       ▼
┌──────────────┐
│   GitHub     │
│(Source Code) │
└──────┬───────┘
       │ Push
       ▼
┌──────────────┐
│   Jenkins    │
│(CI/CD Server)│
├──────────────┤
│ Build Docker │
│ Run Tests    │
│ SAM Deploy   │
└──────┬───────┘
       │
       ▼
┌──────────────┐
│     AWS      │
│ CloudFormation│
│ Lambda + S3   │
└──────────────┘
```

Files I created / key contents

1. calci.py

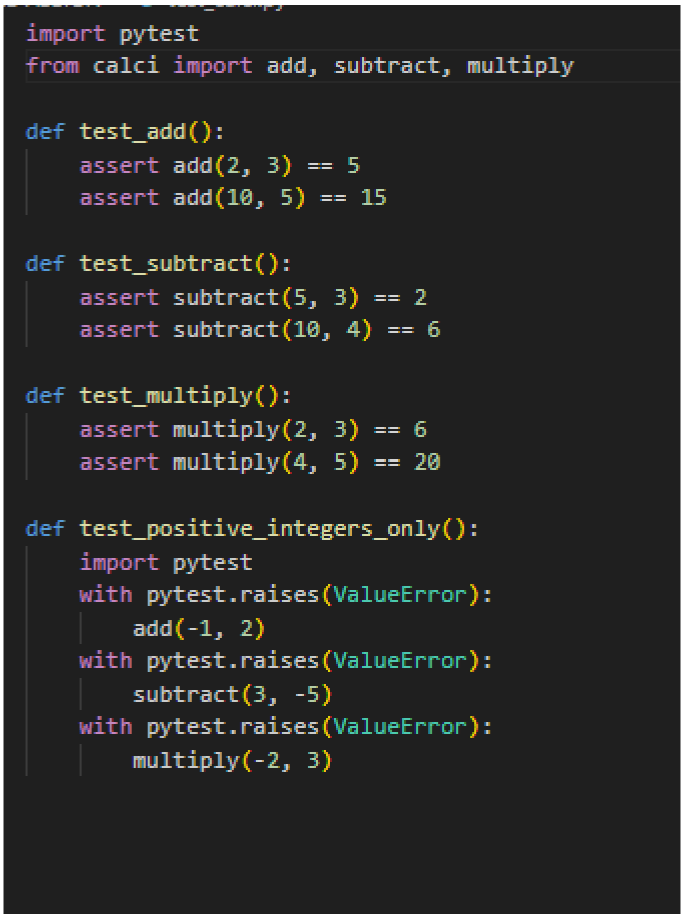


```python
import json
def add(a, b):

    if a < 0 or b < 0:
        raise ValueError("Only positive integers are allowed.")
    return a + b


def subtract(a, b):

    if a < 0 or b < 0:
        raise ValueError("Only positive integers are allowed.")
    return a - b


def multiply(a, b):

    if a < 0 or b < 0:
        raise ValueError("Only positive integers are allowed.")
    return a * b



import json

def lambda_handler(event, context):
    a = event.get("a")
    b = event.get("b")
    operation = event.get("operation")

    if operation == "add":
        result = add(a, b)
    elif operation == "subtract":
        result = subtract(a, b)
    elif operation == "multiply":
        result = multiply(a, b)
    else:
        result = None

    # Return JSON string
    return {
        "statusCode": 200,
        "body": json.dumps({"result": result})
    }
```



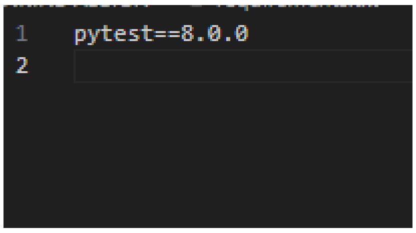
```python
    a = event.get('a', 0)
    b = event.get('b', 0)
    op = event.get('operation', 'add')

    try:
        if op == 'add':
            result = add(a, b)
        elif op == 'subtract':
            result = subtract(a, b)
        elif op == 'multiply':
            result = multiply(a, b)
        else:
            result = "Invalid operation"
    except Exception as e:
        result = str(e)

    return {"result": result}


if __name__ == "__main__":
    print("Simple Calculator")
    print("1. Add\n2. Subtract\n3. Multiply")
    choice = input("Enter choice (1/2/3): ")

    a = int(input("Enter first positive integer: "))
    b = int(input("Enter second positive integer: "))

    if choice == "1":
        print("Result:", add(a, b))
    elif choice == "2":
        print("Result:", subtract(a, b))
    elif choice == "3":
        print("Result:", multiply(a, b))
    else:
        print("Invalid choice!")
```

2. **test_calci.py**



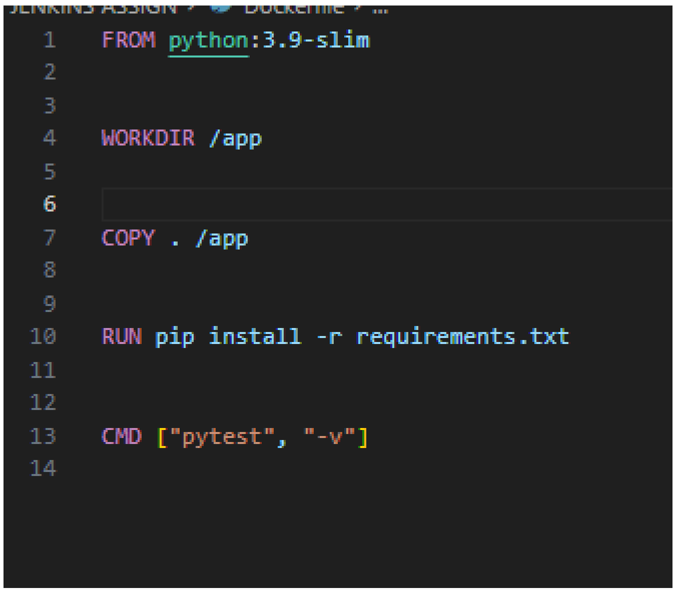
```python
import pytest
from calci import add, subtract, multiply

def test_add():
    assert add(2, 3) == 5
    assert add(10, 5) == 15

def test_subtract():
    assert subtract(5, 3) == 2
    assert subtract(10, 4) == 6

def test_multiply():
    assert multiply(2, 3) == 6
    assert multiply(4, 5) == 20

def test_positive_integers_only():
    import pytest
    with pytest.raises(ValueError):
        add(-1, 2)
    with pytest.raises(ValueError):
        subtract(3, -5)
    with pytest.raises(ValueError):
        multiply(-2, 3)
```

3. **requirements.txt**



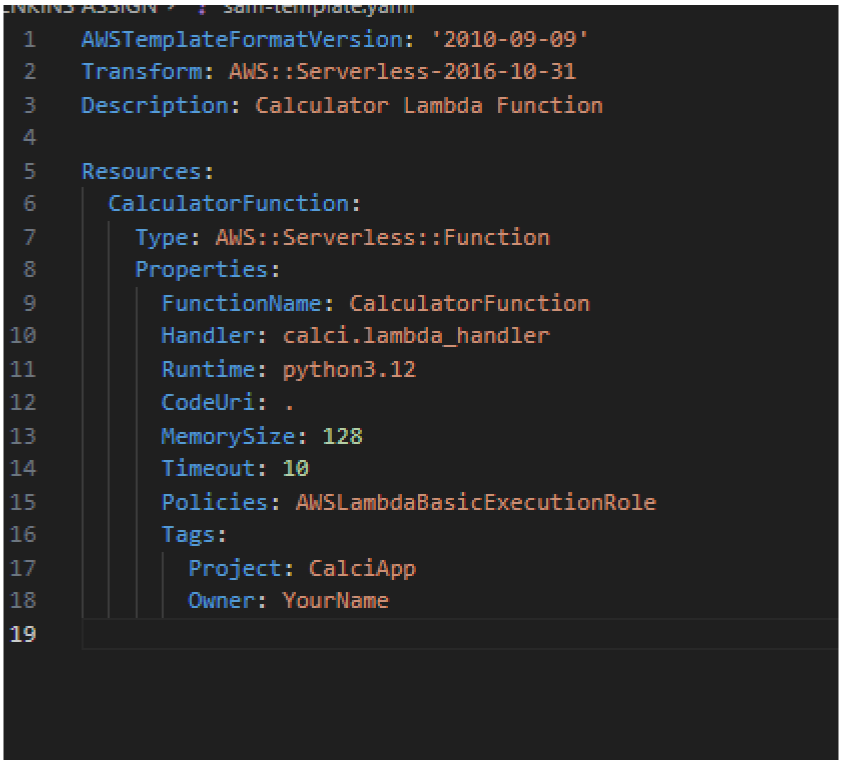
```
pytest==8.0.0

```

4. **Dockerfile**



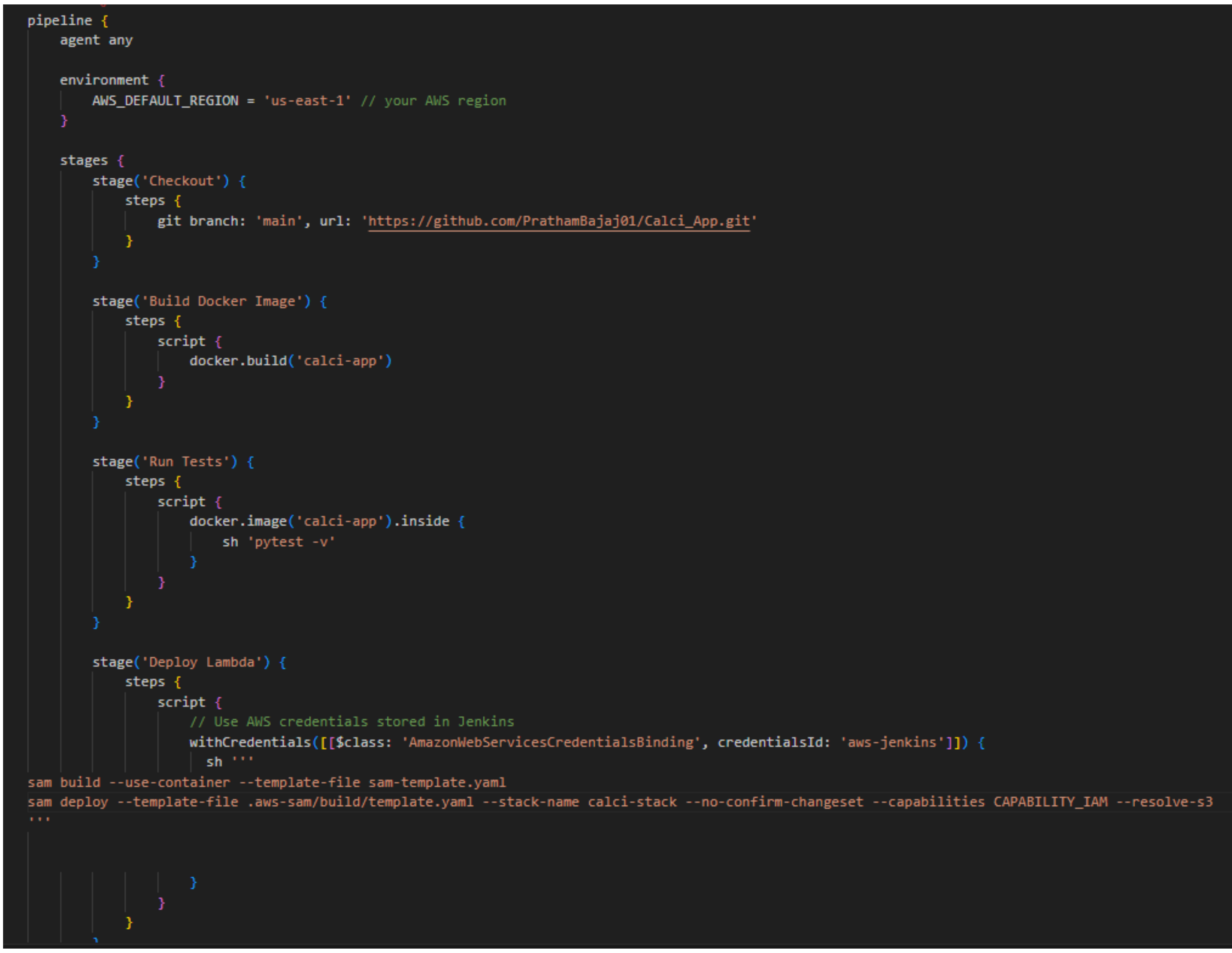
```dockerfile
FROM python:3.9-slim


WORKDIR /app


COPY . /app


RUN pip install -r requirements.txt


CMD ["pytest", "-v"]

```

5. sam-template.yaml

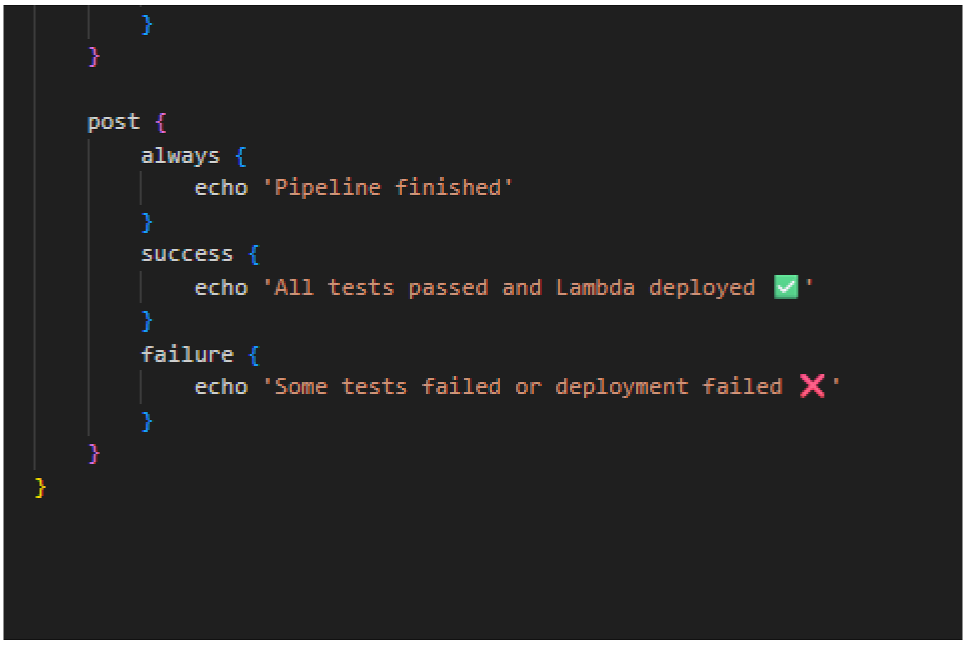


```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Calculator Lambda Function

Resources:
  CalculatorFunction:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: CalculatorFunction
      Handler: calci.lambda_handler
      Runtime: python3.12
      CodeUri: .
      MemorySize: 128
      Timeout: 10
      Policies: AWSLambdaBasicExecutionRole
      Tags:
        Project: CalciApp
        Owner: YourName

```

6. Jenkinsfile



```groovy
pipeline {
    agent any

    environment {
        AWS_DEFAULT_REGION = 'us-east-1' // your AWS region
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/PrathamBajaj01/Calci_App.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build('calci-app')
                }
            }
        }

        stage('Run Tests') {
            steps {
                script {
                    docker.image('calci-app').inside {
                        sh 'pytest -v'
                    }
                }
            }
        }

        stage('Deploy Lambda') {
            steps {
                script {
                    // Use AWS credentials stored in Jenkins
                    withCredentials([[$class: 'AmazonWebServicesCredentialsBinding', credentialsId: 'aws-jenkins']]) {
                        sh '''
sam build --use-container --template-file sam-template.yaml
sam deploy --template-file .aws-sam/build/template.yaml --stack-name calci-stack --no-confirm-changeset --capabilities CAPABILITY_IAM --resolve-s3
'''

                    }
                }
            }
```



```
        }
    }

    post {
        always {
            echo 'Pipeline finished'
        }
        success {
            echo 'All tests passed and Lambda deployed ✅'
        }
        failure {
            echo 'Some tests failed or deployment failed ❌'
        }
    }
}
```

Exact commands I ran locally (WSL / terminal)

1. clone / create repo and files.
2. tests locally:

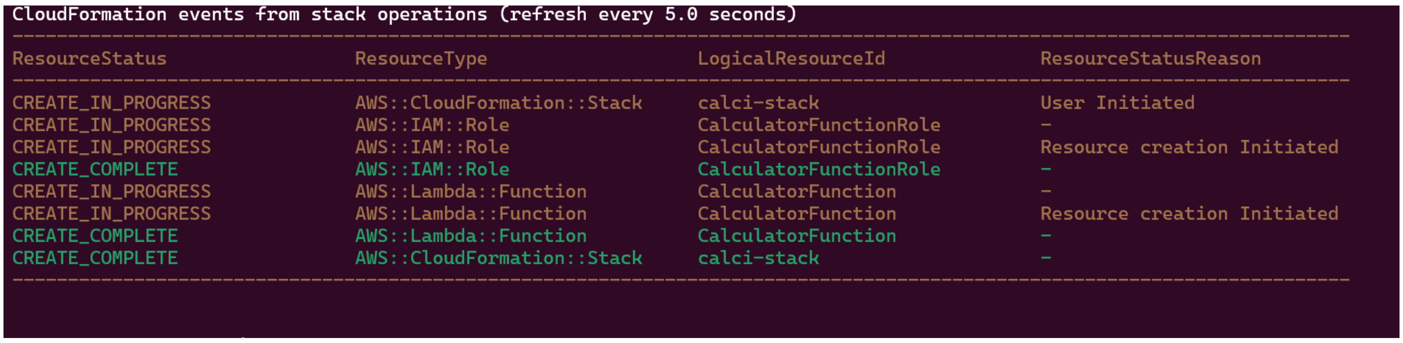
```
# build docker image for local test
docker build -t calci-app .
# run tests inside container
docker run --rm calci-app
# or run pytest locally
pytest -v
```

3. SAM local build & invoke:

```
# build (I used container because my WSL Python version differed from Lambda runtime)
sam build --use-container --template-file sam-template.yaml

# create an event.json (single-line or file with no linebreaks when invoking via aws
CLI):
echo '{"a":10,"b":3,"operation":"multiply"}' > event.json

# local invoke
sam local invoke CalculatorFunction --event event.json
```

4. deploy from local (guided once to save samconfig):

```
sam deploy --guided
# answered stack name: calci-stack, region: us-east-1, allow IAM role creation: y, save
config: y
```



```
CloudFormation events from stack operations (refresh every 5.0 seconds)
-------------------------------------------------------------------------------------------------------------------------
ResourceStatus                ResourceType                  LogicalResourceId             ResourceStatusReason
-------------------------------------------------------------------------------------------------------------------------
CREATE_IN_PROGRESS            AWS::CloudFormation::Stack    calci-stack                   User Initiated
CREATE_IN_PROGRESS            AWS::IAM::Role                CalculatorFunctionRole        -
CREATE_IN_PROGRESS            AWS::IAM::Role                CalculatorFunctionRole        Resource creation Initiated
CREATE_COMPLETE               AWS::IAM::Role                CalculatorFunctionRole        -
CREATE_IN_PROGRESS            AWS::Lambda::Function         CalculatorFunction            -
CREATE_IN_PROGRESS            AWS::Lambda::Function         CalculatorFunction            Resource creation Initiated
CREATE_COMPLETE               AWS::Lambda::Function         CalculatorFunction            -
CREATE_COMPLETE               AWS::CloudFormation::Stack    calci-stack                   -
-------------------------------------------------------------------------------------------------------------------------
```

5. test deployed function (AWS CLI must be configured: aws configure):

```
aws lambda invoke --function-name CalculatorFunction --payload file://event.json
response.json
cat response.json
```

Jenkins setup I did

1. Installed Jenkins and Docker on the same machine (Jenkins node as a Docker host).
2. Installed **AWS Credentials Plugin** in Jenkins.
3. Created an IAM user for Jenkins with programmatic access (Access Key ID + Secret).
   - For simplicity during the lab I temporarily attached `AdministratorAccess` to avoid SAM-managed stack / servicecatalog permission issues.
   - After demo I planned to lock permissions down to the minimal set.
4. Added those AWS keys to Jenkins Credentials → **Kind: AWS Credentials**, ID = `aws-jenkins`.
5. Created a new Pipeline job in Jenkins, configured it to use my GitHub repo and the `Jenkinsfile` in the repo.
6. Enabled GitHub webhook (or Poll SCM as fallback). For webhook testing I used **ngrok** to expose local Jenkins: `ngrok http 8080` and used the forwarded HTTPS URL as the webhook URL.
7. Pushed code to GitHub and observed Jenkins automatically run the pipeline.

How I demonstrated the full flow (steps for demo)

1. Make a small change in `calci.py` (e.g., add a print or tweak behavior).
2. Commit & push to GitHub:

```
git add calci.py
git commit -m "Small change - demo"
git push origin main
```

3. GitHub webhook triggered Jenkins pipeline:
   - Checkout latest code
   - Build Docker image
   - Run pytest (container)
   - Build and deploy via SAM (`sam build --use-container` then `sam deploy ... --resolve-s3`)
4. Open AWS Console → Lambda → verify `CalculatorFunction` updated.
5. Use the AWS console test or `aws lambda invoke` to verify updated behavior.

Difficulties I faced (and how I fixed them)

I list the main, practical issues I encountered and the exact fix I used. I include the ones I encountered in this project.

1. **Wrong name / mismatch: `calci.py` vs `calculator.py`**
   - Problem: tests initially imported `calculator` but my file was `calci.py`.
   - Fix: updated `test_calci.py` to `from calci import ...` and consistently used `calci.py`.
2. **SAM build could not find template**
   - Problem: My SAM template filename was `sam-template.yaml`, but SAM defaults to `template.yaml`. Jenkins/SAM reported `Template file not found`.
   - Fix: either rename `sam-template.yaml` → `template.yaml`, or run `sam build --template-file sam-template.yaml`. In Jenkins I used explicit `--template-file` in build to be safe.
3. **Python runtime mismatch — SAM needed exact runtime**
   - Problem: SAM complained that `python3.9` (or other runtime) was not available locally.
   - Fix: I used `sam build --use-container` so the build runs inside the official Lambda build container with the correct Python runtime. This avoided installing specific Python versions on WSL.
4. **Managed SAM stack (`aws-sam-cli-managed-default`) in `ROLLBACK_COMPLETE`**
   - Problem: previous failed deploys left SAM-managed stack broken; SAM refused to proceed.
   - Fix: I deleted the broken stack in CloudFormation (AWS Console → CloudFormation → select `aws-sam-cli-managed-default` → Delete). After deletion and ensuring IAM permissions, SAM recreated the managed stack successfully.
5. **IAM permission issues (e.g., `servicecatalog:ListApplications` denied)**
   - Problem: SAM attempted calls that my IAM user lacked permissions for; deployments failed/rolled back.
   - Fix: For the assignment I temporarily attached `AdministratorAccess` while debugging. After confirming the pipeline worked, I planned to reduce permissions to the minimal set required:
     - `AWSLambda_FullAccess`, `CloudFormationFullAccess`, `CloudWatchLogsFullAccess`, `IAMFullAccess`, and S3 permissions (`s3:PutObject`, `s3:CreateBucket` if using an explicit bucket). Alternatively, `--resolve-s3` lets SAM create a bucket but the IAM user must have S3 create/put rights.
6. **`Invalid base64` when invoking Lambda with AWS CLI**
   - Problem: When invoking with a file that contained multi-line JSON, AWS CLI returned `Invalid base64`.

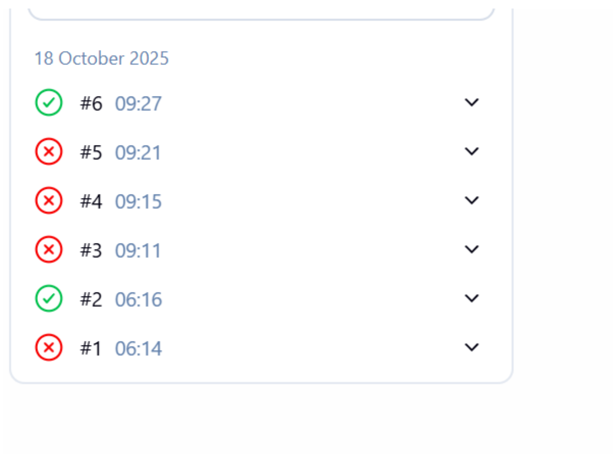
   - Fix: Send the payload as a single-line JSON in `event.json` (or use `--payload "{\"a\":10,...}"` with escaped quotes), and ensure the Lambda returns a JSON-serializable body (I returned `{"statusCode":200,"body": json.dumps({...})}`).

7. **SAM deploy failed: "S3 Bucket not specified"**
   - Problem: SAM needed an S3 bucket for uploading the packaged Lambda code.
   - Fix: I added `--resolve-s3` to `sam deploy` so SAM creates a managed bucket, or I could create a named S3 bucket and pass `--s3-bucket my-bucket`.
8. **Webhook issues: GitHub cannot reach `localhost`**
   - Problem: GitHub failed to reach my local Jenkins at `http://localhost:8080`.
   - Fix: I used `ngrok http 8080` to expose a public HTTPS URL and set that in GitHub webhook (Payload URL: `https://<ngrok-id>.ngrok.io/github-webhook/`). Remember the ngrok URL changes each run unless you have a paid ngrok subdomain.
9. **Template path mismatch after `sam build`**
   - Problem: after build, SAM places the built template at `.aws-sam/build/template.yaml`, but I attempted to deploy `.aws-sam/build/sam-template.yaml`.
   - Fix: deploy using `.aws-sam/build/template.yaml` or use consistent naming. I adjusted Jenkinsfile to use `.aws-sam/build/template.yaml`.
10. **Workspace path / spaces issues on Windows/WSL**
    - Problem: my repo path had spaces (`DG ASSIGNMENTS`), causing command/escape troubles.
    - Fix: properly quoted paths or moved project to simpler path (recommended) like `C:\projects\calci-app` or `~/calculator-app` in WSL.

How I verified success (checklist)

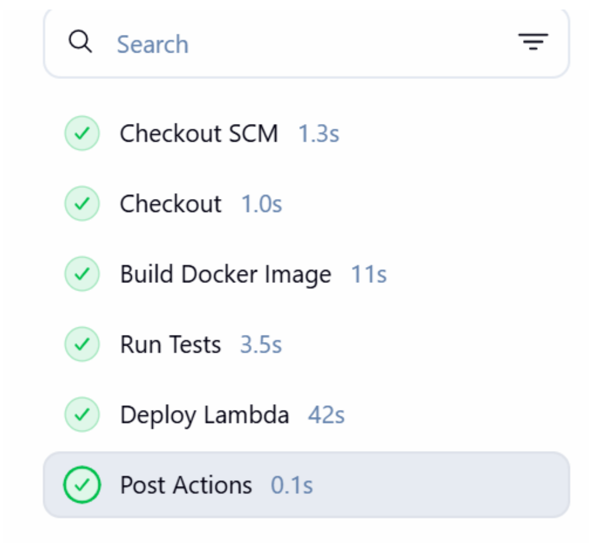
- ☑ `pytest` locally passes (`pytest -v`).
- ☑ `sam local invoke` returns expected result with `event.json`.
- ☑ Jenkins pipeline finishes with “All tests passed and Lambda deployed”.
- ☑ AWS Lambda console shows `CalculatorFunction` in `us-east-1`.
- ☑ `aws lambda invoke --function-name CalculatorFunction --payload file://event.json response.json` returns `{"statusCode":200,"body":"{\"result\":<value>}"}`
    - I parsed response.body to get the result.

Final Results :-

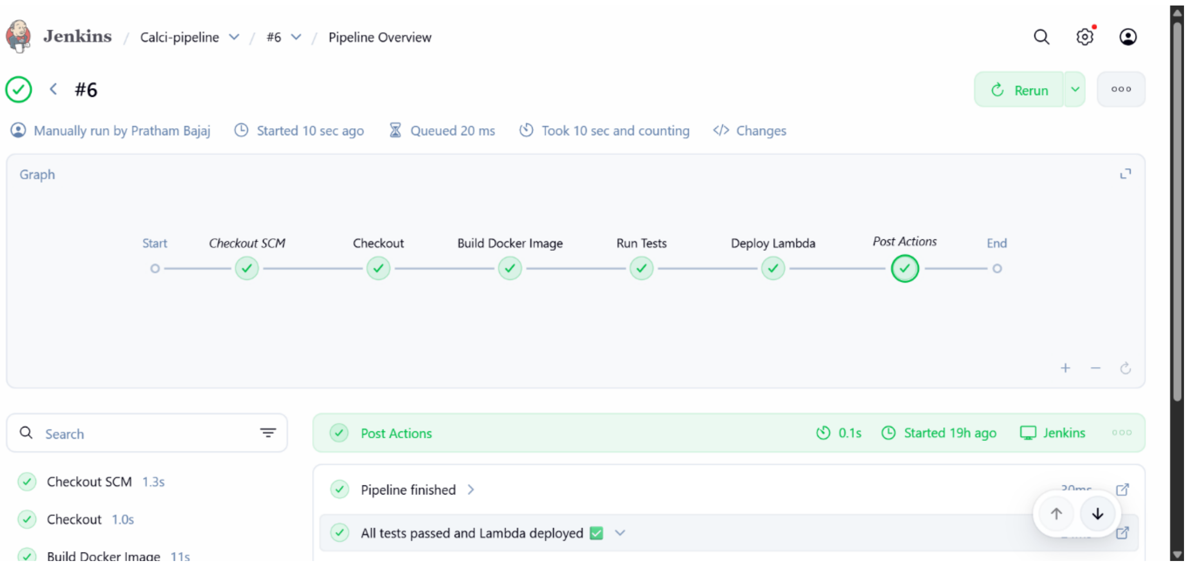


18 October 2025

✓	#6	09:27
✗	#5	09:21
✗	#4	09:15
✗	#3	09:11
✓	#2	06:16
✗	#1	06:14



✓	Checkout SCM	1.3s
✓	Checkout	1.0s
✓	Build Docker Image	11s
✓	Run Tests	3.5s
✓	Deploy Lambda	42s
✓	Post Actions	0.1s



Jenkins / Calci-pipeline / #6 / Pipeline Overview

#6

Rerun

Manually run by Pratham Bajaj Started 10 sec ago Queued 20 ms Took 10 sec and counting Changes

Graph

Search

Checkout SCM 1.3s

Checkout 1.0s

Build Docker Image 11s

Post Actions 0.1s Started 19h ago Jenkins

Pipeline finished 30ms

All tests passed and Lambda deployed ✓