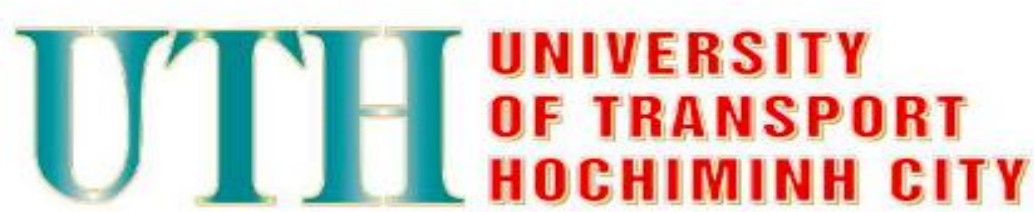
BỘ XÂY DỰNG

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP.HCM

VIỆN ĐÀO TẠO CHẤT LƯỢNG CAO



TÀI LIỆU SRS MÔN THỰC TẬP TỐT NGHIỆP

Tên đề tài: NỀN TẢNG PAAS MINI TÍCH HỢP AI PHÂN TÍCH LỖI

Phiên bản: 1.0

Giảng viên hướng dẫn:	**Thạc sĩ Nguyễn Văn Chiến**
Sinh viên thực hiện: *Nguyễn Vương Minh Khôi*	**MSSV:** 22H1120108
Lớp: CN22CLCD	**010412600018**
GitHub:	**https://github.com/Khoi1909/NexusDeploy**

2025

GIỚI THIỆU MÔN HỌC

MỤC LỤC

MỤC LỤC HÌNH ẢNH

MỤC LỤC BẢNG BIỂU

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

TPHCM, ngày tháng năm 2025

Ký và ghi rõ họ tên

Nguyễn Văn Chiến

CHƯƠNG 1. GIỚI THIỆU

1.1. Mục đích

Tài liệu này (SRS) đặc tả các yêu cầu về chức năng và phi chức năng cho dự án **Nexus Deploy**. Mục đích của tài liệu là cung cấp một mô tả chi tiết, rõ ràng và nhất quán về sản phẩm phần mềm sẽ được xây dựng.

Tài liệu này sẽ là cơ sở để:

- Định hướng quá trình thiết kế, lập trình và kiểm thử.
- Làm tài liệu tham chiếu và đánh giá cho giáo viên hướng dẫn.
- Thiết lập sự hiểu biết chung về các yêu cầu của dự án giữa các bên liên quan.

1.2. Phạm vi dự án

Nexus Deploy là một nền tảng Nền tảng như một Dịch vụ (PaaS - Platform-as-a-Service) hoàn chỉnh, được thiết kế để đơn giản hóa tối đa quy trình triển khai ứng dụng web cho lập trình viên.

Phạm vi của dự án bao gồm các chức năng chính sau:

1. **Tích hợp GitHub:** Người dùng đăng nhập vào hệ thống bằng tài khoản GitHub (thông qua OAuth) và chọn các kho mã nguồn (repository) của họ để triển khai.
2. **Tích hợp liên tục (CI):** Khi người dùng git push lên kho mã nguồn đã chọn, hệ thống sẽ tự động kích hoạt một quy trình (pipeline) build và test trong một môi trường Docker cô lập.
3. **Hỗ trợ AI Phân tích Lỗi:** Nếu quy trình CI thất bại (build hoặc test lỗi), hệ thống sẽ cung cấp tính năng "Tell me why". Khi được kích hoạt, một Mô hình Ngôn ngữ Lớn (LLM) sẽ phân tích log lỗi và đưa ra gợi ý khắc phục.
4. **Triển khai liên tục (CD) & Hosting:** Nếu quy trình CI thành công, hệ thống sẽ tự động build mã nguồn thành một Docker image, lưu trữ image và triển khai (host) ứng dụng dưới dạng một container.
5. **Cung cấp Tên miền tự động:** Mỗi ứng dụng được triển khai thành công sẽ được tự động gán một tên miền con công khai (ví dụ: my-app.khqi.io.vn) với chứng chỉ SSL (HTTPS) hợp lệ.

Dự án này **không** bao gồm việc cung cấp dịch vụ cơ sở dữ liệu (Database-as-a-Service). Người dùng được yêu cầu kết nối đến các dịch vụ CSDL bên ngoài.

1.3. Đối tượng dự kiến

Tài liệu này dành cho các đối tượng sau:

- **Giáo viên hướng dẫn:** Để theo dõi, đánh giá và góp ý về phạm vi và tiến độ của đồ án.
- **Người phát triển (Developer):** Là người trực tiếp xây dựng dự án, dùng tài liệu này làm kim chỉ nam kỹ thuật.
- **Người kiểm thử (Tester):** Để thiết kế các kịch bản kiểm thử (test cases) dựa trên các yêu cầu chức năng.

1.4. Định nghĩa, Thuật ngữ và Viết tắt

- **PaaS** (Platform-as-a-Service): Nền tảng như một Dịch vụ.
- **CI** (Continuous Integration): Tích hợp liên tục.
- **CD** (Continuous Deployment): Triển khai liên tục.
- **SRS** (Software Requirements Specification): Đặc tả Yêu cầu Phần mềm.
- **LLM** (Large Language Model): Mô hình Ngôn ngữ Lớn (ví dụ: Gemini, GPT).
- **OAuth** (Open Authorization): Giao thức ủy quyền mở, dùng để đăng nhập bằng GitHub.
- **Runner:** Tác nhân (worker) do Nexus Deploy lập trình, có nhiệm vụ thực thi các tác vụ CI/CD (build, test).
- **Traefik:** Một Reverse Proxy (Proxy ngược) hiện đại, dùng để tự động định tuyến tên miền và quản lý SSL.
- **Preset:** Một cấu hình định sẵn (ví dụ: Node.js, Python) mà người dùng chọn để hệ thống biết cách build và chạy dự án.
- **Secret:** Các biến môi trường bí mật (API keys, tokens) được mã hóa và lưu trữ.

1.5. Tổng quan tài liệu

Tài liệu này được tổ chức thành các chương:

- **Chương 1 (Giới thiệu):** Cung cấp cái nhìn tổng quan, phạm vi, định nghĩa và mục đích của dự án.
- **Chương 2 (Mô tả tổng quan):** Mô tả các tác nhân (actors) liên quan, các giả định, ràng buộc, và kiến trúc hệ thống cấp cao.
- **Chương 3 (Yêu cầu chức năng):** Đặc tả chi tiết các tính năng mà hệ thống phải thực hiện.
- **Chương 4 (Yêu cầu phi chức năng):** Đặc tả các yêu cầu về hiệu năng, bảo mật, độ tin cậy và các ràng buộc kỹ thuật khác.

CHƯƠNG 2. MÔ TẢ TỔNG QUAN

2.1. Bối cảnh sản phẩm (Product Perspective)

Nexus Deploy là một hệ thống web độc lập, hoạt động như một nền tảng PaaS hoàn chỉnh. Nó được thiết kế để tự động hóa toàn bộ quy trình từ mã nguồn (git push) đến một ứng dụng web đang chạy (HTTPS URL).

Hệ thống sẽ tương tác và phụ thuộc vào các thành phần bên ngoài sau:

- **GitHub:** Dùng làm nhà cung cấp xác thực (OAuth) và nguồn mã nguồn. Nexus Deploy sử dụng API của GitHub để liệt kê kho (repo) và tự động cài đặt Webhook.
- **Traefik:** Đóng vai trò là Reverse Proxy (Proxy ngược) ở tầng biên, chịu trách nhiệm định tuyến các tên miền con (subdomain) đến đúng container ứng dụng và tự động quản lý chứng chỉ SSL.
- **Docker Engine:** Môi trường thực thi lõi. Nexus Deploy sử dụng Docker SDK để tạo, quản lý và hủy các container một cách cô lập cho cả quá trình CI (build/test) và CD (host).
- **LLM API (External):** Một dịch vụ Mô hình Ngôn ngữ Lớn bên ngoài (như Gemini hoặc GPT) được gọi để thực hiện chức năng phân tích log lỗi.

2.2. Tác nhân và Người dùng (Actors and Users)

Hệ thống có hai tác nhân chính:

Tác nhân	Mô tả
Developer (Lập trình viên)	Là người dùng cuối của Nexus Deploy. Họ muốn triển khai ứng dụng của mình một cách nhanh chóng.
GitHub (Hệ thống)	Là hệ thống bên ngoài, có nhiệm vụ thông báo cho Nexus Deploy mỗi khi có sự kiện git push mới.

Bảng 1: Các tác nhân chính

2.3. Sơ đồ Use Case tổng quan

Sơ đồ này mô tả các tương tác cấp cao của tác nhân "Developer" với hệ thống Nexus Deploy.

Hình 1: Usecase Login with GitHub

Hình 2: Usecase Project Management

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 3: Usecase Deploy History

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 4: Usecase Log (Build & Runtime)

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 5: Usecase Secrets

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 6: Usecase Ask AI to Analyze

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 7: Usecase Project Manage (Restart/Stop)

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 8: Usecase Run CI/CD

USE CASE TRONG QUÁ TRÌNH HOÀN THIỆN

Hình 9: Usecase Manage Subscription Plan

2.4. Các giả định (Assumptions)

1. **Hạ tầng:** Hệ thống được giả định chạy trên một máy chủ (Server) đã được cài đặt sẵn Docker, Docker SDK (cho Python), và Traefik.
2. **DNS:** Tên miền chính (ví dụ: khqi.io.vn) và bản ghi Wildcard (*.khqi.io.vn) đã được trỏ chính xác đến IP của máy chủ.
3. **Kiến thức Người dùng:** Người dùng (Developer) có kiến thức cơ bản về Git và hiểu cách tổ chức một dự án web (ví dụ: biết các lệnh build/start).
4. **Phạm vi Ứng dụng:** Các dự án được triển khai là ứng dụng web dựa trên giao thức HTTP/HTTPS.
5. **Tài khoản:** Người dùng bắt buộc phải có tài khoản GitHub.

2.5. Các ràng buộc (Constraints)

1. **Công nghệ (Tech Stack):**
   - Backend và Runner phải được phát triển bằng Python (sử dụng FastAPI và Docker SDK).
   - Frontend phải được phát triển bằng React hoặc Vue.
   - Hàng đợi (Queue) phải sử dụng Redis.
2. **Hạ tầng (Infrastructure):**
   - Toàn bộ hệ thống (CI/CD, Hosting) phải chạy trên nền tảng Docker.
   - Hệ thống bắt buộc phụ thuộc vào Traefik để xử lý định tuyến và SSL.
3. **Bảo mật (Security):**
   - Tất cả các "Secrets" (biến môi trường bí mật) của người dùng phải được mã hóa (ví dụ: AES-256) trước khi lưu vào cơ sở dữ liệu.
   - Các container của người dùng phải được chạy trong một mạng Docker riêng (isolated network) và không có đặc quyền (non-privileged) để đảm bảo an toàn.

4. **Tài nguyên (Resources):**
    - Hệ thống phải có khả năng giới hạn tài nguyên (RAM, CPU) cho mỗi container ứng dụng của người dùng (dựa trên Gói đăng ký - Plan).
5. **API Bên ngoài (External APIs):**
    - Hệ thống phụ thuộc vào tính khả dụng của GitHub API (cho OAuth, Webhook) và LLM API (cho phân tích lỗi). Nếu các API này gặp sự cố, các tính năng liên quan sẽ bị gián đoạn.

CHƯƠNG 3. YÊU CẦU CHỨC NĂNG (FUNCTIONAL REQUIREMENTS)

Chương này mô tả chi tiết các yêu cầu chức năng của hệ thống Nexus Deploy. Mỗi yêu cầu được gán một mã định danh duy nhất (ví dụ: FR1.1) để dễ dàng theo dõi.

3.1. FR1: Quản lý Xác thực (Authentication)

Use Case liên quan: Đăng nhập bằng GitHub

ID	Yêu cầu	Mô tả chi tiết
FR1.1	Đăng nhập bằng GitHub	Hệ thống phải cung cấp nút "Login with GitHub". Khi nhấn vào, người dùng sẽ được chuyển hướng đến trang xác thực của GitHub (OAuth).
FR1.2	Xử lý Callback	Hệ thống phải có một API endpoint (`/auth/github/callback`) để nhận mã ủy quyền (authorization code) từ GitHub sau khi người dùng đồng ý.
FR1.3	Tạo phiên (Session)	Sau khi nhận code, backend phải trao đổi nó với GitHub để lấy `access_token`. Hệ thống phải mã hóa và lưu `access_token` này vào CSDL, gắn liền với tài khoản người dùng và tạo một phiên đăng nhập (ví dụ: JWT) cho người dùng trên Nexus Deploy.
FR1.4	Đăng xuất	Hệ thống phải cung cấp chức năng cho phép người dùng đăng xuất, khi đó phiên đăng nhập (JWT) sẽ bị vô hiệu hóa.
FR1.5	Đồng bộ thông tin	Khi đăng nhập lần đầu, hệ thống phải lấy và lưu thông tin cơ bản của người dùng từ GitHub (username, avatar, email) vào CSDL nội bộ.

Bảng 2: FR1 Quản lý xác thực

3.2. FR2: Quản lý Dự án (Project Management)

Use Case liên quan: Tạo / Quản lý Dự án

ID	Yêu cầu	Mô tả chi tiết
FR2.1	Thêm Dự án mới	Sau khi đăng nhập, người dùng có thể truy cập trang "Thêm Dự án".
FR2.2	Liệt kê Kho (Repo)	Hệ thống phải sử dụng access_token của người dùng để gọi GitHub API, lấy và hiển thị danh sách các kho mã nguồn (repository) mà người dùng có quyền truy cập.
FR2.3	Cài đặt Webhook	Khi người dùng chọn một kho để triển khai, hệ thống phải tự động dùng GitHub API để thêm một Webhook vào kho đó, trỏ về một API endpoint của Nexus Deploy (ví dụ: /api/webhook/github).
FR2.4	Cấu hình Preset	Khi thêm dự án, người dùng **bắt buộc** phải chọn một "Framework Preset" (Cài đặt định sẵn) từ danh sách do hệ thống cung cấp (ví dụ: Node.js, Python (FastAPI), Static Site).
FR2.5	Cấu hình Lệnh (UI)	Dựa trên Preset đã chọn, hệ thống sẽ tự động điền các lệnh mặc định vào các ô "Build Command" (Lệnh Build) và "Start Command" (Lệnh Khởi chạy). Người dùng có thể tùy chỉnh lại các lệnh này.
FR2.6	Cấu hình Port (Nâng cao)	Hệ thống sẽ tự động suy ra cổng (port) nội bộ dựa trên Preset. Tuy nhiên, hệ thống phải cung cấp một ô "Port" (trong cài đặt nâng cao) cho phép người dùng ghi đè (override) cổng nội bộ nếu muốn.
FR2.7	Xóa Dự án	Người dùng có thể xóa một dự án khỏi Nexus Deploy. Hành động này sẽ (1) dừng và xóa container đang chạy (nếu có), (2) xóa dự án và log khỏi CSDL, (3) tự động gỡ Webhook khỏi kho GitHub.

Bảng 3: FR2 Quản lý dự án

3.3. FR3: Quản lý Biến môi trường (Secrets)

Use Case liên quan: Cấu hình Biến môi trường (Secrets)

ID	Yêu cầu	Mô tả chi tiết
FR3.1	Giao diện CRUD	Hệ thống phải cung cấp giao diện cho phép người dùng (trong trang Cài đặt Dự án) Thêm, Sửa, Xóa các biến môi trường (Secrets). Mỗi secret bao gồm Name và Value.
FR3.2	Mã hóa tại Backend	Khi nhận được Value của một secret, backend **bắt buộc** phải sử dụng một khóa chủ (master key) của hệ thống để mã hóa (ví dụ: AES-256) giá trị này trước khi lưu vào CSDL.
FR3.3	Che giấu Giá trị	Trong CSDL và trên Giao diện UI, giá trị (Value) của secret không bao giờ được hiển thị đầy đủ (ví dụ: chỉ hiển thị my_secret = **********).
FR3.4	Tiêm (Inject) vào CI	Khi Runner thực thi **job CI (Build/Test)**, hệ thống phải giải mã và tiêm (inject) tất cả các secret của dự án vào môi trường build.
FR3.5	Tiêm (Inject) vào Host	Khi Runner triển khai **container Host**, hệ thống phải giải mã và tiêm tất cả các secret vào làm biến môi trường cho container đó.

Bảng 4: FR3 Quản lý biến môi trường

3.4. FR4: Quy trình Tích hợp & Triển khai (CI/CD Pipeline)

Use Case liên quan: Run CI/CD

ID	Yêu cầu	Mô tả chi tiết

FR4.1	Kích hoạt (Trigger)	Hệ thống phải có một API endpoint để nhận Webhook sự kiện git push từ GitHub. Endpoint này phải xác thực payload (dùng secret) để đảm bảo tính hợp lệ.
FR4.2	Tạo Tác vụ (Job)	Khi nhận được Webhook hợp lệ, hệ thống phải tạo một bản ghi "Deploy Job" mới trong CSDL với trạng thái **"Pending"** (Đang chờ) và đẩy Job ID vào hàng đợi (Redis Queue).
FR4.3	Nhận Tác vụ (Runner)	Dịch vụ **Runner** (tác nhân) phải liên tục lắng nghe hàng đợi Redis. Khi có job mới, Runner sẽ nhận và cập nhật trạng thái job trong CSDL thành **"Running"** (Đang chạy).
FR4.4	Giai đoạn CI (Build/Test)	Runner phải tạo một môi trường Docker cô lập (dựa trên image nền của Preset) để thực thi các lệnh "Build Command" và "Test Command" (lấy từ FR2.5). Các secret (FR3.4) phải được tiêm vào môi trường này.
FR4.5	Truyền Log (Build)	Trong suốt quá trình CI, Runner phải thu thập và gửi log (stdout/stderr) về Backend theo thời gian thực. Backend phải đẩy log này qua WebSocket để hiển thị trên UI.
FR4.6	Xử lý CI Thất bại	Nếu bất kỳ lệnh nào trong Giai đoạn CI thất bại (trả về non-zero exit code), Runner phải dừng quy trình, cập nhật trạng thái job thành **"Failed"**, và lưu trữ toàn bộ log lỗi vào CSDL.
FR4.7	Giai đoạn CD (Build Image)	Nếu Giai đoạn CI thành công, Runner phải tiến hành build Docker image cuối cùng (dựa trên Dockerfile của dự án hoặc một Dockerfile nền do Preset quy định).
FR4.8	Đẩy Image (Push)	Runner phải gắn thẻ (tag) image vừa build và đẩy (push) nó lên Docker Registry (Docker Hub) đã được cấu hình.

FR4.9	Giai đoạn CD (Host)	Sau khi push image thành công, Runner (hoặc một dịch vụ Deployer riêng) phải kích hoạt Giai đoạn Host (xem **FR6**).
FR4.10	Hoàn tất Tác vụ	Sau khi Giai đoạn Host thành công, Runner cập nhật trạng thái job thành **"Success"**.

Bảng 5: FR4 Quy trình Tích hợp & Triển khai

3.5. FR5: Phân tích Lỗi bằng AI

Use Case liên quan: Ask AI to Analyze

ID	Yêu cầu	Mô tả chi tiết
FR5.1	Kích hoạt (UI)	Đối với các job có trạng thái **"Failed"**, giao diện UI phải hiển thị một nút (ví dụ: "Tell me why") bên cạnh log lỗi.
FR5.2	Gửi Yêu cầu AI	Khi người dùng nhấn nút "Tell me why", Frontend sẽ gọi một API Backend.
FR5.3	Xử lý (Backend)	Backend sẽ truy xuất log lỗi đã lưu (từ FR4.6) của job đó, có thể tiền xử lý (ví dụ: cắt bớt, lọc nhiễu) để tối ưu.
FR5.4	Gọi LLM API	Backend gửi log lỗi đã xử lý đến API của LLM bên ngoài (ví dụ: Gemini) với một prompt (câu lệnh) được thiết kế để yêu cầu phân tích và gợi ý sửa lỗi.
FR5.5	Phân cấp Gói (Standard)	Đối với người dùng gói "Standard", hệ thống chỉ hiển thị gợi ý chung và giải thích lỗi.

FR5.6	Phân cấp Gói (Premium)	(Tham chiếu **FR7.3**): Đối với người dùng gói "Premium", hệ thống sẽ yêu cầu LLM đưa ra các đề xuất sửa lỗi chi tiết (có thể bao gồm đoạn code) và cho phép người dùng xem xét.
FR5.7	Hiển thị Kết quả	Kết quả phân tích từ LLM phải được trả về và hiển thị một cách rõ ràng, dễ đọc trên giao diện UI cho người dùng.

Bảng 6: FR5 Phân tích Lỗi bằng AI

3.6. FR6: Hosting & Quản lý Vòng đời

Use Case liên quan: Project Manage (Restart/Stop), Run CI/CD

ID	Yêu cầu	Mô tả chi tiết
FR6.1	Kích hoạt Triển khai	Giai đoạn này được kích hoạt sau khi FR4.8 (Push image) thành công.
FR6.2	Dọn dẹp Container cũ	Hệ thống phải dừng (stop) và xóa (remove) container cũ (nếu có) của dự án để chuẩn bị cho phiên bản mới.
FR6.3	Chạy Container mới	Hệ thống phải kéo (pull) image mới nhất (từ FR4.8) và chạy nó dưới dạng một container mới (`docker run`).
FR6.4	Tiêm Biến (Host)	Lệnh `docker run` phải tiêm (inject) tất cả các Secret của người dùng (FR3.5) và biến `PORT` (FR2.6 hoặc mặc định của Preset) vào container.
FR6.5	Gán Nhãn (Labels)	Lệnh `docker run` **bắt buộc** phải gán các "labels" (nhãn) cần thiết cho Traefik, bao gồm: `traefik.enable=true`, `Host(...)` (tên miền), `server.port` (cổng nội bộ), và `certresolver=letsencrypt` (để kích hoạt SSL).

FR6.6	Giới hạn Tài nguyên	Lệnh docker run phải áp đặt các giới hạn về tài nguyên (RAM, CPU) dựa trên Gói đăng ký của người dùng.
FR6.7	Xem Log (Runtime)	Hệ thống phải cung cấp một tab "Logs" cho các ứng dụng đang chạy. UI phải kết nối WebSocket để stream log trực tiếp từ container (thông qua docker logs -f).
FR6.8	Khởi động lại (Restart)	Hệ thống phải cung cấp một nút/API "Restart" để thực thi lệnh docker restart [container_name] mà không cần build lại.
FR6.9	Dừng (Stop)	Hệ thống phải cung cấp một nút/API "Stop" để thực thi docker stop và docker rm, giải phóng tài nguyên và gỡ bỏ ứng dụng khỏi tên miền (Traefik sẽ tự động xử lý).

Bảng 7: FR6 Hosting và Quản lý vòng đời

3.7. FR7: Quản lý Gói đăng ký & Phân quyền (Plan & Permission)

Use Case liên quan: Manage Subscription Plan

ID	Yêu cầu	Mô tả chi tiết
FR7.1	Định nghĩa Gói	Hệ thống phải định nghĩa ít nhất hai (2) cấp độ gói: "Standard" và "Premium".
FR7.2	Gói Mặc định	Mọi người dùng mới đăng ký qua GitHub (theo FR1.5) sẽ tự động được gán gói "Standard" làm mặc định.
FR7.3	Bảng Phân quyền	Hệ thống phải duy trì một logic (ví dụ: một cấu hình nội bộ) để định nghĩa và phân biệt quyền lợi/giới hạn của từng gói.

FR7.4	Thực thi Phân quyền	Hệ thống **bắt buộc** phải kiểm tra gói đăng ký của người dùng trước khi thực thi các hành động bị giới hạn: 1. **Trước khi chạy CI (FR4):** Kiểm tra số lượng build đồng thời. 2. **Khi gọi AI (FR5):** Kiểm tra cấp độ phân tích được phép. 3. **Khi deploy Host (FR6):** Áp dụng đúng giới hạn RAM/CPU.
FR7.5	Giao diện Nâng cấp	(Yêu cầu ở mức độ thấp) Hệ thống cần có một trang "Billing/Nâng cấp" trên Giao diện UI, mô tả sự khác biệt giữa các gói. (Việc tích hợp thanh toán thực tế nằm ngoài phạm vi cốt lõi của đồ án này).

Bảng 8: FR7 Quản lý gói đăng ký và phân quyền

CHƯƠNG 4. YÊU CẦU PHI CHỨC NĂNG (NON-FUNCTIONAL REQUIREMENTS)

Chương này đặc tả các yêu cầu phi chức năng, xác định các tiêu chuẩn chất lượng, hiệu suất, bảo mật và vận hành của hệ thống Nexus Deploy.

4.1. NFR1: Yêu cầu về Hiệu năng (Performance)

ID	Yêu cầu	Mô tả chi tiết
NFR1.1	Phản hồi Giao diện (UI)	Thời gian tải các trang chính (Dashboard, Project Settings) phải dưới 2 giây. Các tương tác API (ví dụ: lưu Secrets) phải phản hồi cho người dùng dưới 500ms.
NFR1.2	Kích hoạt Tác vụ (Job Trigger)	Thời gian từ lúc hệ thống nhận được Webhook git push (hợp lệ) đến lúc Job được đưa vào hàng đợi (Redis) và Runner bắt đầu nhận việc phải dưới 5 giây.
NFR1.3	Xử lý Đồng thời	Hệ thống (Runner pool) phải có khả năng xử lý đồng thời nhiều job CI/CD, tuân thủ theo giới hạn của gói đăng ký (tham chiếu **FR7.3**).
NFR1.4	Truyền Log (Real-time)	Log (cả build-time và run-time) phải được stream (luồng) đến giao diện người dùng qua WebSocket với độ trễ dưới 1 giây để đảm bảo trải nghiệm theo dõi thời gian thực.

Bảng 9: NFR1: Yêu cầu về Hiệu năng

4.2. NFR2: Yêu cầu về Bảo mật (Security)

ID	Yêu cầu	Mô tả chi tiết
NFR2.1	Mã hóa Dữ liệu Nhạy cảm	Tất cả các access_token của GitHub và Value của Secrets (tham chiếu **FR3.2**) **bắt buộc** phải được mã hóa (ví dụ: AES-256-GCM) trước khi lưu trữ vào cơ sở dữ liệu.
NFR2.2	Cách ly Container	Các container ứng dụng của người dùng (cả CI và Host) **bắt buộc** phải chạy trong một mạng Docker riêng (isolated network) và không có đặc quyền (--privileged=false) để ngăn chặn leo thang đặc quyền (privilege escalation) vào máy chủ (tham chiếu **Ràng buộc 2.5**).
NFR2.3	Xác thực Webhook	Endpoint nhận Webhook từ GitHub (tham chiếu **FR4.1**) **bắt buộc** phải xác thực chữ ký (signature) của payload (sử dụng một "secret" được chia sẻ) để đảm bảo yêu cầu thực sự đến từ GitHub và không bị giả mạo.
NFR2.4	Bảo mật Web	Giao diện Web (Frontend) và API (Backend) phải được bảo vệ khỏi các lỗ hổng web phổ biến (OWASP Top 10), bao gồm: XSS (Cross-Site Scripting) và CSRF (Cross-Site Request Forgery).

Bảng 10: NFR2 Yêu cầu về Bảo mật

4.3. NFR3: Yêu cầu về Tính Sẵn sàng & Độ tin cậy (Availability & Reliability)

ID	Yêu cầu	Mô tả chi tiết
NFR3.1	Tính Sẵn sàng (Uptime)	Trang Dashboard và các ứng dụng đang được host của người dùng phải có tính sẵn sàng cao (mục tiêu 99.9%). Traefik phải được cấu hình để tự động khởi động lại.
NFR3.2	Xử lý Lỗi Bên ngoài	Nếu các API bên ngoài (GitHub, LLM) không khả dụng hoặc trả lỗi (ví dụ: 500, 429), hệ thống Nexus Deploy không được sập (crash). Thay vào đó, hệ thống phải ghi nhận lỗi và hiển thị thông báo thân thiện cho người dùng.
NFR3.3	Khôi phục Runner	Nếu một tác vụ (Runner) đang chạy dở (Running) bị sập (crash), tác vụ đó sẽ được đánh dấu là **"Failed"**. Hệ thống **sẽ không** tự động thử lại (để tránh các vòng lặp build lỗi). Người dùng phải tự kích hoạt lại quy trình (ví dụ: bằng cách push một commit mới hoặc nhấn nút "Re-deploy" trên UI).

Bảng 11: NFR3 Yêu cầu vè tính sẵn sàng và độ tin cậy

4.4. NFR4: Yêu cầu về Tính Dễ sử dụng (Usability)

ID	Yêu cầu	Mô tả chi tiết
NFR4.1	Luồng Triển khai	Luồng triển khai một dự án mới (từ lúc chọn repo đến lúc có link) phải trực quan, rõ ràng và không yêu cầu người dùng cấu hình các thông số kỹ thuật phức tạp (như port, trừ khi ở chế độ nâng cao - **FR2.6**).
NFR4.2	Trạng thái Hệ thống	Giao diện phải luôn hiển thị rõ ràng trạng thái của dự án (Pending, Running, Success, Failed) và trạng thái của ứng dụng (Đang chạy, Đã dừng).
NFR4.3	Phản hồi Lỗi (AI)	Khi build thất bại, hệ thống phải cung cấp khả năng phân tích lỗi (tham chiếu **FR5**), giúp người dùng (đặc biệt là người mới) chẩn đoán sự cố dễ dàng hơn thay vì chỉ đọc log thô.

Bảng 12: NFR4 Yêu cầu về tính dễ sử dụng

4.5. NFR5: Yêu cầu về Giới hạn Tài nguyên (Resource Constraints)

ID	Yêu cầu	Mô tả chi tiết
NFR5.1	Thực thi Giới hạn	Hệ thống **bắt buộc** phải thực thi chính xác các giới hạn về tài nguyên (RAM, CPU) cho các container host (tham chiếu **FR6.6**) và các giới hạn nghiệp vụ (số dự án, build đồng thời) dựa trên Gói đăng ký của người dùng (tham chiếu **FR7**).

Bảng 13: NFR5 Yêu cầu về giới hạn tài nguyên