TRƯỜNG ĐẠI HỌC ĐÔNG Á

**KHOA CÔNG NGHỆ THÔNG TIN**

-----------🙡⯎⯎⯎🙣-----------



BÁO CÁO KẾT THÚC HỌC PHẦN

**KIỂM THỬ PHẦN MỀM 2**

**Đề tài:**

**CONTINIOUSE TESTING DEVELOPMENT**

**GVHD: ThS. Dương Thị Mai Nga**

**SVTH : Nguyễn Phương Trà**

**A Trưởng**

**Tâm**

**Lớp : ST23C**

*Đà Nẵng, tháng 5-2025*

MỤC LỤC

[PHẦN 6 : CONTINUOUS TESTING EVELOPMENT 3](#_Toc199580157)

[1. Lý thuyết về DevOps 3](#_Toc199580158)

[2 Deployment, Code Coverage and Quality 5](#_Toc199580159)

[3. Thực hành kiểm thử mã Python – Unit Test 6](#_Toc199580160)

[PHẦN 7: CONTINUOUS TESTING EVELOPMENT (TT) 11](#_Toc199580161)

[1. Continuous Integration (CI) 11](#_Toc199580162)

[Khái niệm CI và quy trình thiết lập CI 11](#_Toc199580163)

[2. Cài đặt GitHub Actions Runner 12](#_Toc199580164)

[3. Continuous Delivery (CD) 14](#_Toc199580165)

[4. Cài đặt và Cấu hình CD trên GitHub với Docker Hub 15](#_Toc199580166)

[PHẦN 8: CONTINUOUS TESTING EVELOPMENT (TT) 19](#_Toc199580167)

[1. Advanced CI 19](#_Toc199580168)

[2. Flask-Admin 20](#_Toc199580169)

[3. Triển khai Flask-Admin trên render và test CI/CD 21](#_Toc199580170)

[KẾT LUẬN 27](#_Toc199580171)

# PHẦN 6 : CONTINUOUS TESTING EVELOPMENT

## Lý thuyết về DevOps

**Khái niệm DevOps**

DevOps là một triết lý, một phương pháp luận kết hợp chặt chẽ giữa Phát triển (Development) và Vận hành (Operations) trong chu trình phát triển phần mềm. Mục tiêu chính của DevOps là rút ngắn chu kỳ phát triển hệ thống, tăng tốc độ triển khai phần mềm một cách đáng kể và đảm bảo chất lượng sản phẩm cuối cùng ở mức cao nhất.

DevOps phá vỡ các rào cản truyền thống giữa các nhóm phát triển, vận hành, kiểm thử và bảo mật, khuyến khích sự hợp tác, giao tiếp và chia sẻ trách nhiệm. Nó tập trung vào việc tự động hóa các quy trình, từ việc viết mã, kiểm thử, đóng gói, triển khai cho đến giám sát và phản hồi.

**Các trụ cột chính của DevOps bao gồm:**

* **Văn hóa (Culture):** Khuyến khích sự hợp tác, tin tưởng và chia sẻ trách nhiệm giữa các nhóm.
* **Tự động hóa (Automation):** Tự động hóa càng nhiều quy trình càng tốt để giảm thiểu lỗi do con người và tăng tốc độ.
* **Lean (Tinh gọn):** Loại bỏ lãng phí, tập trung vào việc tạo ra giá trị liên tục.
* **Đo lường (Measurement):** Thu thập dữ liệu và phân tích hiệu suất để liên tục cải thiện.
* **Chia sẻ (Sharing):** Chia sẻ kiến thức, công cụ và kinh nghiệm giữa các nhóm.

**Vai trò của DevOps trong việc tự động hóa quy trình CI/CD**

DevOps đóng vai trò trung tâm trong việc tự động hóa quy trình Tích hợp liên tục (Continuous Integration - CI) và Triển khai liên tục (Continuous Delivery/Deployment - CD). CI/CD là xương sống của DevOps, cho phép các nhóm phát triển và vận hành làm việc hiệu quả hơn, đưa sản phẩm ra thị trường nhanh hơn với chất lượng cao hơn.

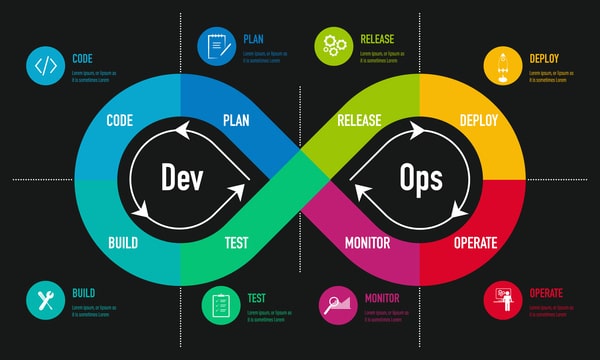
**Continuous integration (CI):**

* + **Tự động hóa Build:** DevOps thúc đẩy việc tự động hóa quá trình biên dịch mã nguồn, tạo ra các bản dựng (builds) nhất quán và đáng tin cậy mỗi khi có thay đổi mã.
  + **Tự động hóa Test:** Các bài kiểm thử đơn vị (unit tests), kiểm thử tích hợp (integration tests) được tự động chạy sau mỗi lần tích hợp mã. Điều này giúp phát hiện lỗi sớm, giảm chi phí sửa lỗi và đảm bảo chất lượng mã.
  + **Phản hồi nhanh:** Nhờ tự động hóa, các nhà phát triển nhận được phản hồi ngay lập tức về chất lượng mã và sự phù hợp của các thay đổi, cho phép họ sửa lỗi kịp thời.

**Continuous deployment (CD):**

* + **Tự động hóa Release:** DevOps giúp tự động hóa quá trình đóng gói ứng dụng và chuẩn bị cho việc triển khai.
  + **Tự động hóa Triển khai:** Các công cụ và quy trình DevOps cho phép tự động triển khai ứng dụng lên các môi trường khác nhau (staging, production) một cách nhất quán và lặp lại. Điều này loại bỏ các bước thủ công dễ gây lỗi và tăng tốc độ triển khai.
  + **Giám sát và Phản hồi:** Sau khi triển khai, các công cụ DevOps tiếp tục giám sát hiệu suất ứng dụng và thu thập phản hồi, giúp các nhóm nhanh chóng xác định và giải quyết các vấn đề phát sinh.

DevOps cung cấp khuôn khổ và các nguyên tắc để xây dựng một đường ống CI/CD hiệu quả, tự động hóa toàn bộ vòng đời phát triển phần mềm, từ cam kết mã nguồn đến triển khai sản phẩm cuối cùng.



*Hình 1: Mô hình “Infinity Loop” trong DevOps*

## 2 Deployment, Code Coverage and Quality

**Cài đặt các công cụ triển khai tự động**

Các công cụ triển khai tự động là nền tảng cho quy trình CI/CD, giúp tự động hóa các bước từ biên dịch mã, kiểm thử đến triển khai ứng dụng.

* **Jenkins:** Là một máy chủ tự động hóa mã nguồn mở hàng đầu, Jenkins hỗ trợ xây dựng, kiểm thử và triển khai bất kỳ dự án nào. Nó có một hệ sinh thái plugin phong phú, cho phép tích hợp với hầu hết các công cụ và công nghệ khác. Jenkins hoạt động bằng cách định nghĩa các "pipeline" (đường ống) – một chuỗi các bước được thực hiện theo thứ tự.
  + **Ưu điểm:** Rất linh hoạt, có thể mở rộng, cộng đồng lớn.
  + **Nhược điểm:** Cần cấu hình thủ công nhiều, yêu cầu máy chủ riêng.
* **GitLab CI/CD:** Là một phần không thể thiếu của nền tảng GitLab, cung cấp khả năng CI/CD tích hợp sẵn. Nó cho phép người dùng định nghĩa các pipeline trực tiếp trong kho mã nguồn thông qua file .gitlab-ci.yml. GitLab CI/CD sử dụng "Runner" để thực thi các job trong pipeline.
  + **Ưu điểm:** Tích hợp sâu với GitLab, dễ sử dụng, cấu hình qua YAML.
  + **Nhược điểm:** Phụ thuộc vào GitLab.
* **GitHub Actions:** Là một tính năng CI/CD được tích hợp trực tiếp vào GitHub. Nó cho phép tự động hóa các quy trình làm việc (workflows) dựa trên các sự kiện trong kho mã nguồn (ví dụ: push, pull request). Các workflow được định nghĩa bằng file YAML và chạy trên các "runner" do GitHub quản lý hoặc tự host.
  + **Ưu điểm:** Tích hợp sâu với GitHub, miễn phí cho các kho mã nguồn công khai, dễ cấu hình.
  + **Nhược điểm:** Phụ thuộc vào GitHub.

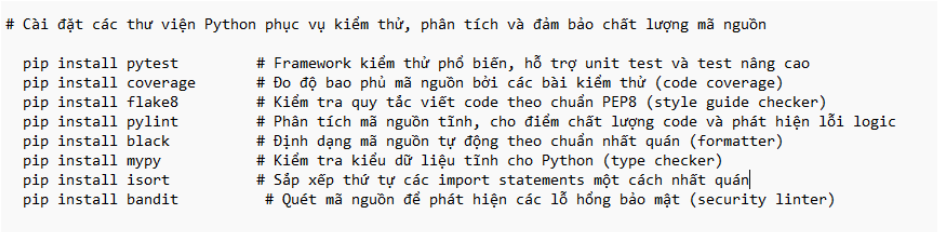
**Cài đặt các thư viện Python phục vụ kiểm thử, phân tích và đảm bảo chất lượng mã nguồn**

**Bước 1: Tạo môi trường ảo**

python -m venv venv #tạo môi trường ảo

venv\Scripts\activate #kích hoạt môi trường ảo

**Bước 2:**

****

## 3. Thực hành kiểm thử mã Python – Unit Test

Ví dụ đơn giản với pytest:

Tạo file calculator.py

A black background with white text

AI-generated content may be incorrect.

Tạo file test\_calculator.py

A black screen with colorful text

AI-generated content may be incorrect.

Chạy câu lệnh pytest test\_calculator.py

Kết quả khi test A screenshot of a computer program

AI-generated content may be incorrect.

Việc đánh giá chất lượng mã nguồn là một phần quan trọng của quy trình CI/CD, giúp đảm bảo mã nguồn sạch, dễ bảo trì và ít lỗi.

* **pytest:** Là một framework kiểm thử Python mạnh mẽ và dễ sử dụng. Nó cho phép viết các bài kiểm thử đơn vị và kiểm thử tích hợp một cách ngắn gọn và rõ ràng.

**Cài đặt:** pip install pytest

**Sử dụng:** pytest (để chạy tất cả các test trong thư mục hiện tại)

* **coverage.py:** Là một công cụ để đo lường độ bao phủ mã nguồn (code coverage) của các chương trình Python. Nó giúp xác định phần trăm mã nguồn đã được thực thi bởi các bài kiểm thử.

**Đo lường độ bao phủ mã nguồn bằng code coverage**

Độ bao phủ mã nguồn (Code Coverage) là một chỉ số quan trọng trong kiểm thử phần mềm, đo lường tỷ lệ phần trăm mã nguồn của ứng dụng đã được thực thi bởi các bài kiểm thử. Nó giúp xác định những phần mã nguồn nào chưa được kiểm thử, từ đó giúp các nhà phát triển bổ sung các bài kiểm thử cần thiết.

**Các loại độ bao phủ phổ biến:**

* **Statement Coverage (Độ bao phủ câu lệnh):** Đo lường số lượng câu lệnh đã được thực thi.
* **Branch Coverage (Độ bao phủ nhánh):** Đo lường số lượng nhánh (ví dụ: if/else, for, while) đã được thực thi.
* **Function Coverage (Độ bao phủ hàm):** Đo lường số lượng hàm đã được gọi.
* **Line Coverage (Độ bao phủ dòng):** Tương tự Statement Coverage, đo lường số lượng dòng mã đã được thực thi.
* **Cách đo lường (sử dụng coverage.py):**

**Thực hành kiểm thử mã Python – Code Coverage & Quality**

**Đo độ bao phủ kiểm thử với coverage.py**

**Bước 1: Cài đặt:**

Pip install pytest coverage

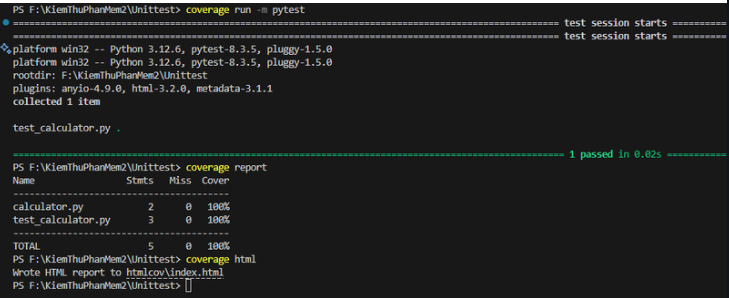
**Bước 2: Chạy kiểm thử và đo coverage**

coverage run -m pytest

coverage report

coverage html

**Kết quả:**

****

**Kiểm tra chất lượng mã với flake8, pylint**

**Bước 3: Cài đặt:**

pip install flake8

pip install pylint

**Bước 4: Chạy kiểm kiểm tra chất lượng mã**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Các chỉ số đánh giá chất lượng mã nguồn (code quality metrics)**

Ngoài độ bao phủ mã nguồn, có nhiều chỉ số khác được sử dụng để đánh giá chất lượng và khả năng bảo trì của mã nguồn.

* **Maintainability Index (Chỉ số khả năng bảo trì):**
  + Là một chỉ số tổng hợp, thường được tính từ 0 đến 100, cho biết mức độ dễ dàng để duy trì và sửa đổi mã nguồn.
  + Chỉ số cao hơn (ví dụ: trên 60-80) cho thấy mã dễ bảo trì hơn, trong khi chỉ số thấp hơn có thể chỉ ra mã phức tạp, khó hiểu và khó thay đổi.
  + Nó thường được tính toán dựa trên các yếu tố như độ phức tạp cyclomatic, số lượng dòng mã vật lý, và số lượng toán tử/toán hạng (Halstead Complexity Metrics).
* **Cyclomatic Complexity (Độ phức tạp Cyclomatic):**
  + Đo lường số lượng đường dẫn độc lập trong mã nguồn của một hàm hoặc một module.
  + Một giá trị cao cho thấy mã có nhiều nhánh điều kiện và vòng lặp, làm cho nó khó hiểu, khó kiểm thử và dễ phát sinh lỗi hơn.
  + Thông thường, độ phức tạp cyclomatic dưới 10-15 được coi là tốt. Giá trị trên 20-30 thường là dấu hiệu của "mã mùi" (code smell) cần được tái cấu trúc.
  + Nó được tính toán bằng cách tạo một biểu đồ luồng điều khiển của mã và áp dụng công thức: M=E−N+2P, trong đó E là số cạnh, N là số nút, và P là số thành phần kết nối (thường là 1 cho một hàm).
* **Các chỉ số khác:**
  + **Lines of Code (LOC):** Tổng số dòng mã. Mặc dù không phải là chỉ số chất lượng trực tiếp, nhưng số lượng LOC quá lớn có thể chỉ ra sự phức tạp.
  + **Duplication (Mã trùng lặp):** Phần trăm mã nguồn bị trùng lặp. Mã trùng lặp làm tăng kích thước dự án, khó bảo trì và dễ gây lỗi.
  + **Linting/Static Analysis Warnings:** Số lượng cảnh báo từ các công cụ phân tích tĩnh (ví dụ: Pylint, Flake8 cho Python) về các vấn đề về phong cách mã hóa, lỗi tiềm ẩn hoặc vi phạm quy tắc.
  + **Test-to-Code Ratio:** Tỷ lệ giữa số lượng dòng mã kiểm thử và số lượng dòng mã sản phẩm.

|  |  |
| --- | --- |
| **Tiêu chí** | **Công cụ hỗ trợ trong Python** |
| Style (PEP8) | flake8, black |
| Bug tiềm ẩn | pylint, mypy |
| Phức tạp mã nguồn | radon |
| Format đồng bộ | black |

Việc theo dõi và cải thiện các chỉ số này giúp các nhóm phát triển duy trì một codebase chất lượng cao, dễ mở rộng và bền vững.

# PHẦN 7: CONTINUOUS TESTING EVELOPMENT (TT)

## 1. Continuous Integration (CI)

### Khái niệm CI và quy trình thiết lập CI

**Continuous Integration (CI - Tích hợp liên tục)** là một thực hành phát triển phần mềm mà theo đó các nhà phát triển thường xuyên tích hợp các thay đổi mã nguồn của họ vào một kho mã nguồn chung (ví dụ: Git repository). Mỗi lần tích hợp được xác minh bằng một bản dựng tự động (automated build) và các bài kiểm thử tự động (automated tests) để phát hiện lỗi tích hợp càng sớm càng tốt.

**Mục tiêu chính của CI:**

* Giảm thiểu "đau đớn" của việc tích hợp bằng cách thực hiện nó thường xuyên.
* Phát hiện lỗi tích hợp sớm, giảm chi phí và thời gian sửa lỗi.
* Đảm bảo rằng mã nguồn luôn ở trạng thái có thể triển khai được (deployable state).
* Cung cấp phản hồi nhanh chóng cho các nhà phát triển về chất lượng mã của họ.

**Quy trình thiết lập CI điển hình:**

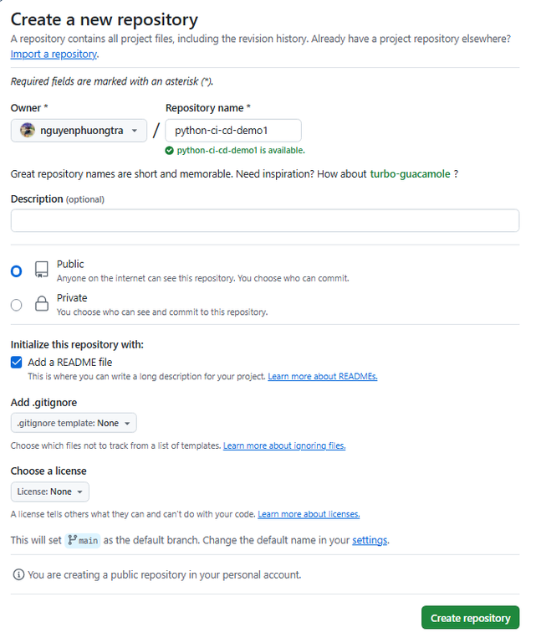
1. **Hệ thống kiểm soát phiên bản (Version Control System - VCS):** Sử dụng một VCS như Git để quản lý mã nguồn. Mọi thay đổi đều được cam kết (commit) và đẩy (push) lên kho mã nguồn chung.
2. **Máy chủ CI (CI Server):** Cài đặt một máy chủ CI (ví dụ: Jenkins, GitLab CI/CD, GitHub Actions).
3. **Cấu hình Pipeline:** Định nghĩa một "pipeline" (đường ống) tự động hóa các bước CI. Pipeline này thường được cấu hình trong một file nằm trong kho mã nguồn (ví dụ: .gitlab-ci.yml, .github/workflows/\*.yml, Jenkinsfile).
4. **Kích hoạt tự động:** Pipeline được kích hoạt tự động mỗi khi có một thay đổi mã nguồn được đẩy lên kho chung (ví dụ: push vào nhánh main, tạo pull request).
5. **Các bước trong Pipeline:**
   * **Fetch/Clone Code:** Lấy mã nguồn mới nhất từ kho.
   * **Build:** Biên dịch mã nguồn (nếu có), tạo ra các artifact (ví dụ: file thực thi, gói ứng dụng).
   * **Test:** Chạy các bài kiểm thử tự động (unit tests, integration tests).
   * **Code Quality Check:** Chạy các công cụ phân tích tĩnh, đo lường code coverage.
   * **Package/Containerize:** Đóng gói ứng dụng (ví dụ: tạo Docker image).
   * **Notify:** Gửi thông báo về kết quả build/test (thành công/thất bại) cho các nhà phát triển.
6. **Phản hồi nhanh:** Nếu bất kỳ bước nào trong pipeline thất bại, hệ thống CI sẽ ngay lập tức thông báo cho các nhà phát triển để họ có thể sửa lỗi kịp thời.

## 2. Cài đặt GitHub Actions Runner

GitHub Actions Runner là một ứng dụng mã nguồn mở thực thi các job trong workflow của GitHub Actions. Nó có thể được cài đặt trên nhiều hệ điều hành khác nhau (Linux, Windows, macOS) và có thể chạy các job trong nhiều môi trường (shell, Docker).

**Cài đặt và cấu hình CI trên Github Actions**

**Bước 1: Tạo repository trên GitHub**



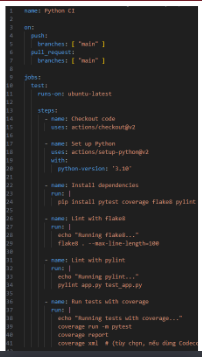
**Bước 2: Tạo workflow GitHub Actions**

**Tạo folder .github\workflows\python-ci.yml**

A screenshot of a computer program

AI-generated content may be incorrect.

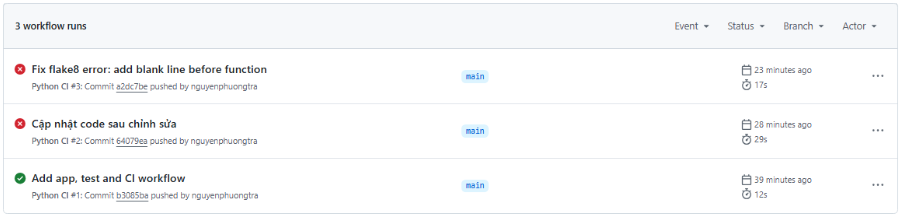
**Bước 3: Cấu hình pytest, flake8, pylint, coverage trong file python-ci.yml**



**Bước 4:Commit và đẩy lên để chạy CI**

* **git init**
* **git remote add origin https://github.com/nguyenphuongtra/python-ci-cd-demo.git**
* **git add .**
* **git commit -m "First commit"**
* **git branch -M main**
* **git push -u origin main**

**Bước 5:Kiểm tra CI chạy**



Sau khi cấu hình và chạy, Runner sẽ bắt đầu lắng nghe các job từ GitHub Actions và thực thi chúng dựa trên các labels đã được cấu hình.

**Tích hợp các bước như: build, test, deploy trong CI pipeline**

Việc tích hợp các bước build, test, deploy vào một pipeline CI/CD là cốt lõi để tự động hóa toàn bộ quy trình.

## 3. Continuous Delivery (CD)

**Thiết lập môi trường staging hoặc production trên GitHub Actions**

Continuous Delivery (CD - Giao hàng liên tục) là một phần mở rộng của CI, đảm bảo rằng phần mềm có thể được phát hành nhanh chóng và bền vững bất cứ lúc nào. Nó bao gồm việc tự động hóa quá trình phát hành phần mềm đến các môi trường khác nhau, từ môi trường kiểm thử (staging) đến môi trường sản phẩm (production).

Trong GitHub Actions, việc thiết lập môi trường được thực hiện thông qua từ khóa environment trong các job của workflow. Điều này giúp GitHub theo dõi các triển khai, hiển thị trạng thái của môi trường và cho phép bạn xem lịch sử triển khai.

Khi các job này chạy, GitLab sẽ hiển thị các môi trường này trong giao diện người dùng dưới mục "Deployments > Environments", cho phép bạn dễ dàng theo dõi trạng thái của từng môi trường và xem lịch sử triển khai.

**Cài đặt tự động deploy sau khi test đạt yêu cầu**

Để tự động triển khai sau khi các bài kiểm thử đạt yêu cầu, bạn chỉ cần đảm bảo rằng job triển khai (deploy job) được đặt trong một giai đoạn sau giai đoạn kiểm thử (test stage) và không có điều kiện when: manual hoặc when: delayed không mong muốn.

**Nguyên tắc:**

* **Thứ tự Stage:** Đảm bảo deploy stage nằm sau test stage trong định nghĩa stages.
* **Job Dependencies:** Job triển khai có thể phụ thuộc vào artifacts từ các job trước đó (ví dụ: bản dựng ứng dụng, kết quả kiểm thử).
* **Điều kiện kích hoạt:** Job triển khai sẽ tự động chạy nếu tất cả các job trong các stage trước đó (bao gồm cả test stage) đều thành công.

**Tích hợp CI/CD với Docker để triển khai ứng dụng**

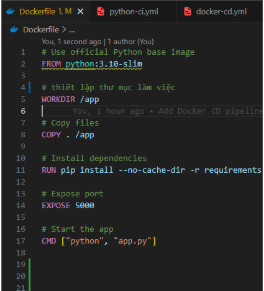
Docker là một công nghệ container hóa phổ biến, cho phép đóng gói ứng dụng và tất cả các phụ thuộc của nó vào một "container" độc lập, di động. Tích hợp Docker vào CI/CD giúp đảm bảo rằng ứng dụng sẽ chạy nhất quán trên mọi môi trường.

**Quy trình tích hợp điển hình:**

1. **Dockerfile:** Tạo một Dockerfile trong thư mục gốc của dự án để định nghĩa cách xây dựng Docker image của ứng dụng.
2. **Build Docker Image trong CI:** Trong pipeline CI, thêm một job để xây dựng Docker image từ Dockerfile.
3. **Push Image lên Registry:** Sau khi build thành công, đẩy Docker image lên một Docker Registry (ví dụ: Docker Hub, GitLab Container Registry, Google Container Registry).
4. **Deploy Container:** Trong job triển khai, kéo (pull) Docker image từ Registry và chạy nó trên máy chủ đích.

## 4. Cài đặt và Cấu hình CD trên GitHub với Docker Hub

**Bước 1: Tạo file Dockerfile**



**Bước 2 - Tạo file requirements.txt bằng câu lệnh:**

**pip freeze > requirements.txt**

**Bước 3: Trong thư mục .github/workflows/**

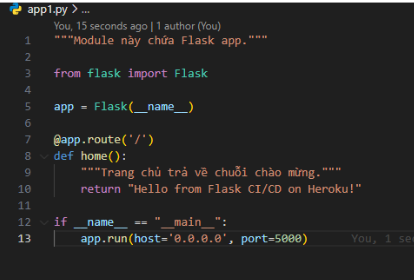
**tạo file docker-build.yml.**

A screen shot of a computer program

AI-generated content may be incorrect.

**Bước 4: Tạo file app1.py**

**pip install flask # để sử dụng**



**Bước 5: Tạo GitHub Secrets**

**Vào GitHub repo → Settings → Secrets and variables → Actions**

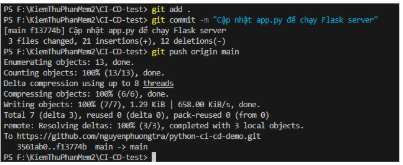
A screenshot of a computer

AI-generated content may be incorrect.

**Bước 6: Tạo Access Token trên Docker Hub**

* **Đăng nhập https://hub.docker.com/.**
* **Nhấp Account Settings (góc trên cùng bên phải) → Security.**
* **Nhấp Generate new token.**
* **Đặt tên (ví dụ: GitHub\_Actions\_CICD).**
* **Chọn quyền Read, Write, Delete.**
* **Nhấp Generate, sao chép token (ví dụ: dckr\_pat\_xxx).**
* **Thêm vào secret DOCKER\_PASSWORD trên GitHub**

**Bước 7:Push code lên GitHub**



**Bước 8: Kiểm tra ứng dụng**

* **docker pull nguyenphuongtra24/flask-ci-cd-demo:latest**
* **docker run -p 5000:5000 nguyenphuongtra24/flask-ci-cd-demo:latest**

A screen shot of a computer

AI-generated content may be incorrect.

**Kết quả:  
A screenshot of a computer

AI-generated content may be incorrect.**

# PHẦN 8: CONTINUOUS TESTING EVELOPMENT (TT)

## 1. Advanced CI

Tối ưu pipeline: chia nhiều stages, parallel jobs, caching

Để tối ưu hóa hiệu suất và hiệu quả của pipeline CI/CD, có một số kỹ thuật nâng cao được sử dụng:

* **Chia nhiều Stages (Giai đoạn):**
  + **Mục đích:** Tổ chức pipeline thành các nhóm công việc logic và tuần tự. Mỗi stage phải hoàn thành thành công trước khi stage tiếp theo bắt đầu.
  + **Lợi ích:** Dễ quản lý, dễ theo dõi lỗi (biết lỗi xảy ra ở giai đoạn nào), và cho phép các job trong cùng một stage chạy song song.
  + **Ví dụ:** build -> test -> security\_scan -> deploy.
* **Parallel Jobs (Các job song song):**
  + **Mục đích:** Chạy nhiều job cùng một lúc để giảm tổng thời gian thực hiện pipeline.
  + **Cách thực hiện:** Đặt các job vào cùng một stage. GitLab CI/CD sẽ tự động chạy song song các job trong cùng một stage nếu có đủ Runner khả dụng.
  + **Lợi ích:** Tăng tốc độ pipeline đáng kể, đặc biệt hữu ích khi có nhiều loại kiểm thử (unit, integration, E2E) hoặc khi cần build cho nhiều nền tảng.

Trong ví dụ này, unit\_test và integration\_test sẽ chạy song song.

* **Caching (Bộ nhớ đệm):**
  + **Mục đích:** Lưu trữ các file hoặc thư mục được tạo ra trong các lần chạy pipeline trước đó để tái sử dụng trong các lần chạy sau. Điều này giúp giảm thời gian cài đặt phụ thuộc hoặc biên dịch lại mã.
  + **Cách thực hiện:** Sử dụng từ khóa cache trong .gitlab-ci.yml. Bạn cần định nghĩa paths (các thư mục cần cache) và key (để xác định cache).
  + **Lợi ích:** Giảm đáng kể thời gian chạy pipeline, đặc biệt là các bước cài đặt phụ thuộc (ví dụ: pip install, npm install).

Cache sẽ được tải về trước khi job chạy và được lưu lại sau khi job hoàn thành.

**Sử dụng workflow templates trong GitHub Actions để tái sử dụng cấu hình**

Khi các dự án phát triển lớn hơn hoặc có nhiều dự án với cấu hình CI/CD tương tự, việc tái sử dụng cấu hình trở nên rất quan trọng để tránh trùng lặp và dễ quản lý. GitHub Actions hỗ trợ "Reusable Workflows" (Workflows có thể tái sử dụng) để đưa các phần của workflow vào các workflow khác.

**Cách tạo một Reusable Workflow:**

1. Tạo một file workflow mới (ví dụ: .github/workflows/reusable\_python\_tests.yml) trong kho của bạn.
2. Sử dụng từ khóa on: workflow\_call để biến nó thành một workflow có thể gọi được.
3. Bạn có thể định nghĩa inputs và outputs để truyền dữ liệu vào và ra khỏi workflow này.

## 2. Flask-Admin

**Cài đặt Flask và Flask-Admin**

**Flask** là một micro-framework web nhẹ cho Python. Nó rất linh hoạt, cho phép các nhà phát triển xây dựng các ứng dụng web từ đơn giản đến phức tạp mà không bị ràng buộc bởi quá nhiều quy ước.

**Flask-Admin** là một tiện ích mở rộng của Flask, cung cấp một giao diện quản trị (admin interface) mạnh mẽ và dễ sử dụng để quản lý dữ liệu trong ứng dụng Flask của bạn. Nó cho phép bạn tạo, đọc, cập nhật, xóa (CRUD) các bản ghi trong cơ sở dữ liệu một cách nhanh chóng mà không cần viết nhiều mã giao diện.

**Các bước cài đặt:**

1. **Tạo môi trường ảo (khuyến nghị):**
2. python -m venv venv
3. source venv/bin/activate # Trên Linux/macOS
4. # venv\Scripts\activate # Trên Windows
5. **Cài đặt Flask và Flask-Admin:**
6. pip install Flask Flask-Admin Flask-SQLAlchemy

# Flask-SQLAlchemy để làm việc với cơ sở dữ liệu

**Thiết kế giao diện quản trị với Flask-Admin: danh sách, thêm, sửa, xóa bảng dữ liệu**

Sau khi cài đặt, bạn có thể tích hợp Flask-Admin vào ứng dụng Flask của mình. Flask-Admin hoạt động tốt với nhiều ORM (Object-Relational Mapper) như SQLAlchemy, Peewee, hoặc có thể tùy chỉnh để làm việc với các nguồn dữ liệu khác.

## 3. Triển khai Flask-Admin trên render và test CI/CD

**Bước 1: Tạo folder flask\_admin\_CI-CD**

Tạo môi trường ảo và kích hoạt venv

* python -m venv venv
* venv\Scripts\activate

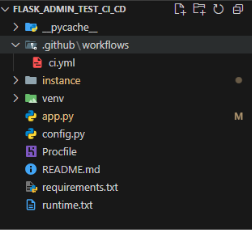
Cài thư viện để hỗi trợ chạy flask-admin

* pip install flask flask-admin flask-sqlalchemy gunicorn

Tạo file requirements.txt bằng câu lệnh:

* pip freeze > requirements.txt

**Dưới đây là cấu trúc project:**



**Bước 2: Tạo các file app.py,, Procfile, runtime.txt**

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect.

**Bước 3: Khởi tạo Database**

**Chạy câu lệnh bằng Python:**

from app import app, db

with app.app\_context():

db.create\_all()

exit()

**A screen shot of a computer

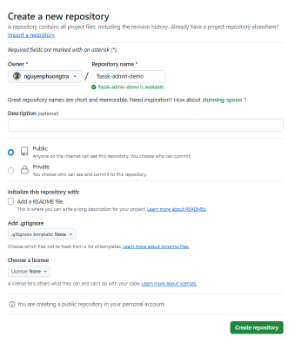
AI-generated content may be incorrect.**

**Sau khi chạy sẽ tạo ra một file database**

**A black rectangle with white text

AI-generated content may be incorrect.**

**Bước 4: Tạo pository trên github**

****

**Bước 5: Khởi tạo Git và đẩy lên GitHub**

**A close-up of a computer code

AI-generated content may be incorrect.**

**Bước 6: Vào tab Actions trên GitHub**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Bước 7: Triển khai lên Render**

**Truy cập Render.com, chọn New > Web Service.**

**Kết nối repo flask-admin-demo.**

**A red line drawn on a white background

AI-generated content may be incorrect.**

**Bước 8: Cấu hình**

* **Name: flask-admin-demo**
* **Branch: main**
* **Runtime: Python 3**
* **Build Command: pip install -r requirements.txt**
* **Start Command: gunicorn app:app**
* **Plan: Free**

**Bước 9:Nhấn Create Web Service, chờ deploy**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Sau khi build xong truy cập vào đường dẫn để xem kết quả:**

[**https://flask-admin-demo-b6st.onrender.com**](https://flask-admin-demo-b6st.onrender.com)

**Kết quả:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

Khi bạn chạy ứng dụng và truy cập /admin, bạn sẽ thấy một giao diện quản trị đầy đủ chức năng cho phép bạn tương tác với dữ liệu của mình.

**Quản lý vai trò, xác thực người dùng trong giao diện**

Mặc định, Flask-Admin không tích hợp sẵn hệ thống xác thực người dùng. Để bảo vệ giao diện quản trị, bạn cần tự triển khai hoặc sử dụng các tiện ích mở rộng xác thực của Flask (ví dụ: Flask-Login, Flask-Security).

**Cách tiếp cận cơ bản để bảo vệ Flask-Admin:**

Bạn có thể tạo một lớp ModelView tùy chỉnh và ghi đè phương thức is\_accessible() để kiểm tra xem người dùng hiện tại có quyền truy cập hay không.

Với cấu hình này, chỉ những người dùng đã đăng nhập và có trường is\_admin là True mới có thể truy cập vào giao diện quản trị của Flask-Admin. Các phần khác của ứng dụng Flask vẫn có thể truy cập công khai hoặc được bảo vệ bằng các quy tắc xác thực khác.

Báo cáo này đã bao gồm các phần chính bạn yêu cầu, từ lý thuyết DevOps đến các công cụ CI/CD, chất lượng mã nguồn và cách sử dụng Flask-Admin để xây dựng giao diện quản trị. Hy vọng nó sẽ hữu ích cho bạn!

# KẾT LUẬN

Báo cáo này đã đi sâu vào **Phát triển kiểm thử liên tục**, một trụ cột quan trọng trong quy trình phát triển phần mềm hiện đại. Chúng ta đã bắt đầu với **DevOps** – một triết lý kết hợp chặt chẽ Phát triển và Vận hành, nhằm phá vỡ rào cản và thúc đẩy hợp tác. Mô hình "Infinity Loop" minh họa rõ nét cách DevOps tạo nên văn hóa tự động hóa, tinh gọn, đo lường và chia sẻ kiến thức.

Trọng tâm của báo cáo là mối quan hệ cộng sinh giữa DevOps và **CI/CD (Tích hợp liên tục/Phân phối liên tục)**. Chúng ta đã khám phá cách CI/CD hình thành xương sống của quá trình phát triển hiệu quả, cho phép phản hồi nhanh chóng và đảm bảo mã nguồn luôn sẵn sàng để triển khai. Các công cụ như Jenkins, GitLab CI/CD và GitHub Actions đã được đề cập như những giải pháp triển khai tự động mạnh mẽ.

Bên cạnh đó, tầm quan trọng của **chất lượng mã nguồn** cũng được nhấn mạnh thông qua việc sử dụng pytest cho kiểm thử đơn vị, coverage.py để đo độ bao phủ mã, cùng với flake8 và pylint cho phân tích tĩnh. Các chỉ số như **Maintainability Index** và **Cyclomatic Complexity** giúp đánh giá và cải thiện chất lượng mã.

Chúng ta cũng đã thực hành thiết lập một quy trình CI trên **GitHub Actions**, tích hợp các bước xây dựng, kiểm thử và đóng gói với **Docker**. Đồng thời, việc cấu hình **Phân phối liên tục** để tự động triển khai ứng dụng lên các môi trường như Render cũng đã được trình bày, cho thấy sức mạnh của việc kết hợp CI/CD với Docker để đảm bảo tính nhất quán.

Cuối cùng, báo cáo đã giới thiệu các kỹ thuật **tối ưu hóa CI/CD nâng cao** như chia nhiều giai đoạn, chạy song song các job và sử dụng bộ nhớ đệm (caching). **Flask-Admin** cũng được đưa vào như một giải pháp hiệu quả để xây dựng giao diện quản trị, và quá trình triển khai nó với CI/CD đã được minh họa.

Tóm lại, việc áp dụng các nguyên tắc và thực hành này – từ những thay đổi văn hóa của DevOps đến việc triển khai kỹ thuật các pipeline CI/CD, kiểm thử mạnh mẽ và đảm bảo chất lượng – là yếu tố then chốt giúp các tổ chức đẩy nhanh tốc độ phát triển phần mềm, nâng cao chất lượng sản phẩm và duy trì lợi thế cạnh tranh. Bằng cách tích hợp, kiểm thử và phân phối liên tục, các nhóm có thể đạt được sự linh hoạt cao hơn, giảm thiểu rủi ro và cuối cùng là mang lại giá trị lớn hơn cho người dùng.