Pour utiliser GitHub Actions avec vos scripts Python et déployer des workflows d'intégration et de test automatisés pour ce projet, voici comment procéder. Je vous guide étape par étape pour configurer un workflow de base sur GitHub Actions qui va :

1. Exécuter des tests pour vérifier que votre code fonctionne comme prévu.
2. Lancer votre application avec FastAPI et Flask si nécessaire (en mode test).
3. Tester l’interaction entre les services API.

Voici la marche à suivre pour définir un workflow GitHub Actions adapté à votre projet :

**Étape 1 : Structure de votre projet sur GitHub**

Assurez-vous que votre projet contient les dossiers et fichiers suivants :

* api\_nov.py (votre script FastAPI)
* app\_nov.py (votre script Flask)
* requirements.txt (fichier listant les dépendances, que vous pouvez générer avec pip freeze > requirements.txt)
* Un dossier tests ou un fichier de tests, par exemple test\_app.py pour tester les fonctionnalités de app\_nov.py.

**Étape 2 : Créer un fichier de configuration GitHub Actions**

Dans votre dépôt GitHub, créez un dossier .github/workflows à la racine du projet. Ce dossier contiendra le fichier de configuration YAML pour GitHub Actions.

**Étape 3 : Écrire le fichier de configuration YAML pour GitHub Actions**

Créez un fichier .github/workflows/main.yml avec le contenu suivant :

yaml

Copier le code

name: CI Pipeline for FastAPI and Flask Project

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

test:

runs-on: ubuntu-latest

services:

db:

image: postgres:13

env:

POSTGRES\_DB: db\_pco

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgre

ports:

- 5432:5432

options: >-

--health-cmd "pg\_isready -U postgres"

--health-interval 10s

--health-timeout 5s

--health-retries 5

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set up Python

uses: actions/setup-python@v2

with:

python-version: '3.8'

- name: Install dependencies

run: |

python -m pip install --upgrade pip

pip install -r requirements.txt

- name: Run FastAPI server

run: |

uvicorn api\_nov:app --host 127.0.0.1 --port 8000 &

sleep 5

- name: Run Flask app server

run: |

python app\_nov.py &

sleep 5

- name: Run tests

run: |

pytest tests/test\_app.py --maxfail=1 --disable-warnings -v

**Explication du fichier YAML**

* **name** : Nom de votre workflow, ici "CI Pipeline for FastAPI and Flask Project".
* **on** : Spécifie quand le workflow doit s'exécuter (ici lors des push et pull\_request sur la branche main).
* **jobs** : Définit un travail (job) nommé test avec une instance Ubuntu.
* **services** : Démarre un service PostgreSQL pour simuler la base de données locale.
* **steps** : Liste des étapes du workflow :
  + **Checkout code** : Clone votre dépôt.
  + **Set up Python** : Installe Python 3.8.
  + **Install dependencies** : Installe les dépendances depuis requirements.txt.
  + **Run FastAPI server** : Lance api\_nov.py sur localhost:8000.
  + **Run Flask app server** : Lance app\_nov.py en arrière-plan.
  + **Run tests** : Exécute les tests avec pytest.

**Étape 4 : Configurer les tests avec Pytest**

Assurez-vous que le fichier test\_app.py (ou vos autres tests) est compatible avec pytest. Par exemple, dans test\_app.py :

python

Copier le code

import requests

def test\_api\_predict():

url = "http://127.0.0.1:8000/predict"

params = {"text\_input": "example text"}

response = requests.post(url, params=params)

assert response.status\_code == 200

assert "nombre\_segments\_algo\_1" in response.json()

Ce test fait une requête POST à votre API et vérifie le statut de la réponse.

**Étape 5 : Configurer MLflow et base de données PostgreSQL**

Pour permettre à MLflow et à SQLAlchemy de se connecter à la base de données PostgreSQL dans GitHub Actions, vérifiez que DATABASE\_URL est défini dans app\_nov.py et utilise les mêmes identifiants que le service PostgreSQL défini dans main.yml.

**Exécution du Workflow**

Une fois ces étapes configurées :

1. Poussez le projet sur la branche main de votre dépôt GitHub.
2. GitHub Actions détectera le fichier main.yml et exécutera les étapes définies à chaque push ou pull\_request.
3. Vous pourrez vérifier l’exécution de chaque étape dans l’onglet **Actions** de votre dépôt GitHub.

**Étape 6 : Validation des résultats**

Une fois le workflow exécuté, vérifiez les logs et les résultats des tests pour vous assurer que tout fonctionne comme prévu.