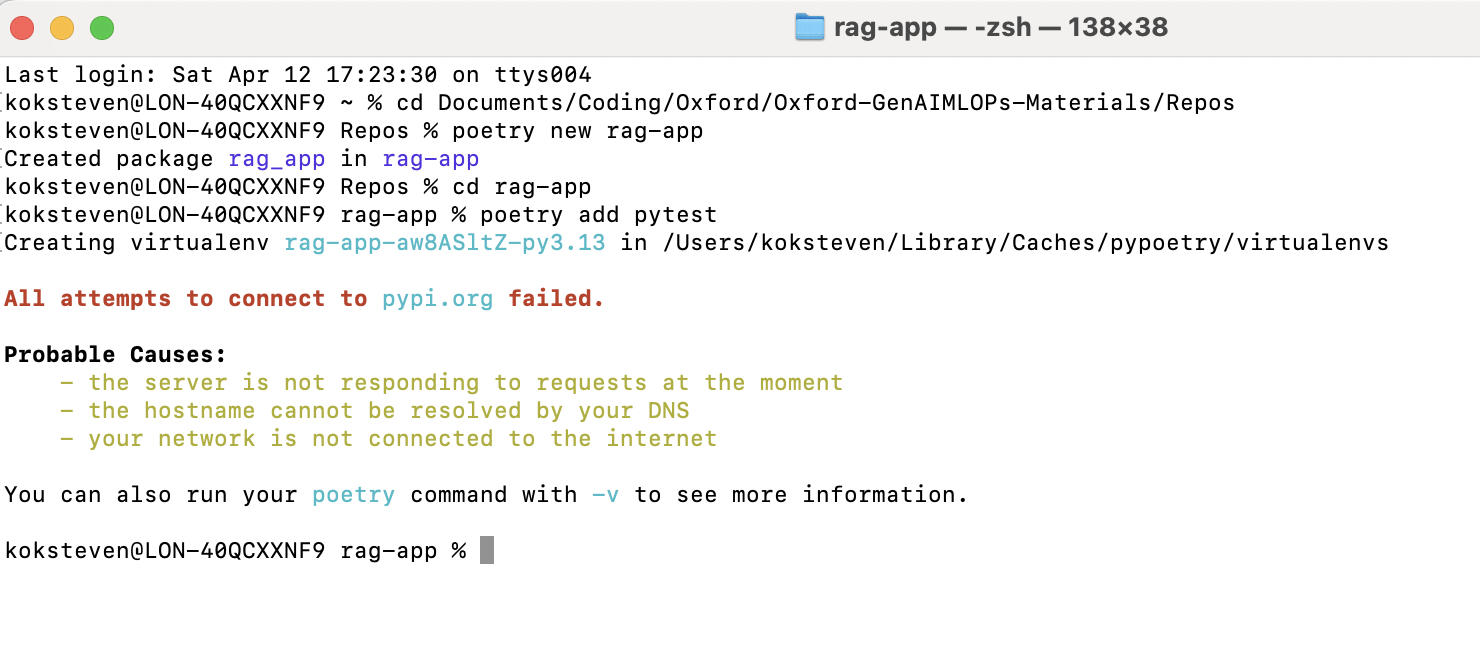
# Assignment 6 LLMOPs with AWS

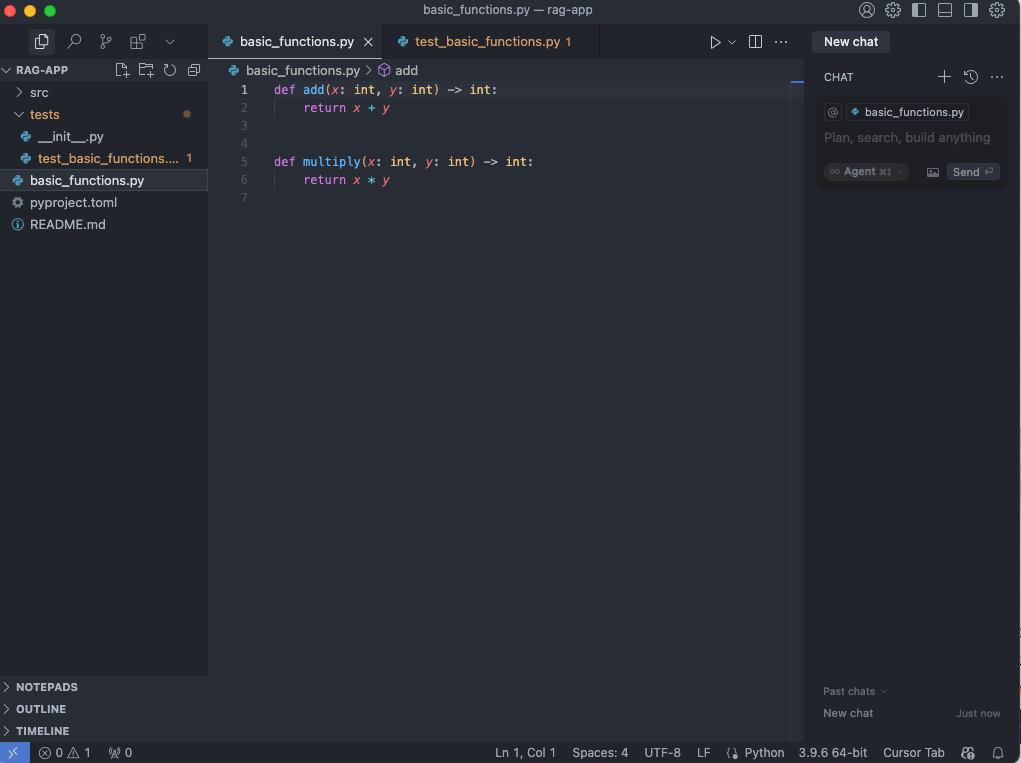
## Steven Kok

## 00 – Python Fundamentals – A Warmp Up

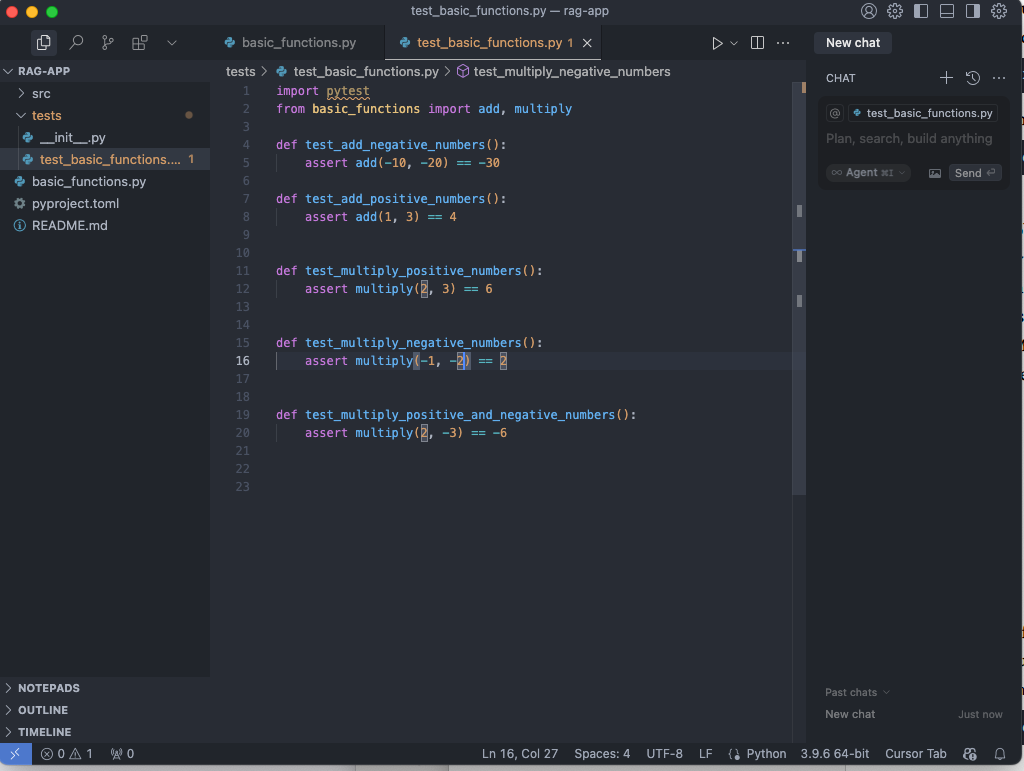
### 1. Create a directory called ‘rag-app’ and use the Python dependency management solution poetry to initialize your project. Use poetry to install the pytest.



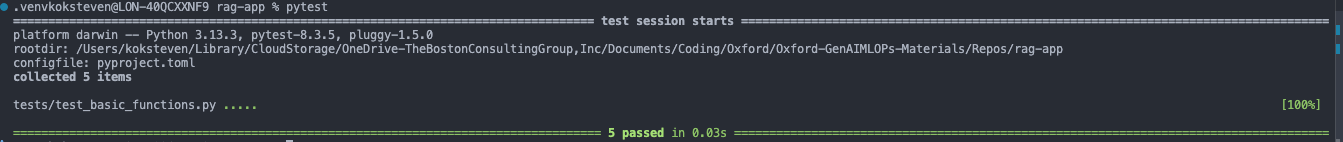
### 2. Write a small python script with functions within it that will add two numbers, subtract two numbers and multiply two numbers. You can call the script ‘basic\_functions.py’.



### 3. Write unit tests for all three of these functions inside a folder called tests (create this under your ‘rag-app’ folder) and put these inside a Python script called test\_basic\_functions.py.

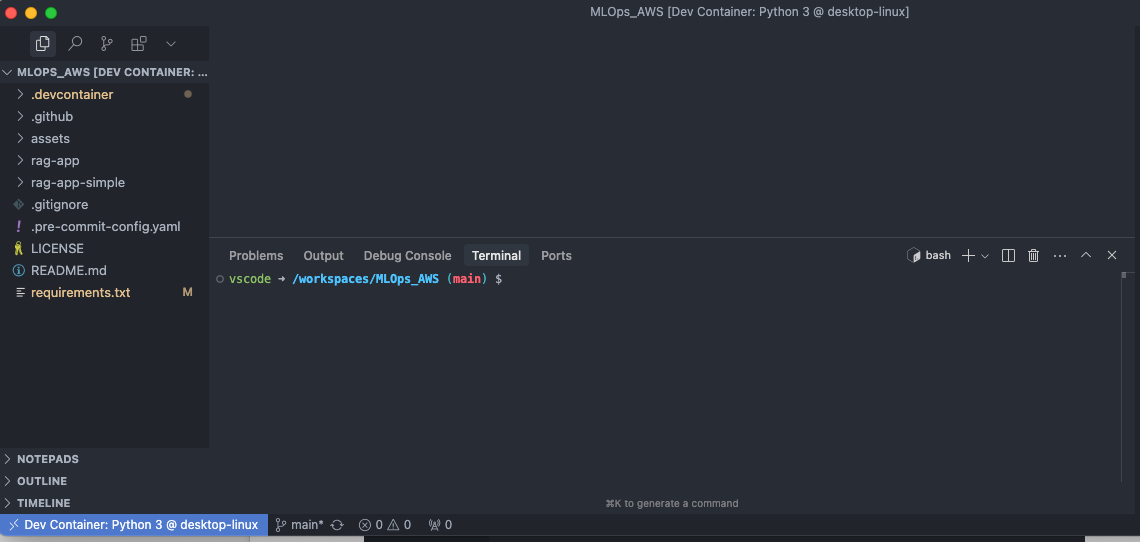


### 4. Run pytest to return a unit test completion report.

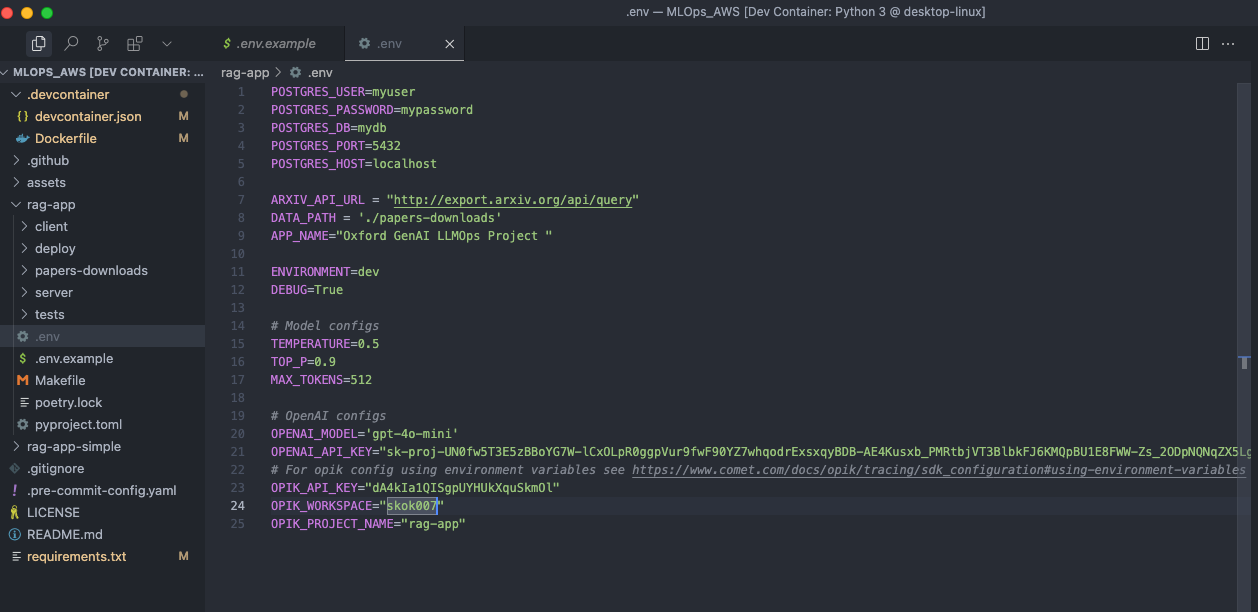


## 01 – Project Setup, FastAPI & Docker

### 1. Clone the project repository, 2. Open the repository in IDE and 3. Rebuild and reopen container



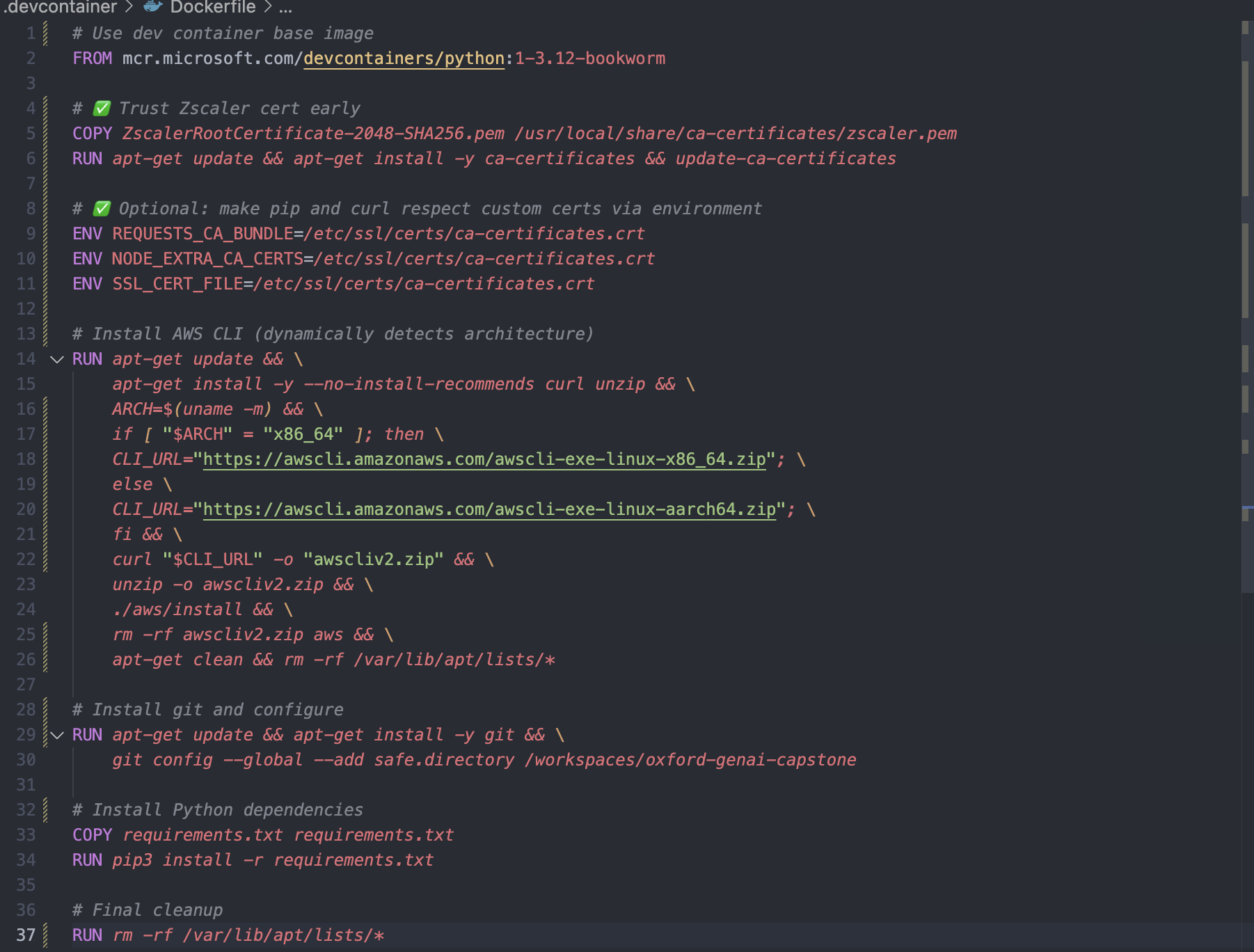
### 4. create .env, 5. Fill missing variables add Opik details



### 7. make install

I had multiple issues with this as I have a corporate laptop behind Zscaler. Had to re-train on the use of certificates for npm and docker.

Adjusted dockerfile here



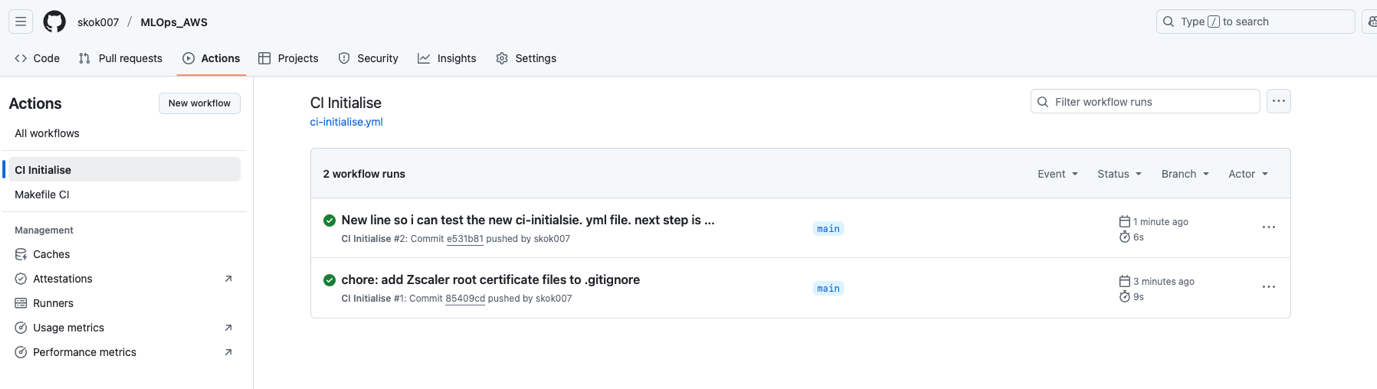
### 8 make run-app

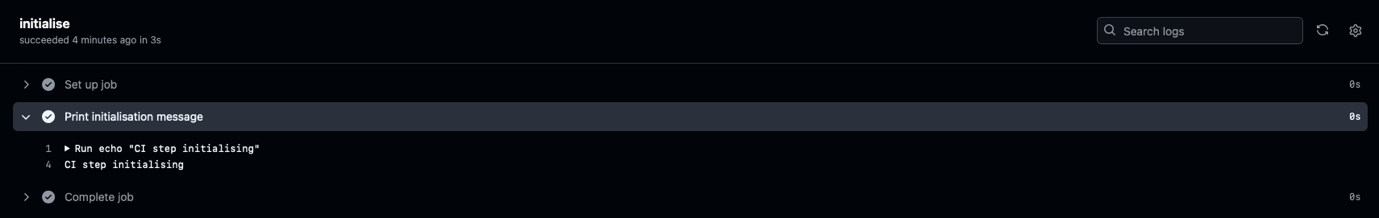
# 02 GitHub Actions

## 1 Create a ci-initialise.yml and build a simple workflow that will print the message CI step initalising



## 3 ,4 make small change and commit and push this change

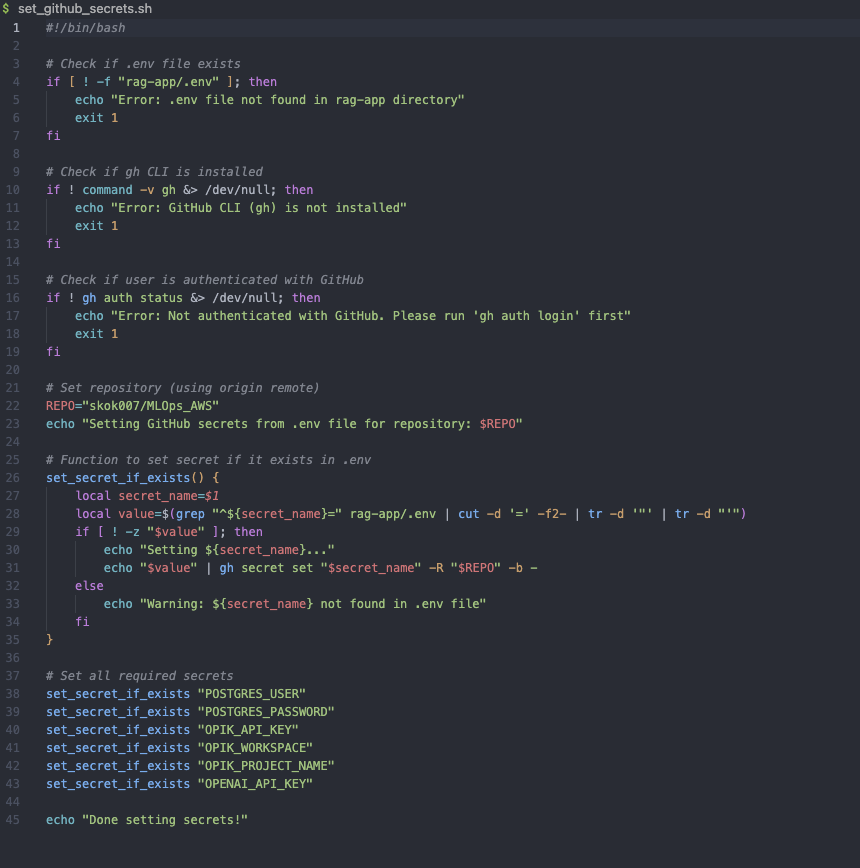




## Adapt the yml file so that it triggers a run of pytest

By trying to do this I realised a lot of the tests failed. And the reason for this is that there were some aspects of the workflow missing; including the installation of dependencies, and needed to setup secrets in Github to store all the necessary api keys and global variables.

So first I proceeded to setup secrets. I wanted to do this as securely as possible, so created a script that extracted the secrets from the .env files



The adjusted workflow has the following requirements:

I'll add detailed comments to the workflow and then provide a summary of the requirements.

First, let me update the workflow with detailed comments:

*1. \*\*Trigger Conditions:\*\**

*- Run on every push to any branch*

*- Automatically triggered by code changes*

*2. \*\*Environment Setup:\*\**

*- Run on Ubuntu latest*

*- Use Python 3.12 (project requirement)*

*- Set up Poetry for dependency management*

*3. \*\*Security Requirements:\*\**

*- Store sensitive values in GitHub Secrets:*

*- Database credentials (POSTGRES\_USER, POSTGRES\_PASSWORD)*

*- API keys (OPIK\_API\_KEY, OPENAI\_API\_KEY)*

*- Workspace configurations (OPIK\_WORKSPACE, OPIK\_PROJECT\_NAME)*

*4. \*\*Application Configuration:\*\**

*- Set test environment variables*

*- Configure database connection settings*

*- Set up API endpoints and paths*

*- Configure model generation parameters*

*- Set up OpenAI and Opik API settings*

*5. \*\*Performance Optimization:\*\**

*- Cache Poetry installation*

*- Cache Python dependencies*

*- Cache pip packages*

*- Use specific Poetry version (1.8.2)*

*6. \*\*Testing Requirements:\*\**

*- Run all tests in the tests/ directory*

*- Generate verbose test output*

*- Create JUnit XML test reports*

*- Store test results as artifacts*

*7. \*\*Error Handling:\*\**

*- Continue workflow even if tests fail*

*- Preserve test results for failed runs*

*- Use non-interactive mode for Poetry*

*- Configure virtualenvs in-project for reliability*

*8. \*\*Directory Structure:\*\**

*- Work in the rag-app directory*

*- Use working-directory for consistent path handling*

*- Maintain proper project structure*

The testing process found a multitude of errors in the code. Especially with the Opik setup. So I eventually commented out the tests for this stage. Will try to fix later

The final ci-initialise.yml here:

*# This workflow runs automated tests whenever code is pushed to any branch*

name: CI Initialise

*# Define when the workflow should run*

on:

*# push:*

*# branches:*

*# - '\*' # Run on push to any branch*

workflow\_dispatch: *# Manual trigger only*

jobs:

test:

*# Run on the latest Ubuntu runner*

runs-on: ubuntu-latest

*# Add PostgreSQL service*

services:

postgres:

image: ankane/pgvector:latest *# Use pgvector image instead of standard postgres*

env:

POSTGRES\_DB: test\_db

POSTGRES\_USER: test\_user

POSTGRES\_PASSWORD: test\_password

ports:

- 5432:5432

options: >-

--health-cmd pg\_isready

--health-interval 10s

--health-timeout 5s

--health-retries 5

*# Define environment variables needed for the application*

*# These variables are used by the application's Settings class*

env:

*# Basic application settings*

ENVIRONMENT: test

APP\_NAME: rag-app

DEBUG: "true"

*# Database connection settings*

POSTGRES\_HOST: localhost

POSTGRES\_DB: test\_db

POSTGRES\_USER: test\_user

POSTGRES\_PASSWORD: test\_password

POSTGRES\_PORT: 5432

*# API endpoints and paths*

ARXIV\_API\_URL: https://export.arxiv.org/api/query

DATA\_PATH: ./data

*# Model generation parameters*

TEMPERATURE: "0.7"

TOP\_P: "0.9"

MAX\_TOKENS: "1000"

*# Opik API settings (for AI model evaluation)*

OPIK\_API\_KEY: ${{ secrets.OPIK\_API\_KEY }}

OPIK\_WORKSPACE: ${{ secrets.OPIK\_WORKSPACE }}

OPIK\_PROJECT\_NAME: ${{ secrets.OPIK\_PROJECT\_NAME }}

OPIK\_ENVIRONMENT: test *# Add this to distinguish test environment*

*# OpenAI settings*

OPENAI\_MODEL: gpt-3.5-turbo

OPENAI\_API\_KEY: ${{ secrets.OPENAI\_API\_KEY }}

*# Poetry version to use*

POETRY\_VERSION: "1.8.2"

steps:

*# Step 1: Check out the code*

- uses: actions/checkout@v4

*# Step 2: Set up Python environment*

- name: Set up Python

uses: actions/setup-python@v5

with:

python-version: '3.12' *# Use Python 3.12 as required by the project*

cache: 'pip' *# Enable pip caching for faster dependency installation*

*# Step 3: Cache Poetry installation*

*# This speeds up the workflow by reusing the Poetry installation*

- name: Cache Poetry installation

uses: actions/cache@v3

with:

path: ~/.local

key: poetry-${{ runner.os }}-${{ env.POETRY\_VERSION }}

*# Step 4: Cache Poetry dependencies*

*# This speeds up the workflow by reusing the virtual environment*

- name: Cache Poetry dependencies

uses: actions/cache@v3

with:

path: ~/.cache/pypoetry

key: poetry-deps-${{ runner.os }}-${{ hashFiles('\*\*/poetry.lock') }}

restore-keys: |

poetry-deps-${{ runner.os }}-

*# Step 5: Install Poetry package manager*

- name: Install Poetry

run: |

# Install specific version of Poetry

curl -sSL https://install.python-poetry.org | python3 - --version ${{ env.POETRY\_VERSION }}

# Configure Poetry to create virtualenvs in the project directory

poetry config virtualenvs.create true

poetry config virtualenvs.in-project true

*# Step 6: Install project dependencies*

- name: Install dependencies

working-directory: ./rag-app

run: poetry install --no-interaction --no-root *# Install dependencies without user interaction*

*# Step 7: Install rag-app repository (demonstration)*

- name: Install rag-app repository

run: |

echo "Installing rag-app repository..."

cd rag-app

poetry install --no-interaction

echo "Repository installation complete!"

*# Step 8: Run the test suite*

- name: Run tests

working-directory: ./rag-app

run: |

poetry run pytest tests/ -v --junitxml=test-results.xml # Run tests and generate XML report

*# Step 9: Upload test results as artifacts*

*# This makes test results available even if tests fail*

- name: Upload test results

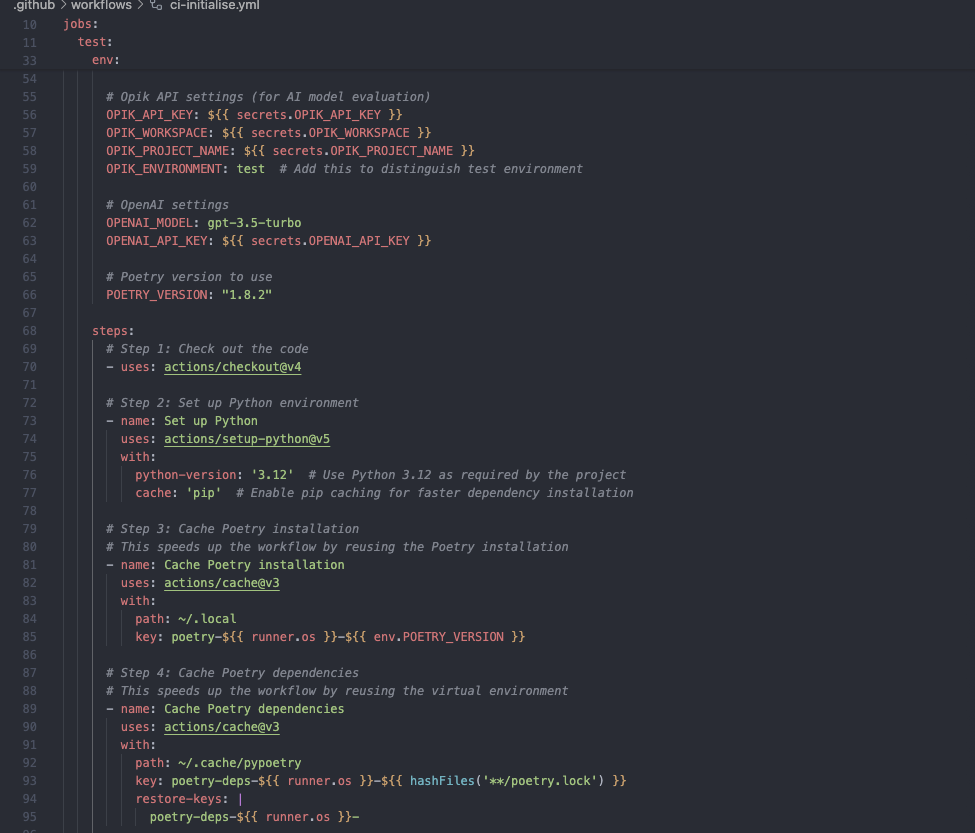
uses: actions/upload-artifact@v4

if: always() # Run even if tests fail

with:

name: test-results

path: rag-app/test-results.xml



# 03 – Database Setup

## 1. Create a container for the database

I had to again address the issues of being behind Zscaler. So had to move to http vs. https in the dockerfile. Added this code:

*# Override repositories to use HTTP instead of HTTPS to skip SSL verification*

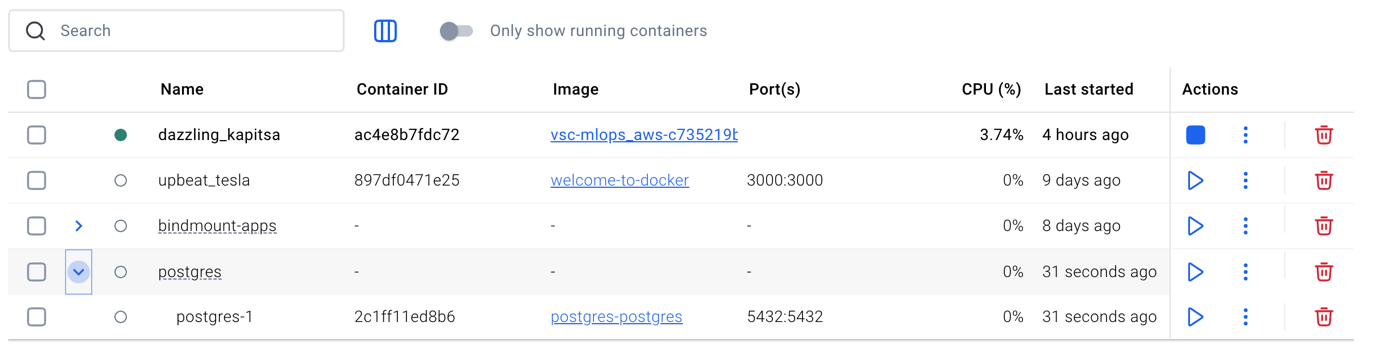
RUN echo "http://dl-cdn.alpinelinux.org/alpine/v3.21/main" > /etc/apk/repositories && \

echo "http://dl-cdn.alpinelinux.org/alpine/v3.21/community" >> /etc/apk/repositories && \

apk update && apk add --no-cache ca-certificates && update-ca-certificates

COPY ZscalerRootCertificate-2048-SHA256.pem /usr/local/share/ca-certificates/zscaler.crt

After this it worked. See here docker dashboard:



## 2. Then I ran the test with this command with all the right credentials:

PGPASSWORD=mypassword psql -h localhost -p 5432 -U myuser -d mydb

mydb=# \dt

List of relations

Schema | Name | Type | Owner

--------+--------+-------+--------

public | papers | table | myuser

(1 row)

mydb=# \du

List of roles

Role name | Attributes

-----------+------------------------------------------------------------

myuser | Superuser, Create role, Create DB, Replication, Bypass RLS

mydb=#

## 3. adding comments to init\_pgvector.sql

*/\* Create the vector extension \*/*

CREATE EXTENSION IF NOT EXISTS vector;

*/\**

*\* Vector dimension explanation:*

*\* The dimension 384 is commonly used with models like:*

*\* - all-MiniLM-L6-v2 (384 dimensions)*

*\* - all-mpnet-base-v2 (384 dimensions)*

*\**

*\* Different models have different embedding dimensions:*

*\* - BERT base: 768 dimensions*

*\* - BERT large: 1024 dimensions*

*\* - OpenAI text-embedding-ada-002: 1536 dimensions*

*\* - OpenAI text-embedding-3-small: 1536 dimensions*

*\* - OpenAI text-embedding-3-large: 3072 dimensions*

*\**

*\* If using a different model, adjust the vector dimension accordingly.*

*\* For example, if using OpenAI's text-embedding-ada-002, change to:*

*\* embedding vector(1536)*

*\*/*

CREATE TABLE papers (

id SERIAL PRIMARY KEY,

title TEXT NOT NULL,

summary TEXT NOT NULL,

chunk TEXT NOT NULL,

embedding vector(384)

);

4. Data ingestion and Embedding

1- Where in the code is the search against arXiv API?

Here:

Here: if \_\_name\_\_ == "\_\_main\_\_":

*# papers = fetch\_papers(query="ti:perovskite", max\_results=10)*

papers = fetch\_papers\_paginated(

*query*="ti:perovskite", *max\_results*=20, *results\_per\_page*=5, *wait\_time*=5

)

2 Try different variants of your query string using the API quickstart examples

These are the query variants I used

*# ========================*

*# Query Variant 1: Basic title-only search*

*# ========================*

*# papers = fetch\_papers\_paginated(*

*# query="ti:perovskite", max\_results=20, results\_per\_page=5, wait\_time=5*

*# )*

*# ========================*

*# Query Variant 2: Title or abstract contains "perovskite"*

*# ========================*

*# papers = fetch\_papers\_paginated(*

*# query="ti:perovskite OR abs:perovskite", max\_results=20, results\_per\_page=5, wait\_time=5*

*# )*

*# ========================*

*# Query Variant 3: Filter to category - materials science in condensed matter*

*# ========================*

*# papers = fetch\_papers\_paginated(*

*# query="(ti:perovskite OR abs:perovskite) AND cat:cond-mat.mtrl-sci", max\_results=20, results\_per\_page=5, wait\_time=5*

*# )*

*# ========================*

*# Query Variant 4: Keyword combo (solar + perovskite in title)*

*# ========================*

*# papers = fetch\_papers\_paginated(*

*# query="ti:perovskite AND ti:solar", max\_results=20, results\_per\_page=5, wait\_time=5*

*# )*

*# ========================*

*# Query Variant 5: Specific author (Michael Grätzel as example – which comes back empty)*

*# ========================*

*# papers = fetch\_papers\_paginated(*

*# query="ti:perovskite AND au:Grätzel", max\_results=20, results\_per\_page=5, wait\_time=5*

*# )*

3 run ingestion results