Calibration Pipeline CI/CD Plan

This project automates the build and deployment process for Python applications and SQLite databases on AWS EC2 for the Calibrate Pipeline project at RFO.

# 1. Version Control with Git

* **Git**: This project uses Git for version control of the Python code. Database structure and management is all done through Python. The imagelib repository has been set up in the vmoa GitHub environment.
* **Branching**: Development work will be done on the calibratepipeline branch, test deployments are done by pushing code to the calpipelinetest branch, and production deployments by pushing to the master branch of the imagelib repository.

# 2. Build Automation and CI/CD Pipeline

* This project uses GitHub Actions to set up a pipeline that triggers automatically when code is pushed to the relevant branch.
* Basic build processes that are implemented in GitHub Actions are:
  + Pipenv – to set up, manage and install python dependencies within a virtual python environment.
  + Pytest – to run unit tests and regression tests on a set of python code and to halt the build if tests fail.
  + Pylint – to run syntax checks on python code and to halt the build if the tests fail.
* GitHub Actions are used to integrate deployment to AWS EC2 for both test and production environments using AWS CodeDeploy.
* All GitHub Actions run on instances hosted by GitHub for the duration of the workflow execution.

# 3. Monitoring with Prometheus/Grafana

* This project uses an instance of Prometheus and Grafana running on the rfopower Raspberry Pi hosted in the observatory.
* The AWS instance hosts a node\_exporter service that send data on the EC2 instance itself and on the processes that comprise the imagelib system.

# 4. Setting up a Development Environment

## Mac OS

1. Install pipenv  
   > brew install python  
   > brew install pipx  
   > pipx install pipenv  
   > pipx ensurepath  
   >source ~/.bashrc
2. Create a project directory  
   > git clone imagelib

# 4. GitHub Actions Examples

Here are examples of how you can use GitHub Actions for various tasks in your Python project, from running tests to deploying to AWS EC2.

## Basic GitHub Actions Workflow for Python Project

This workflow installs dependencies, runs linting, and runs tests.

# .github/workflows/python-app.yml

name: Python Application

on:

push:

branches:

- master

pull\_request:

branches:

- master

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python 3.9

uses: actions/setup-python@v4

with:

python-version: '3.9'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Lint with pylint

run: |

pip install pylint

# Stop the build if there are Python syntax errors or undefined names

pylint . --count --select=E9,F63,F7,F82 --show-source --statistics

# Exit with errors if any linting issues are found

pylint . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

- name: Run tests with pytest

run: |

pip install pytest

pytest

## GitHub Actions for Deploying to AWS EC2

# .github/workflows/aws-codedeploy.yml

name: Deploy via AWS CodeDeploy

on:

push:

branches:

- master

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python 3.9

uses: actions/setup-python@v4

with:

python-version: '3.9'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-west-2

- name: Deploy to AWS CodeDeploy

run: |

aws deploy create-deployment \

--application-name MyApp \

--deployment-group-name MyApp-DG \

--s3-location bucket=my-bucket,key=my-app.zip,bundleType=zip \

--deployment-config-name CodeDeployDefault.OneAtATime

### Explanation:

* **AWS CLI**: The workflow uses the aws-actions/configure-aws-credentials action to authenticate with AWS and then uses AWS CodeDeploy for deployment.
* **S3 and CodeDeploy**: The deployment package is stored in S3, and AWS CodeDeploy handles the deployment to EC2 or other supported services.
* **Secrets**: Store AWS access keys in GitHub secrets (AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY).

Here are examples of how to use pipenv and poetry for managing Python dependencies and pytest for testing.

**1. Using pipenv**

pipenv is a tool that manages dependencies in a virtual environment while also generating a Pipfile and Pipfile.lock to track your project dependencies.

**Install pipenv**

bash

Copy code

pip install pipenv

**Creating a New Project with pipenv**

1. Navigate to your project folder.
2. Run pipenv to create a virtual environment and install dependencies.

bash

Copy code

pipenv install requests # This creates a Pipfile and installs 'requests' package

This will create a Pipfile in your project directory.

**Pipfile example:**

toml

Copy code

[[source]]

name = "pypi"

url = "https://pypi.org/simple"

verify\_ssl = true

[packages]

requests = "\*"

[dev-packages]

pytest = "\*"

[requires]

python\_version = "3.9"

1. To activate the virtual environment, run:

bash

Copy code

pipenv shell

1. To install development dependencies like pytest:

bash

Copy code

pipenv install pytest --dev

1. To run your tests with pipenv:

bash

Copy code

pipenv run pytest

**3. Writing Unit Tests with pytest**

Here’s a simple example of writing unit tests using pytest.

**1. Install pytest (via pipenv or poetry):**

bash

Copy code

pipenv install pytest --dev

# or

poetry add --dev pytest

**2. Create a sample Python file (calculator.py):**

python

Copy code

# calculator.py

def add(a, b):

return a + b

def subtract(a, b):

return a - b

**3. Create a test file (test\_calculator.py) to test the calculator.py file:**

python

Copy code

# test\_calculator.py

from calculator import add, subtract

def test\_add():

assert add(2, 3) == 5

assert add(-1, 1) == 0

def test\_subtract():

assert subtract(5, 3) == 2

assert subtract(0, 3) == -3

**4. Run pytest:**

* Using pipenv:

bash

Copy code

pipenv run pytest

* Using poetry:

bash

Copy code

poetry run pytest

**4. Example of pytest Output**

When you run pytest, the test results will look like this:

bash

Copy code

=========================== test session starts ============================

platform linux -- Python 3.9.0, pytest-6.2.5, py-1.10.0, pluggy-0.13.1

rootdir: /path/to/project

collected 2 items

test\_calculator.py .. [100%]

============================ 2 passed in 0.03s =============================

**Summary:**

* **pipenv**: Creates Pipfile and Pipfile.lock for dependency management.
* **poetry**: Manages dependencies using pyproject.toml and simplifies packaging.
* **pytest**: A powerful framework to write and run tests.

Here’s a complete example of how to use **AWS CodeDeploy** to deploy your Python application on an EC2 instance. We’ll go through creating the necessary configuration files, setting up CodeDeploy, and automating deployments via GitHub Actions.

**1. Set Up AWS CodeDeploy for EC2**

**Step 1: Create an IAM Role for CodeDeploy**

1. Go to the **IAM Console**.
2. Create a new role for **EC2** and attach the following policies:
   * AmazonEC2RoleforAWSCodeDeploy
   * AmazonS3FullAccess (if your code will be in S3)

**Step 2: Create an EC2 Instance**

1. Launch an EC2 instance (Amazon Linux or Ubuntu) with the IAM role you created.
2. SSH into your EC2 instance and install the CodeDeploy agent.

For **Amazon Linux**:

bash

Copy code

sudo yum update

sudo yum install ruby

sudo yum install wget

cd /home/ec2-user

wget https://aws-codedeploy-us-west-2.s3.us-west-2.amazonaws.com/latest/install

chmod +x ./install

sudo ./install auto

For **Ubuntu**:

bash

Copy code

sudo apt-get update

sudo apt-get install ruby wget

cd /home/ubuntu

wget https://aws-codedeploy-us-west-2.s3.us-west-2.amazonaws.com/latest/install

chmod +x ./install

sudo ./install auto

Verify the CodeDeploy agent is running:

bash

Copy code

sudo service codedeploy-agent status

**Step 3: Create an Application in AWS CodeDeploy**

1. Go to the **AWS CodeDeploy Console**.
2. Create a new application.
   * Application Name: MyApp
   * Compute platform: **EC2/On-Premises**
3. Create a new **Deployment Group**:
   * Deployment Group Name: MyApp-DG
   * Service Role: Choose the IAM role you created for CodeDeploy.
   * Environment Configuration: Choose the EC2 instances where the app will be deployed (using tags or instance IDs).

**2. Create AppSpec File for CodeDeploy**

AWS CodeDeploy uses an appspec.yml file to define how to deploy your application.

Create the following appspec.yml file in your project root directory:

yaml

Copy code

version: 0.0

os: linux

files:

- source: /

destination: /home/ec2-user/myapp

hooks:

AfterInstall:

- location: scripts/install\_dependencies.sh

timeout: 300

runas: ec2-user

ApplicationStart:

- location: scripts/start\_server.sh

timeout: 300

runas: ec2-user

**Explanation:**

* **Files**: Specifies where to copy the source files (in this case, to /home/ec2-user/myapp on the EC2 instance).
* **Hooks**:
  + **AfterInstall**: Runs a script to install dependencies (e.g., Python packages).
  + **ApplicationStart**: Starts the application (or service) after deployment.

**3. Create Deployment Scripts**

You need to create the scripts referenced in the appspec.yml.

**1. scripts/install\_dependencies.sh:**

This script installs Python dependencies and sets up the environment.

bash

Copy code

#!/bin/bash

# Navigate to the application directory

cd /home/ec2-user/myapp

# Install Python dependencies

pip3 install -r requirements.txt

**2. scripts/start\_server.sh:**

This script starts the application (or a web server).

bash

Copy code

#!/bin/bash

# Navigate to the application directory

cd /home/ec2-user/myapp

# Start the application (e.g., Flask or Django)

nohup python3 app.py > app.log 2>&1 &

**4. Automate Deployment with AWS CodeDeploy**

You can automate the process of deploying your app to EC2 using GitHub Actions. When you push code to GitHub, GitHub Actions will package your app, upload it to S3, and trigger a CodeDeploy deployment.

Here’s an example of a GitHub Actions workflow for this:

yaml

Copy code

# .github/workflows/deploy-to-codedeploy.yml

name: Deploy to AWS CodeDeploy

on:

push:

branches:

- main

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Zip the project files

run: zip -r myapp.zip .

- name: Configure AWS credentials

uses: aws-actions/configure-aws-credentials@v1

with:

aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

aws-region: us-west-2

- name: Upload to S3

run: |

aws s3 cp myapp.zip s3://myapp-bucket/myapp.zip

- name: Deploy to CodeDeploy

run: |

aws deploy create-deployment \

--application-name MyApp \

--deployment-group-name MyApp-DG \

--s3-location bucket=myapp-bucket,key=myapp.zip,bundleType=zip \

--deployment-config-name CodeDeployDefault.OneAtATime

**Explanation:**

* **GitHub Actions Workflow**:
  + **Zip the project**: Compresses your app into a myapp.zip file.
  + **Configure AWS credentials**: Authenticates with AWS using the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY stored in GitHub secrets.
  + **Upload to S3**: Uploads the zip file to your S3 bucket.
  + **Deploy to CodeDeploy**: Triggers an AWS CodeDeploy deployment using the zip file stored in S3.

**5. Triggering a Deployment**

Once you push your code to GitHub's main branch, this GitHub Actions workflow will automatically:

* Zip your code.
* Upload it to S3.
* Trigger a deployment to the specified EC2 instances using AWS CodeDeploy.

**6. Monitoring and Rollback**

You can monitor the deployment process in the **AWS CodeDeploy Console**. AWS CodeDeploy supports automatic rollback if something goes wrong during deployment (e.g., if an error occurs in AfterInstall or ApplicationStart).

To enable rollback:

* When creating or editing the deployment group, select "Enable automatic rollback" under **Rollback configurations**.

**Summary**

1. **Set up AWS CodeDeploy and install the agent** on EC2 instances.
2. **Create an AppSpec file** (appspec.yml) to define how your app is deployed.
3. **Write scripts** to handle installing dependencies and starting your app.
4. **Use GitHub Actions** to automate the process of deploying the app, uploading it to S3, and triggering AWS CodeDeploy.
5. **Monitor** the deployment in the AWS CodeDeploy Console and set up rollback for safety.

This pipeline ensures smooth and automated deployments from GitHub to your EC2 environment via AWS CodeDeploy.